Revisiting strong Gröbner bases over Euclidean domains

Daniel Lichtblau
August 22, 2003
Wolfram Research, Inc.
100 Trade Centre Dr
Champaign, IL 61820
danl@wolfram.com

Abstract: Buchberger and Kandri−Rody and Kapur defined a strong Gröbner basis for a polynomial ideal over a Euclidean domain in a way that gives rise to canonical reductions. This retains what is perhaps the most important property of Gröbner bases over fields. A difficulty is that these can be substantially harder to compute than their field counterparts. We extend their results for computing these bases to give an algorithm that is effective in practice. In particular we show how to use S−polynomials (rather than "critical pairs") so that the algorithm becomes quite similar to that for fields, and thus known strategies for the latter may be employed. We also show how Buchberger's important criteria for detection of unneeded S−polynomials can be extended to work over a Euclidean domain. We then provide simple examples as well as applications to solving equations in quotient rings, Hensel lifting, Hermite normal form computations, and reduction of univariate polynomial lattices. These serve to demonstrate why Gröbner basis computations over such rings are indeed worthy of consideration.

Keywords and phrases: Gröbner basis, Euclidean domain, Hensel lifting, Hermite normal form, linear diophantine systems, lattice reduction.

1 Introduction

Since their introduction by Bruno Buchberger in the 1960's, the theory and application of Gröbner bases has been developed extensively. While the original version worked with polynomial rings defined over fields, this has been extended in different ways to other types of base ring such as Euclidean domains or principal ideal domains. Text−book expositions for this may be found in [Becker, Weispfenning, and Kredel 1993] and [Adams and Loustaunau 1994]. As might be expected, the less structured the base ring, the more problematic becomes the theory behind and/or computation of such bases. Moreover while the definitions for the field case are common throughout the literature, one encounters variations when working over other rings, motivated by the wish (or decreasing ability) to preserve various aspects of the field case. One particular variant, proposed independently in [Kandri−Rody and Kapur 1984], and [Buchberger 1985, section 8], defines what is termed a "strong" Gröbner basis over the integers. As demonstrated in [Kandri−Rody and Kapur 1988] this extends more generally to polynomial rings over Euclidean domains. Their motivation was to define these bases in such a way that canonical reductions to normal form are essentially unchanged from the field case. A by−product was that one also retains similarity to the field case in the algorithm for computing these. As a generalization, [Pan 1989] developed similar ideas but in the setting of polyno−mial rings over effectively computable principal ideal domains. In this paper we restrict attention to Euclidean domains firstly because that is the setting wherein one may preserve the notion of canonical forms, and secondly in order to avoid questions of computability.

There are at least two reasons to want a strong Gröbner basis over a Euclidean domain. One is that as noted above we obtain canonical forms, and these are very useful in computations modulo polynomial ideals. The second is that reduction is now cheap; with a weak Gröbner basis one must compute greatest−common−divisors in the base ring in order to perform reduction (see [Becker, Weispfenning, and Kredel 1993]), whereas with a strong basis one only needs use a division algorithm (the price of course is that the basis computation itself may be more costly).

The theory behind strong bases was largely resolved by the early 1990's but details regarding efficient computation and preservation of the simplicity of Buchberger's algorithm are scattered through the references. The intent of this paper is to gather this under one roof, so to speak, and to make explicit mention of any improvements and simplifica−tions of which we are aware. For example, a straightforward reduction algorithm can be slow as it essentially emulates the extended GCD algorithm but with coefficient arithmetic carried over to polynomials. To remedy this

[Kandri−Rody and Kapur 1984; Kandri−Rody and Kapur 1988] use the extended GCD explicitly on coefficients, but make use of two types of S−polynomial and require a restriction on how reduction may be performed. Another issue is that with the notable exception of [Möller 1988] the literature says relatively little about extending the Buchberger criteria for eliminating redundant S−polynomials ([Adams and Loustaunau 1994] also considered these criteria in some exercises, but in the context of what appears to be a very different algorithm for working over a PID). We give a form that is very much in the spirit of the field case in [Buchberger 1985]. We also give several useful applications of special cases of such bases.

It will turn out that our basis is identical to the D−Gröbner basis discussed in chapter 10 of [Becker, Weispfenning, and Kredel 1993] (they refer to the two types of polynomial as S−polynomials and G−polynomials). But we will have a cheaper way to compute this basis because of more general reduction and the availability of redundancy criteria, as well as fewer S−polynomials to consider. Thus we believe this algorithm demonstrates a reasonable blend of ease of implementation and runtime efficiency. The algorithm we will discuss is implemented in the kernel of *Mathematica* [Wolfram 1999] (*Mathematica* (TM) is a registered trademark of Wolfram Research, Incorporated). General information about Gröbner bases in *Mathematica* may be found in [Lichtblau 1996].

The outline of this paper is as follows. First we cover the basic definitions, when working in a polynomial ideal over the integers, of term ordering, canonical rewriting, S−polynomials, and Gröbner bases. We then extend the theory presented in [Buchberger 1985] and [Kandri−Rody and Kapur 1988] so that it is more like the case where one works over a field. We next extend the well−known Buchberger criteria for detecting unnecessary S−polynomials in advance of performing actual (and often time consuming) reductions. We follow with several general examples. We then show some specialized applications that, among other things, connect these bases to important areas elsewhere in computational mathematics.

In the sequel we restrict our attention almost exclusively to the integers for clarity of exposition. It should be noted that definitions and theorems in this paper extend readily to all Euclidean domains over which one can perform effective computation, provided one can canonically select elements in a way that will be made precise for the integer case below. It is then straightforward to adapt the ideas behind this case to the other common Euclidean rings e.g. Gaussian integers or univariate polynomials.

2 Notation and definitions

First we establish notation. We work in the polynomial ring of n indeterminates over the integers, $\mathbb{Z}[x_1, \ldots, x_n]$. A power product is a product of the form $\prod_{j=1}^{n} (x_j)^{e_j}$. A term, or monomial, is a power product times an integer coeffi− cient (note that some authors define one or the other of these to be what we call a power product). We will typically denote monomials as $c_j t_j$ where c_j is an integer coefficient and t_j is a power product. One sees immediately that any polynomial in our ring can be written as a sum of terms with distinct power products; this is the usual definition of a polynomial in expanded form. Our typical usage of letters (possibly subscripted or otherwise annotated) in the sequel will be as follows: $\{a, b, c, d, e\}$ are coefficients in our ring, $\{f, g, h, p, q, r\}$ are polynomials, $\{i, j, k, m, n\}$ are integers, and $\{s, t, u, v\}$ are power products.

As in the field case we define well−founded orderings on the power products. Let $\{j_1, \ldots j_n\}$ denote the (ordered) exponent vector of nonnegative integers for a given power product (that is, j_1 is the exponent of x_1, etc.). Suppose u, v, and w are any three such exponent vectors, 0 is the exponent vector consisting of all zeros, and sums of expo− nent vectors are of course performed element−wise and correspond to products of power products.

Definition 1: A total ordering among such exponent vectors (and hence among power products) is well founded provided

(i) $0 < u$ for non−zero u
(ii) $u \leq v \Longleftrightarrow u + w \leq v + w$

For example we have the oft−used "pure lexicographic" ordering wherein $\{j_1, \ldots j_n\} > \{m_1, \ldots m_n\}$ whenever

$j_i = m_i$ for all $1 \le i < k \le n$ and $j_k > m_k$. For naming purposes we will sometimes call a term ordering T. In the sequel when power products are compared it is always assumed that this is done with respect to a well founded order.

Definition 2: We will regard our polynomials as sums of terms in descending term order. That is, if $p = c_1\ t_1 + \ldots + c_n\, t_n$ then we have $t_1 > t_2 > \ldots > t_n$ (of course this depends on the particular choice of term order). The term $c_1\ t_1$ is denoted the "head" term. In the language of rewriting rules one says that the head term "reduces" to minus the sum of the remaining terms. For a pair of power products $v = \{k_1, \ldots k_n\}$ and $w = \{m_1, \ldots m_n\}$ we say that w divides v if $m_j \le k_j$ for all $1 \le j \le n$. For abbreviation purposes we will write $\mathrm{HPP}[p] = t_1$, ("PP" for "power product"), $\mathrm{HCoeff}[p] = c_1$, and $\mathrm{HMonom}[p] = c_1\ t_1$.

We will assume the reader is familiar with the basic ideas of Gröbner bases in polynomial rings over fields. Good references for this include [Buchberger 1985; Cox, Little, and O’Shea 1992; Becker, Weispfenning, and Kredel 1993; Adams and Loustaunau 1994]. Recall that one of several equivalent definitions is that one obtains a canonical form when reducing a given polynomial by such a basis. The various definitions are no longer equivalent when one works over a more general ring, and it is this particular one that gives rise to strong Gröbner bases when the base ring is a Euclidean domain. Before this can be described we must first see what is meant by reduction, as this is altered from the field case.

First we will impose an ordering on elements in the coefficient ring. For our later purposes this too will be a total ordering, which we will denote by <<. In particular, suppose our Euclidean norm on an element c in the ring is denoted by $|\,c\,|$. Then whenever $\left|c_1\right| < \left|c_2\right|$ we require that $c_1 << c_2$. For integers we could for example use absolute values with ties broken by sign. So, following [Kandri−Rody and Kapur 1984; Buchberger 1985, section 8], we may take for our ordering

$$0 << 1 << -1 << 2 << -2 << \ldots$$

As will become clear, what we really require is a way to obtain unique minimal remainders in the division algorithm. This extra ordering suffices for that task.

Definition 3: Given a monomial $m = c\,t$ and a polynomial $p = \sum c_j\, t_j$ with t_1 the leading power product we say that p reduces m provided

(i) $t_1 \mid t$ (that is, we have $t = s_1\ t_1$).

(ii) Using the division algorithm to write $c = a\,c_1 + d$, we have $a \ne 0$ (or, equivalently, $|d| < |c|$). In this case we write $m \to m - a\,s_1\ p$. More generally we may allow any multiplier a such that the remainder satisfies $|d| < |c|$, but the quotient a from the division algorithm is the only one we use in actual practice.

Similarly if q and p are polynomials we say p reduces q provided it reduces some monomial m of q. Note that reduction depends on term order in general. We make this explicit in a shorthand notation: if the resulting polyno−mial is r we write $q \xrightarrow{\{p,\ T\}} r$. Generally we will be interested in head term reductions, but for purposes of obtaining canonical forms we will reduce lower terms as well. Note that it is in reductions that minimal remainders become important: we require that the reduced polynomial be "smaller" either in head power product or coefficient. There is a small subtlety that should be made explicit. Our division algorithm must work in such a way that the quotient of 2 and 3 is 1, with a remainder of -1 (because -1 is smaller than 2 in the Euclidean norm).

Definition 4: Given a polynomial q and a set of polynomials F we say that q is reducible by F if there is a polyno−mial p in F that reduces q. There may be many such, and one may get different reductions. The point of a Gröbner basis is that we will get a unique result once no further reductions can be applied, regardless of choices for reducing polynomials that were made along the way. If some chain of reductions from q leads to a polynomial r (regardless of whether it might be further reduced by F), we write $q \xrightarrow{\{F,\ T\}} r$.

We now mention why this form of reduction is useful. As we will see, one can obtain a basis computation algorithm that is quite similar to that for fields. This is quite important if one is to write (almost) generic code that is at the same time optimized for different coefficient domains. Indeed, we want to use heuristics that are borrowed from the field case to the greatest extent possible, and the fewer departures from that case the more readily we are able to do this. This may also be carried beyond the Buchberger algorithm. Specifically we note that there has been much work over the years to do Gröbner basis conversion. One such method in particular, the Gröbner walk [Collart, Kalkbren−ner, and Mall 1997], appears to be extendable to Euclidean domain base rings. Yet another reason to have this form of reduction is that it is fast; weak bases rely on slower GCD computations rather than division.

We define two types of S−polynomial. Recall that the idea behind these in the field case is to combine head terms using the LCM of the lead power products, and then kill off the lead coefficient. In the Euclidean domain case we can do this only if one lead coefficient divides the other, or if we will allow coefficient multipliers that are both nonunits. Moreover we must allow for reducing rather than entirely removing head coefficients. For example, the pair $\{2x, 3y\}$ will, in contrast to the field case, give rise to the S−polynomial xy. While two flavors of S−polynomial marks a departure from the field case, we will see later how these may be used in an algorithm that is virtually identical to Buchberger's.

Definition 5 (S−polynomials): We are given polynomials $p_j = c_j t_j + r_j$ where $t_j = \mathrm{HPP}[p_j]$ for $j \in \{1, 2\}$. Without loss of generality we may assume $|c_1| \leq |c_2|$. Let

$$\{c, \{a_1, a_2\}\} = \mathrm{ExtendedGCD}[c_1, c_2]$$

(that is, c is the GCD with $c = a_1 c_1 + a_2 c_2$). Let $t = \mathrm{PolynomialLCM}[t_1, t_2]$ with cofactors s_1 and s_2 so that $t = s_1 t_1 = s_2 t_2$. Finally take $d = \mathrm{LCM}[c_1, c_2]$ with cofactors b_1 and b_2 so that $d = b_1 c_1 = b_2 c_2$. With this we define two types of S−polynomial:

$$\mathrm{Spoly}_1[p_1, p_2] = a_1 s_1 p_1 + a_2 s_2 p_2$$

$$\mathrm{Spoly}_2[p_1, p_2] = b_1 s_1 p_1 - b_2 s_2 p_2$$

Note that the head term of $\mathrm{SPoly}_1[p_1, p_2]$ had coefficient c and power product t, and in $\mathrm{SPoly}_2[p_1, p_2]$ we have killed off that power product. Also note that when c_1 divides c_2 then $\mathrm{SPoly}_1[p_1, p_2]$ is simply a power product multiple of p_1 (because $a_1 = 1$ and $a_2 = 0$). In this case it will obviously reduce to zero, and only $\mathrm{SPoly}_2[p_1, p_2]$ will be of interest. Finally note that due to choices of cofactor, SPoly_1 is not uniquely defined; this will not matter for our purposes and we merely require that an extended gcd algorithm exist. Anticipating later results, we now define, for each pair, a unique S−polynomial.

Definition 6: Again given polynomials $p_j = c_j t_j + r_j$ with $t_j = \mathrm{HPP}[p_j]$ for $j \in \{1, 2\}$ and $|c_1| \leq |c_2|$. If c_1 divides c_2 then $\mathrm{SPoly}[p_1, p_2] = \mathrm{SPoly}_2[p_1, p_2]$, otherwise $\mathrm{SPoly}[p_1, p_2] = \mathrm{SPoly}_1[p_1, p_2]$. We remark that this is in essence "definition CP3" in [Kandri−Rody and Kapur 1984]. It is also the efficient generalization of the definition from [Buchberger 1985]. In that case one uses quotient and remainder to remove as much of the leading coefficient as possible from the lead term of the S−polynomial. When one lead coefficient divides the other it may be entirely removed and we have SPoly_2. When this does not happen, iterating the process emulates the Euclidean algorithm so after some number of steps we would obtain SPoly_1.

We are now ready to define precisely a strong Gröbner basis.

Definition 7: A set of polynomials G in $\mathbb{Z}[x_1, \ldots, x_n]$ is called a strong Gröbner basis over (the base ring) $\mathbb{Z}$ and with respect to a given term ordering T if, given any polynomial $p \in \mathbb{Z}[x_1, \ldots, x_n]$, it has a canonical reduction by $\{G, T\}$. What this means is that no matter what polynomials from G we use at any given step in the process, when we can no longer reduce it we have a unique form. Restated, if $p \xrightarrow{\{G,\ T\}} r_1$ and $p \xrightarrow{\{G,\ T\}} r_2$ and neither r_1 nor r_2 can be further reduced by F, then $r_1 = r_2$.

Last we will need a notion from the theory of Gröbner bases over principal ideal rings.

Definition 8: Given a set of polynomials $G = \{g_1, \ldots, g_n\} \subset \mathbb{Z}[x_1, \ldots, x_n]$ and a polynomial f with $f = \sum h_j\, g_j$.

We call this a strong standard representation of f with respect to G provided $\mathrm{HMonom}[f] = \mathrm{HMonom}[h_j\, g_j]$ for some j and $\mathrm{HPP}[h_k\, g_k] < \mathrm{HPP}[f]$ for all $k \neq j$ (obviously this is with respect to some given term order).

We see that in a strong standard representation one kills off the head term with exactly one summand. There is also a notion of a weak standard representation, wherein we allow multiple terms with the same head power product, that is useful in construction of what are called "weak" Gröbner bases. These in turn may be used to construct strong Gröbner bases as in [Möller 1989; Adams and Loustaunau 1994]. We do not pursue that approach here. Instead we will work directly with strong standard representations. These in fact give rise to strong Gröbner bases over principal ideal rings. The characterization in that case is that all elements of the ideal have a strong standard representation; we lose canonical forms of arbitrary polynomials. Is is easy to see that existence of such representations is equivalent to one of the common characterizing features from the field case: G is a strong Gröbner basis for the ideal I provided that for any $f \in I$ there is some $g \in G$ with $\mathrm{HMonom}[g] \mid \mathrm{HMonom}[f]$ (we require now that both lead coefficient and power product of f be divisible by those of g).

3 Main results

We want to establish a type of Buchberger result connecting Gröbner bases to reduction of S−polynomials. We will do this in steps.

Theorem 1: Given a set of polynomials $G = \{g_1, \ldots, g_n\}$ in $A = \mathbb{Z}[x_1, \ldots, x_i]$ and a term order T. Let I be the ideal generated by G. Then following are equivalent.

(i) Every $g \in G$ has a strong standard representation.

(ii) Every $f \in A$ has a canonical reduction by $\{G, T\}$ (in other words, G is a Gröbner basis with respect to order T).

Proof: (i)$\Longrightarrow$(ii) is similar to 10.22 and 10.23 in [Becker, Weispfenning, and Kredel 1993]. Suppose we have $f \xrightarrow{\{G,\ T\}} h_1$ and $f \xrightarrow{\{G,\ T\}} h_2$ with h_1 and h_2 both fully reduced. We need to show that $h_1 = h_2$. Since $h_1 - h_2 \in I$ it has a strong standard representation. Let $\mathrm{HMonom}[h_1 - h_2] = c\, t$ and $h_1 - h_2 = \sum q_j\, g_j$ be a strong standard representation with $\mathrm{HPP}[q_k\, g_k] = c\, t$. Let c_1 respectively c_2 be the coefficient of t in h_1 respectively h_2 . First suppose $c_1 = 0$. Then $\mathrm{HMonom}[h_2] = c\, t$ and hence h_2 is not fully reduced, contradicting our assumption. Thus $c_1 \neq 0$ and similarly we see that $c_2 \neq 0$. Hence $(c_1 - c_2)\, t$ reduces but neither $c_1\, t$ nor $c_2\, t$ reduces by G. Thus $b_k = \mathrm{HCoeff}[g_k]$ divides $(c_1 - c_2)$. Moreover

$$\mathrm{Quotient}[c_1, b_k] = \mathrm{Quotient}[c_2, b_k] = 0$$

for otherwise at least one of h_1 and h_2 could not be fully reduced. Thus c_1 and c_2 are in the same residue class modulo b_k and so they are equal. This shows that the head term of $h_1 - h_2$ is zero, in other words $h_1 = h_2$ as desired.

(ii)$\Longrightarrow$(i) is similar in style to 10.8 of [Becker, Weispfenning, and Kredel 1993]. Suppose $f \in I$, $f = c\,t + r$ where $t = \mathrm{HPP}[f]$. By assumption of canonical reduction we have $f \xrightarrow{\{G,\ T\}} 0$. Thus we may write $f = \sum h_j\, g_j$ where $\underset{j}{\mathrm{Max}}\left[\mathrm{HPP}\left[h_j\, g_j\right]\right] = t$ (we remark that this is already a weak standard representation of f). Let $J = \left\{j : \mathrm{HPP}\left[h_j\, g_j\right] = t\right\}$. Assume for a contradiction that $m = \sharp J > 1$, that t is minimal among all power products (with respect to T) for which this happens, and that $|c|$ is minimal among coefficients for which there is no strong standard representation involving this head power product t. These assumptions are tenable because we work with well ordered monomials over a totally ordered Euclidean domain.

For notational convenience assume without loss of generality that $J = \{1,\ \ldots,\ m\}$. Now let

$$\{\tilde{c},\ \{s_1\ ,\ \ldots,\ s_m\}\} = \mathrm{ExtendedGCD}\left[\mathrm{HCoeff}\left[g_1\ \right],\ \ldots\ \mathrm{HCoeff}[g_m]\right]$$

and $u_j = t/\mathrm{HPP}[g_m]$ for $1 \le j \le m$.

We next define $g = u_1\ \ s_1\ \ g_1\ \ + \ldots + u_m\, s_m\, g_m$. Then by construction $\mathrm{HMonom}[g] = \tilde{c}\, t$. If $|\tilde{c}| = |c|$ then $m = 1$because we use Euclidean reduction that forces $\left|\mathrm{HCoeff}\left[h_j\, g_j\right]\right| \le |c|$ for $1 \le j \le m$, yet by construction as a GCD we have $|\tilde{c}| \le \left|\mathrm{HCoeff}\left[h_j\, g_j\right]\right|$ for $1 \le j \le m$. Thus $|\tilde{c}| < |c|$.

By minimality of |c| there is a strong standard representation $g = \sum q_j\, g_j$ with $\mathrm{HPP}[q_k\, g_k] = t$ and $\mathrm{HPP}\left[q_j\, g_j\right] < t$ for all $j \neq k$. As $c = \mathrm{HCoeff}\left[h_1\ \ g_1\ \right] + \ldots + \mathrm{Hcoeff}[h_m\, g_m]$and $\tilde{c} = \mathrm{GCD}\left[\mathrm{HCoeff}\left[g_1\ \right],\ \ldots\ \mathrm{HCoeff}[g_m]\right]$ we see that $\tilde{c} \mid c$, so we have $c = d\,\tilde{c}$ for some d. Finally let $\tilde{f} = f - d\, g$. Then $\mathrm{HPP}[\tilde{f}] < t$ and hence $\tilde{f}$ has a strong standard representa−tion by our minimality hypothesis which we write as $\tilde{f} = \sum p_j\, g_j$. But then $d \sum q_j\, g_j + \sum p_j\, g_j$ is seen to be a strong standard representation of f. □

Theorem 2: Given a set of polynomials G in $\mathbb{Z}\left[x_1\ ,\ \ldots,\ x_n\right]$ and a term order T, the following are equivalent.

(i) G is a Gröbner basis with respect to term order T.

(ii) For every pair of polynomials $\left\{p_1\ ,\ p_2\ \right\} \subset G$ we have $\mathrm{SPoly}_1\ \left[p_1\ ,\ p_2\ \right] \xrightarrow{\{G,\ T\}} 0$ and $\mathrm{SPoly}_2\ \left[p_1\ ,\ p_2\ \right] \xrightarrow{\{G,\ T\}} 0$.

(iii) For every pair of polynomials $\left\{p_1\ ,\ p_2\ \right\} \subset G$ we have $\mathrm{SPoly}\left[p_1\ ,\ p_2\ \right] \xrightarrow{\{G,\ T\}} 0$.

We use both types of S−polynomial in the second equivalent statement because it is a bit easier to show that this yields a Gröbner basis. We then show that the third statement is equivalent to the second. This is useful among other reasons because one wants to retain the original Buchberger algorithm intact to the extent possible, and certainly having one rather than two S−polynomials for a given pair furthers this goal.

Proof: (i)$\Longrightarrow$(ii) is from the definition of a Gröbner basis. We now show (ii)$\Longrightarrow$(i) (this is similar to 10.11 in [Becker, Weispfenning, and Kredel 1993]).

Suppose $G = \left\{g_1\ ,\ \ldots,\ g_n\right\}$ and f is in the ideal generated by G. We may write $f = \sum h_j\, g_j$. Let $t\ = \underset{j}{\mathrm{Max}}\left[\mathrm{HPP}\left[h_j\, g_j\right]\right]$, $J = \left\{j : \mathrm{HPP}\left[h_j\, g_j\right] == t\right\}$, and $\tilde{t} = \mathrm{HPP}[f]$. We may assume t is minimal among such representations. Let $m = \sharp J$. If $m = 1$ and $t = \tilde{t}$ then we have a strong standard representation, so we assume otherwise. If $t = \tilde{t}$ then obviously $m > 1$. On the other hand, if $t > \tilde{t}$ then we require at least two terms in the representation to have power product of t in order to kill off that term. Hence $m > 1$. Reordering if necessary, without loss of generality we may assume $J = \{1,\ \ldots,\ m\}$.

We now set up some notation. Write $g_j\ =\ c_j\, t_j\ +\ r_j$ and $h_j\ =\ b_j\, s_j\ + q_j$ where $t_j = \mathrm{HPP}\left[g_j\right]$ and $s_j = \mathrm{HPP}\left[h_j\right]$. Note that $s_j\, t_j = t$ for $1 \le j \le m$. Let $t_{1\ ,\ 2}\ = \mathrm{PolynomialLCM}\left[t_1\ ,\ t_2\ \right]$, $v = t/t_{1\ ,\ 2}$, $u_1\ = t_{1\ ,\ 2}\ /t_1$, and $u_2\ = t_{1\ ,\ 2}\ /t_2$.

From this we see at once that $s_1 = u_1 v$ and $s_2 = u_2 v$. We will assume for a contradiction that $|b_1 c_1| + ... + |b_m c_m|$ is minimal among all representations of f that have a largest power product of t. Again, such a representation must exist for well ordered monomials over a totally ordered Euclidean domain.

Let $\{c, \{d_1, d_2\}\} = \text{ExtendedGCD}[c_1, c_2]$, $e_1 = \text{LCM}[c_1, c_2]/c_1$, $e_2 = -\text{LCM}[c_1, c_2]/c_2$. So e_1 and e_2 are minimal in norm such that $e_1 c_1 + e_2 c_2 = 0$.

In terms of these definitions we have

$$\text{Spoly}_1[g_1, g_2] = d_1 u_1 g_1 + d_2 u_2 g_2$$

$$\text{Spoly}_2[g_1, g_2] = e_1 u_1 g_1 + e_2 u_2 g_2$$

Now $b_1 c_1 + b_2 c_2 = d c$ for some d; moreover there exists e such that $b_1 = d d_1 + e e_1$ and $b_2 = d d_2 + e e_2$. Since $b_1 \neq 0$ and $b_2 \neq 0$ by construction, and $c = \text{GCD}[c_1, c_2]$, it follows that $|b_1 c_1| + |b_2 c_2| > |d c|$.

We now have

$$h_1 g_1 + h_2 g_2 = (d d_1 + e e_1) u_1 v g_1 + q_1 g_1 + (d d_2 + e e_2) u_2 v g_2 + q_2 g_2 =$$
$$d v \,\text{SPoly}_1[g_1, g_2] + e v \,\text{SPoly}_2[g_1, g_2] + (q_1 g_1 + q_2 g_2)$$

By hypothesis the S−polynomials reduce to zero. Now $v \,\text{HPP}[\text{SPoly}_2[g_1, g_2]] < t$, $\text{HPP}[q_1 g_1] < t$, and $\text{HPP}[q_2 g_2] < t$. Moreover $\text{HCoeff}[d v \,\text{SPoly}_1[g_1, g_2]] = d c$. We thus have a representation of $h_1 g_1 + h_2 g_2$ as a sum $\sum p_k g_k$ whereby, letting $K = \{k : \text{HPP}[p_k g_k] = t\}$, we obtain

$$\sum_{k \in K} |\text{HCoeff}[p_k g_k]| = |d c| < |b_1 c_1| + |b_2 c_2|$$

But then we may use this representation to replace $h_1 g_1 + h_2 g_2$ in the representation of f, and this contradicts minimality of $|b_1 c_1| + ... + |b_m c_m|$.

Since (ii) is stronger than (iii) it is clear that (ii)$\Longrightarrow$(iii). We show (iii)$\Longrightarrow$(ii).

Let $p_j = c_j t_j + r_j$ with $\text{HPP}[p_j] = t_j$ for $j \in \{1, 2\}$. Assume without loss of generality that $|c_1| \leq |c_2|$. If $c_1 \mid c_2$ then $\text{SPoly}_1[p_1, p_2]$ is trivially a product of p_2 and hence known to reduce, and thus we need only use $\text{SPoly}_2[p_1, p_2]$. So we may suppose that $c_1 \nmid c_2$. Let $\{c, \{a_1, a_2\}\} = \text{ExtendedGCD}[c_1, c_2]$ with $c_1 = d_1 c$ and $c_2 = d_2 c$. Note that $a_1 d_1 + a_2 d_2 = 1$and in particular a_1 and a_2 are relatively prime. Let $t = \text{PolynomialLCM}[t_1, t_2]$ with $s_1 = t/t_1$ and $s_2 = t/t_2$. Then

$$q = \text{SPoly}_1[p_1, p_2] = a_1 c_1 s_1 t_1 + a_1 s_1 r_1 + a_2 c_2 s_2 t_2 + a_2 s_2 r_2 = c t + a_1 s_1 r_1 + a_2 s_2 r_2$$

$$\text{SPoly}_2[p_1, p_2] = (d_2 c_1 s_1 t_1 + d_2 s_1 r_1) - (d_1 c_2 s_2 t_2 + d_1 s_2 r_2) = d_2 s_1 r_1 - d_1 s_2 r_2$$

Thus

$$h_1 = \text{SPoly}_2[p_1, q] = c_1 s_1 t_1 + s_1 r_1 - d_1 (c t + a_1 s_1 r_1 + a_2 s_2 r_2)$$
$$= (1 - d_1 a_1) s_1 r_1 - d_1 a_2 s_2 r_2 = a_2 d_2 s_1 r_1 - a_2 d_1 s_2 r_2 = a_2 \,\text{SPoly}_2[p_1, p_2]$$

and similarly $h_2 = \text{SPoly}_2[p_2, q] = a_1 \,\text{SPoly}_2[p_1, p_2]$.

Also by definition 6 it is clear that $\mathrm{SPoly}_2\left[p_j, q\right] = \mathrm{SPoly}\left[p_j, q\right]$ for $j \in \{1, 2\}$.

Since a_1 and a_2 are relatively prime we obtain $\mathrm{SPoly}_1\left[h_1, h_2\right] = \mathrm{SPoly}_2\left[p_1, p_2\right]$. This shows that provided $\mathrm{SPoly}_1\left[p_1, p_2\right]$ is not trivial we will eventually obtain $\mathrm{SPoly}_2\left[p_1, p_2\right]$ by iterating SPoly. Hence for any pair $\{p_1, p_2\}$ we need only use $\mathrm{SPoly}\left[p_1, p_2\right]$ as given in definition 6. □

From the theorems above it is now not hard to see that our bases are the same as the D−bases of [Pan 1989; Becker, Weispfenning, and Kredel 1993]. What is different is the mode of computation insofar as we allow Euclidean reduction of lead coefficients rather than insist on divisibility. This will tend to make them smaller sooner, and thus could offer an advantage in efficiency. Note that this only applies when working over a Euclidean domain, and so the algorithm in the above references has the advantage of greater generality, albeit ours has greater flexibility in choices of reducing polynomial.

There are other ways to improve computational efficiency. It is known from long experience that the common bottleneck to the algorithm is the reduction of S−polynomials. Buchberger himself was the first to give criteria under which certain S−polynomials could be ignored (see [Buchberger 1985] and references therein). We recover in part his criteria from the field case.

Theorem 3 (Buchberger's criterion 1): Suppose $p_j = c_j t_j + r_j$ with $\mathrm{HPP}\left[p_j\right] = t_j$ for $j \in \{1, 2\}$ and $c_1 \mid c_2$. Suppose further that the lead power products t_1 and t_2 are coprime, that is, $\mathrm{PolynomialLCM}\left[t_1, t_2\right] = t_1 t_2$. Then $\mathrm{SPoly}\left[p_1, p_2\right]$ will reduce to zero and hence is superfluous.

Note that we are using $\mathrm{SPoly}_2\left[p_1, p_2\right]$ in this case. While the divisibility requirement for lead coefficients might seem unduly strong, one will observe that the algorithm proceeds in such a way as to make coefficients small with respect to Euclidean norm. Thus in practice this requirement may not be terribly restrictive.

Proof: $\mathrm{SPoly}\left[p_1, p_2\right] = c_1 t_1 p_2 - c_2 t_2 p_1 = \left(p_1 - r_1\right) p_2 - \left(p_2 - r_2\right) p_1 = r_2 p_1 - r_1 p_2$. As t_1 divides the head term of $r_2 p_1$ while t_2 does not, and t_2 divides the head term of $r_1 p_2$ while t_1 does not, these do not collapse further. But clearly $r_2 p_1 \xrightarrow{p_1} 0$ and $r_1 p_2 \xrightarrow{p_2} 0$, so $\mathrm{SPoly}\left[p_1, p_2\right] \longrightarrow 0$. □

Theorem 4 (Buchberger's criterion 2): Given $p_j = c_j t_j + r_j$ with $\mathrm{HPP}\left[p_j\right] = t_j$ for $j \in \{1, 2, 3\}$ with $\mathrm{PolynomialLCM}\left[t_1, t_2\right]$ divisible by t_3. Suppose $\mathrm{SPoly}\left[p_1, p_3\right]$ and $\mathrm{SPoly}\left[p_2, p_3\right]$ have strong standard representations (thus far these are the conditions for criterion 2 to be in effect in the field case). If either $c_1 \mid c_3 \mid c_2$ or $c_3 \mid c_1 \mid c_2$ then $\mathrm{SPoly}\left[p_1, p_2\right]$ will have a strong standard representation and hence is superfluous.

Note that again we are working with $\mathrm{SPoly}_2\left[p_1, p_2\right]$. Obviously the roles of p_1 and p_2 can be interchanged. Moreover, while the divisibility conditions again appear to be restrictive, in general one obtains alot of polynomials with lead coefficient a unit and these make the conditions not so uncommon.

Proof: Let $t = \mathrm{PolynomialLCM}\left[t_1, t_2\right]$. Assume inductively that if f and g have strong standard representations, and $\mathrm{HPP}[f] < t$, $\mathrm{HPP}[g] < t$, then so does $f + g$.

Define power product multipliers $u_{j,k} = \mathrm{PolynomialLCM}\left[t_j, t_k\right] / t_j$. Then $\mathrm{SPoly}\left[p_1, p_2\right] = \frac{c_2}{c_1} u_{1,2} p_1 - u_{2,1} p_2$

First we assume $c_3 \mid c_1 \mid c_2$. Then $\text{SPoly}[p_1, p_3] = u_{1,3}\, p_1 - \frac{c_1}{c_3} u_{3,1}\, p_3$ and $\text{SPoly}[p_2, p_3] = u_{2,3}\, p_1 - \frac{c_2}{c_3} u_{3,2}\, p_3$.

Since $t_3 \mid \text{PolynomialLCM}[t_1, t_2]$ we know that $u_{1,3} \mid u_{1,2}$ and similarly $u_{2,3} \mid u_{2,1}$. We thus may write

$$\frac{c_2}{c_1} \frac{u_{1,2}}{u_{1,3}} \text{SPoly}[p_1, p_3] - \frac{u_{2,1}}{u_{2,3}} \text{SPoly}[p_2, p_3] =$$

$$\frac{c_2}{c_1} u_{1,2}\, p_1 - \frac{c_2}{c_3} \frac{u_{1,2}}{u_{1,3}} u_{3,1}\, p_3 - \left(u_{1,2}\, p_1 - \frac{c_2}{c_3} \frac{u_{2,1}}{u_{2,3}} u_{3,2}\, p_3 \right) =$$

$$\text{SPoly}[p_1, p_2] - \frac{c_2}{c_3} p_3 \left(\frac{u_{1,2}}{u_{1,3}} u_{3,1} - \frac{u_{2,1}}{u_{2,3}} u_{3,2} \right)$$

Now $u_{1,2}\, t_1 = u_{2,1}\, t_2$, $u_{1,3}\, t_1 = u_{3,1}\, t_3$, and $u_{2,3}\, t_2 = u_{3,2}\, t_3$. This implies $\frac{u_{1,2}}{u_{1,3}} u_{3,1} = \frac{u_{2,1}\, t_2}{u_{3,1}\, t_3} u_{3,1} = \frac{u_{2,1}\, t_2}{t_3}$ and similarly $\frac{u_{2,1}}{u_{2,3}} u_{3,2} = \frac{u_{2,1}\, t_2}{t_3}$. Hence the parenthesized term van−ishes, and so

$$\text{SPoly}[p_1, p_2] = \frac{c_2}{c_1} \frac{u_{1,2}}{u_{1,3}} \text{SPoly}[p_1, p_3] - \frac{u_{2,1}}{u_{2,3}} \text{SPoly}[p_2, p_3]$$

Now use the hypothesis that each summand has a strong standard representation with head power product smaller than *t*. Then so does the sum.

The case where $c_1 \mid c_3 \mid c_2$ is similar. For the first step, one instead shows

$$\text{SPoly}[p_1, p_2] = \frac{c_2}{c_3} \frac{u_{1,2}}{u_{1,3}} \text{SPoly}[p_1, p_3] - \frac{u_{2,1}}{u_{3,2}} \text{SPoly}[p_2, p_3]. \quad \square$$

A more general treatment of this criterion may be found in [Möller 1985], based on generating sets of homogeneous syzygy modules. We use this version because it is simple to code; as it is in essence the usual Buchberger criterion 2 one can adapt "standard" code for the field case with only minor modification (as indeed is done in the *Mathematica* implementation).

One will note that the criteria above pertain to the second type of S−polynomial, and naturally it would be nice to have a criterion for eliminating as redundant an S−polynomial of the first type. There is such a criterion implicit in theorem 10.11 of [Becker, Weispfenning, and Kredel 1993].

Theorem 5: Given $p_j = c_j\, t_j + r_j$ with $\text{HPP}[p_j] = t_j$ for $j \in \{1, 2, 3\}$ with $t_{i,j} = \text{PolynomialLCM}[t_i, t_j]$. Suppose $t_3 \mid t_{1,2}$. Let $\{c, \{a_1, a_2\}\} = \text{ExtendedGCD}[c_1, c_2]$ and further suppose $c_3 \mid c$. In other words, the head mono−mial of p_3 divides the head monomial of $\text{SPoly}[p_1, p_2]$ (the latter is top−D−reducible, in the terminology of [Becker, Weispfenning, and Kredel 1993]). Then $\text{SPoly}[p_1, p_2]$ is redundant.

Proof: Let $u_{i,j} = t_{i,j} / t_i$ for $j \in \{1, 2, 3\}$. Let $c / c_3 = d$ and $t_{1,2} / t_3 = v$. Then

$$\text{SPoly}[p_1, p_2] = a_1\, u_{1,2}\, p_1 + a_2\, u_{2,1}\, p_2 = c\, t_{1,2} + a_1\, u_{1,2}\, r_1 + a_2\, u_{2,1}\, r_2$$

Also

$$\text{SPoly}[p_1, p_3] = \text{SPoly}_2[p_1, p_3] =$$
$$(c_1 t_1 + r_1) u_{1,3} - (c_1/c_3)(c_3 t_3 + r_3) u_{3,1} = u_{1,3} r_1 - (c_1/c_3) u_{3,1} r_3$$

and similarly $\text{SPoly}[p_2, p_3] = u_{2,3} r_2 - (c_2/c_3) u_{3,2} r_3$.

Now

$$\frac{u_{1,2} u_{3,1}}{u_{1,3}} = \frac{(t_{1,2}/t_1)(t_{1,3}/t_3)}{t_{1,3}/t_1} = t_{1,2}/t_3 = v$$

and a similar computation shows that $\frac{u_{2,1} u_{3,2}}{u_{2,3}} = v$. Also

$$a_1 \frac{c_1}{c_3} + a_2 \frac{c_2}{c_3} - d = \frac{1}{c_3}(a_1 c_1 + a_2 c_2) - d = \frac{c}{c_3} - d = 0$$

Hence

$$\text{SPoly}[p_1, p_2] - d v p_3 - a_1 \frac{u_{1,2}}{u_{1,3}} \text{SPoly}[p_1, p_3] - a_2 \frac{u_{2,1}}{u_{2,3}} \text{SPoly}[p_2, p_3] =$$
$$c t_{1,2} + a_1 u_{1,2} r_1 + a_2 u_{2,1} r_2 - (c t_{1,2} + d v r_3)$$
$$-\left(a_1 u_{1,2} r_1 - a_1 \frac{c_1}{c_3} \frac{u_{1,2} u_{3,1}}{u_{1,3}} r_3\right) - \left(a_2 u_{2,1} r_2 - a_2 \frac{c_2}{c_3} \frac{u_{2,1} u_{3,2}}{u_{2,3}} r_3\right)$$
$$= \left(a_1 \frac{c_1}{c_3} \frac{u_{1,2} u_{3,1}}{u_{1,3}} + a_2 \frac{c_2}{c_3} \frac{u_{2,1} u_{3,2}}{u_{2,3}} - d v\right) r_3 = 0$$

We have thus a strong standard representation of $\text{SPoly}[p_1, p_2]$ and this suffices to show that it is redundant. □

We now have our algorithm, essentially the same as the Buchberger algorithm for polynomial rings over fields. We list all pairs of polynomials, marking as processed all those that the criteria warrant. We now iteratively select a pair whose S−polynomial is not yet marked, reduce it, and if the result is not zero, we form new pairs. Again use the criteria to mark redundant pairs. We continue this iteration until we have no more pairs to process, at which point all S−polynomials can be reduced to zero. Termination in a finite number of steps is proven e.g. in [Kandri−Rody and Kapur 1988] by noting that $\mathbb{Z}[x_1, \ldots, x_n]$ is Noetherian and hence an ascending chain condition applies to its ideals.

One will note that our algorithm puts a certain emphasis on SPoly_1, wherein the lead coefficient is the GCD of the leading coefficients of the critical pair. This is in contrast to algorithms in [Möller 1988; Pan 1989; Weispfenning, and Kredel 1993; Adams and Loustaunau 1994] where the emphasis is more on SPoly_2 in which, as with the field case, one entirely kills off a leading coefficient. Given the dearth of available implementations, it is an open question as to which approach is computationally more effective in general.

4 Some special cases

Before proceeding to examples we will discuss an important special class of Euclidean domains. While one can show that the theory developed above carries over in a general way, for the specific and very important case where our base ring is the set of univariate polynomials in x over a (computable) field $\mathbb{F}$ one can do better. Suppose we are given a set of polynomials in some set of indeterminates over $\mathbb{F}[x]$. One augments the indeterminates with x, extend−ing the term order so that every power of x is smaller than all power products containing other variables. One next computes a Gröbner basis for the input in this setting of polynomials in one more variable over $\mathbb{F}$. Theorem 4.5.12 in [Adams and Loustaunau 1994] shows that this is in fact a strong Gröbner basis for the ideal over the original base ring $\mathbb{F}[x]$ (this fact had also been mentioned in [Kandri−Rody and Kapur 1988]). Our experience is that the benefits of computing a basis over a field outweigh any efficiencies developed for working over a Euclidean domain, hence we use this tactic in *Mathematica*. We show applications to working over such polynomial rings in the examples.

If our base ring as $\mathbb{Z}_{p^n}$ and we work with univariate polynomials over this ring then we have an example of a finite−chain ring. For working with polynomials over such a ring we could use the results presented in [Norton and Sălăgean 2001] . They show, among other things, that weak and strong Gröbner bases for ideals over such rings are equivalent. They also present a structure theorem for the univariate case and apply it to cyclic codes. As will be seen in the examples below, one can instead regard the ring as a quotient of $\mathbb{Z}[x]$ and use the computational methods of this paper. We will return to this special case below, in the context of Hensel lifting. In a sense this is the analog of the way that Gröbner bases for polynomial ideals over fields generalize the Euclidean algorithm for the univariate case.

The other well known direction in which Gröbner bases generalize an older concept is that of row reduction. Hence another special case for bases over Euclidean domains is when all the polynomials are linear. We also show an application of this case to matrix normal forms in the sequel. A minor modification will moreover yield a polynomial lattice reduction.

5 Examples

In this and later sections we show several examples. We omit most proofs that the algorithms do as we state. Such proofs would use arguments based on term ordering and integer sizes, and are generally straightforward.

We first show some simple examples adapted from [Adams and Loustaunau 1994]. For purposes of assessing speed, we note that all timings were done with version 5 of *Mathematica* running on a 1.5 GHz Intel processor under the Linux operating system.

For the first example we wish to compute a basis for an ideal in the polynomial ring $\mathbb{Z}\left[\sqrt{-5}\right][x, y]$. Note that our base ring, $\mathbb{Z}\left[\sqrt{-5}\right]$, is not a Euclidean domain (or even a unique factorization domain). In such cases one may resort to a common tactic of adding a new variable and defining polynomial so that in effect we work over a quotient ring; in this example it will be $\mathbb{Z}[x, y, \alpha] \big/ \left\{\alpha^2 + 5\right\}$. So our base ring will be the integers and we have added a variable and a polynomial relation equating that variable to $\sqrt{-5}$ (up to a conjugate, as these are indistinguishable to this method without further variables and defining polynomials). For this to work as desired we must have the new variable ordered lexicographically lower than all others. We then remove the first polynomial from the basis, which, due to this ordering, is exactly the defining polynomial for that algebraic extension element.

```
Rest[
  GroebnerBasis[{2 x y − α y, (1 + α) x^2 − x y, α^2 + 5}, {x, y, α}, CoefficientDomain → Integers]]
```

$$\{25\ y + 10\ y^2 - 5\ y\alpha,\ 15\ y + 5\ y^2 + y^2\ \alpha,\ -25\ y + xy + 5\ y^3 + 12\ y\alpha,\ 6\ x^2 + 10\ y + 5\ y^2 - 3\ y\alpha,\ x^2 - 25\ y + 5\ y^3 + x^2\ \alpha + 12\ y\alpha\}$$

The basis in that reference is a bit different due to different notions of coefficient handling, but the one above serves the same purposes.

As a second example, we will find a basis for the ideal intersection $\{3x - 2, 5y - 3\} \cap \{xy - 6\}$ in $\mathbb{Z}[x, y]$. This may be done as below. Note that we again use and subsequently eliminate an auxiliary variable, this time ordered lexico−graphically greater than the others (specifying it as the third argument tells `GroebnerBasis` it is to be eliminated).

```
GroebnerBasis[Flatten[{w {3 x − 2, 5 y − 3}, (1 − w) {x y − 6}}], {x, y},
  w, CoefficientDomain → Integers, MonomialOrder → EliminationOrder]
```

$$\{18 - 30\ y - 3\ xy + 5\ xy^2,\ 12 - 18\ x - 2\ xy + 3\ x^2\ y,\ 6 - 6\ y - 7\ xy + xy^2 + x^2\ y^2\}$$

Again, we do not obtain the identical basis due to differences in basis definition. Specifically, theirs does not have our third polynomial. This is because they find a weak Gröbner basis and that requires fewer polynomials. The disadvantage to that, as noted earlier, is that one now must work harder to reduce with these, and moreover one cannot readily obtain canonical forms.

To get some idea of algorithm speed, we now show a more strenuous computation.

```
polys = {7 x^2 y^2 + 8 x y^2 + 3 x z − 11, 11 y^2 z + 4 x^2 y + x y z^2 + 2,
      5 x^2 y z + x^2 + 2 z^2 + 5 z, 7 x y z + 3 x y + 5 x + 4 y + 7};
Timing[gbdlex = GroebnerBasis[polys, {x, y, z}, CoefficientDomain → Integers,
      MonomialOrder → DegreeLexicographic]]
```

```
{1.05 Second , {34 475 640 417 355 562 336 236 396 270 436 281 195 926 ,
  10 898 452 513 151 823 962 606 330 508 750 762 670 219 + z,
  6 355 322 887 725 405 337 810 105 619 887 333 184 234 − y,
  −14 760 987 199 637 601 090 452 154 096 210 512 593 721 + x}}
```

A similar basis computed over the field of rationals is about 70 times faster using the same hardware and software (the result, as might be expected, is {1} because we started with more polynomials than variables). So the fact that the Euclidean domain case takes almost two orders of magnitude longer is not entirely a surprise insofar as the eventual result contains much more information. If we remove the first polynomial then the tasks are in some sense more similar and correspondingly the relative time ratio of computing over the rationals vs. the integers drops to under one order of magnitude.

An application of finding bases over the integers was pointed out to the author by Dan Grayson [Grayson 1995], and in fact was implemented by him in *Mathematica* around 1988 (using the Gröbner basis over integers algorithm from [Kandri−Rody and Kapur 1984]). Given a system of $n + 1$ polynomials in n unknowns, find a modulus m such that the system is exactly determined modulo m, and return all solutions (which lie in $(\mathbb{Z}_m)^n$). With reference to the previous example, the above system is seen to be exactly determined in the quotient ring $\mathbb{Z}_{34\,475\,640\,417\,355\,562\,336\,236\,396\,270\,436\,281\,195\,926}$.

A related application is to do computations involving ideals defined over quotient rings that may contain zero divisors. As an example we will find all solutions in the ring $\mathbb{Z}_{5\,072\,012\,170\,009}$ to a system below.

```
gb = GroebnerBasis[
   {5 072 012 170 009, −4 984 359 602 099 + x^2 − 3 y^2 − 9 x z, −1 780 431 462 965 + 7 x y + 5 y^3 + z^2,
    −4 585 397 367 278 + x^3 − 3 y^2 + z − 12 z^3}, {x, y, z}, CoefficientDomain → Integers]
```

```
{5 072 012 170 009 , 1 174 872 829 454 + 12 173 501 962 z − 1 363 165 624 472 z^2 +
   1 654 998 137 452 z^3 + 928 181 308 002 z^4 − 239 795 324 199 z^5 − 1 646 238 538 583 z^6 −
   982 686 930 325 z^7 − 1 734 356 432 441 z^8 − 1 928 316 724 538 z^9 + 2 384 106 829 761 z^10 −
   2 266 219 400 230 z^11 − 139 245 405 743 z^12 + 895 384 068 341 z^13 + 161 928 956 428 z^14 +
   2 194 204 640 034 z^15 − 1 243 172 466 690 z^16 − 1 196 909 984 892 z^17 + z^18 ,
  2 247 545 052 503 + y + 788 535 951 374 z + 2 214 230 166 342 z^2 + 955 710 141 543 z^3 +
   2 160 238 766 386 z^4 − 2 474 194 692 542 z^5 − 1 684 716 364 278 z^6 + 2 157 370 757 916 z^7 −
   1 072 725 791 722 z^8 + 1 173 330 106 507 z^9 − 1 057 647 942 280 z^10 −
   1 511 353 993 603 z^11 + 1 327 624 312 048 z^12 − 581 007 814 126 z^13 +
   1 772 345 363 132 z^14 − 185 000 519 654 z^15 − 1 538 648 034 589 z^16 − 456 160 565 195 z^17 ,
  −899 617 339 822 + x + 2 209 081 769 554 z − 509 675 450 156 z^2 + 566 438 534 091 z^3 +
   1 828 943 883 971 z^4 − 1 778 487 828 359 z^5 − 1 120 529 181 700 z^6 + 1 238 816 552 216 z^7 −
   1 898 793 743 218 z^8 + 1 286 010 808 749 z^9 + 893 019 914 153 z^10 + 172 896 055 599 z^11 +
   1 872 411 543 380 z^12 + 1 420 313 673 322 z^13 − 880 454 763 764 z^14 −
   1 202 867 057 825 z^15 − 1 977 589 465 047 z^16 − 2 210 999 439 349 z^17 }
```

To obtain solutions one would proceed exactly as if working over a field. Specifically, we first find roots of the univariate polynomial, then back substitute each solution to solve for the remaining variables. We show the first step explicitly. This involves root finding in a quotient ring of the integers. The principles behind this are well known (factor the modulus, find roots modulo each prime factor, lift to accomodate powers of primes, use Chinese Remainder Algorithm to combine roots modulo powers of primes). The "hard" step, computationally speaking, is often the factorization of the modulus.

```
Roots[gb[[2]] == 0, z, Modulus → gb[[1]]]
```

```
z == 99 999 || z == 1 848 935 269 876 || z == 3 102 255 902 823
```

This functionality is now built into *Mathematica*, in the function `Reduce`:

```
Timing[Reduce[{−4 984 359 602 099 + x^2 − 3 y^2 − 9 x z, −1 780 431 462 965 + 7 x y + 5 y^3 + z^2,
      −4 585 397 367 278 + x^3 − 3 y^2 + z − 12 z^3} == 0, {x, y, z}, Modulus → 5 072 012 170 009]]
```

```
{0.22 Second , (x == 77 777 && y == 88 888 && z == 99 999 ) ||
   (x == 1 712 760 123 092 && y == 3 989 577 716 979 && z == 1 848 935 269 876 ) ||
   (x == 2 127 801 384 642 && y == 3 379 908 964 470 && z == 3 102 255 902 823 )}
```

Another area of application for Gröbner bases over the integers is in computations with finitely presented groups, as discussed in chapter 10 of [Sims 1994]. Among other tools one requires a module Gröbner basis. This is something we show in section 7 below.

6 Application: Hensel lifting of univariate polynomials

We now show an application that uses the special case of polynomials in one variable over the integers modulo a power n of a prime p. We begin with a simple example rigged so that the correct result is obvious.

```
poly = Expand[(x^5 + 18 x^4 + 34 x^3 + 5 x^2 + 21 x + 30) (x^4 + 24 x^3 + 22 x^2 + 17 x + 15)];
```

We will first factor the polynomial modulo a small prime, removing the (possibly trivial) constant factor.

```
mod = 11;
fax = FactorList[poly, Modulus → mod];
fax = First /@ Rest[fax]
```

$$\left\{4+6\,x+2\,x^{3}+x^{4},\;\; 8+10\,x+5\,x^{2}+x^{3}+7\,x^{4}+x^{5}\right\}$$

Next we wish to make the factors correct modulo a power of the prime. This correction step is referred to as Hensel lifting [von zur Gathen and Gerhard 1999, chapter 15] and is used in most algorithms for factoring polynomials over the rationals. It is typically done by iterations of Newton's method in a p−adic setting, but Gröbner bases may instead be used to advantage. In effect we take p−adic gcds of our polynomial and each factor raised to the indicated power, and these gcds are the lifted factors. For this particular example we will take the factors, square them, compute Gröbner bases over the integers of the set {*poly*, *squaredfactor*, *squaredmodulus*}, and extract the last elements of these bases. This will correspond to quadratic Hensel lifting, insofar as a factor that is correct modulo some value p becomes correct modulo p^2 . We will in so doing recover the original factors up to sign.

```
(Last[GroebnerBasis[{mod^2, poly, #1}, CoefficientDomain → Integers]] &) /@ fax^2
```

$$\left\{-15-17\,x-22\,x^{2}-24\,x^{3}-x^{4},\;\; 30+21\,x+5\,x^{2}+34\,x^{3}+18\,x^{4}+x^{5}\right\}$$

This recovered the actual factors because we arranged an example for which the modular factors each corresponded to an actual factor, and moreover the factors were monic, had coefficients of the same sign, and these were all less than half the prime squared. Hence they are recovered exactly from one quadratic Hensel lift. The question to be answered is why these Gröbner basis computations gave the quadratic Hensel lifts of the modular factors. We address this next.

Theorem 6: Given a square free univariate polynomial f over the rationals, and an integer p such that the leading coefficient of f is not divisible by p, f is square free modulo p, and $f \equiv_p g_0\, h_0$. Assume $s = \mathrm{GCD}\left[g_0{}^2,\, f\right]$ exists modulo p^2 . Then s is the Hensel lift of g_0 modulo p^2 .

Note that this p−adic gcd may be computed, as above, by a Gröbner basis over the integers. Indeed it is simply a convenient shorthand for running the Euclidean algorithm under the assumption that no zero divisors are encoun− tered along the way.

Proof: We are given $f \equiv_p g_0\, h_0$. Suppose the quadratically lifted equation is $f \equiv_{p^2} g_1\, h_1$ where $g_1 \equiv_p g_0$ and $h_1 \equiv_p h_0$. The assumptions imply that the degrees of g_0 and g_1 are equal (and likewise with the cofactors). We may write $g_1 = g_0 + p\, t_0$. Then a simple computation shows that $g_1\,(g_0 - p\, t_0) \equiv_{p^2} g_0{}^2$. We see that $g_1 \mid f$ and $g_1 \mid g_0{}^2$ modulo p^2 . Now let $s = \mathrm{GCD}\left[g_0{}^2,\, f\right]$. Then we have $g_1 \mid s$. In order to show these are equal up to unit multiples (which proves the theorem), it suffices to show that $\mathrm{degree}[g_1] \geq \mathrm{degree}[s]$.

Suppose $\mathrm{degree}[s] > \mathrm{degree}[g_1]$. Then $\mathrm{degree}[s] > \mathrm{degree}[g_0]$. Since $s \mid f$ modulo p^2 we have $s \mid f$ modulo p. But also $s \mid g_0{}^2$ so the strict degree inequality implies that s is not square free modulo p. Hence f is not square free modulo p, contradicting our assumption. □

One may observe that a polynomial factorization code based on this result will have a probabilistic aspect. We might inadvertently use an "unlucky" prime wherein at some step of the lifting process a GCD does not exist. This can happen if a leading coefficient in the process becomes noninvertible because it is a product of p. It is not hard to see that for a given polynomial there can only be finitely many such unlucky primes. Moreover provided one uses a random prime that is large compared to the degree of factors and degree of lifting required, the probability will be low that the prime is unlucky.

To give some indication of efficiency we now demonstrate on a more challenging example. It stems from a factorization example presented in [van Hoeij 2002]. We first set up the polynomial in question; its roots are all the sums of pairs of roots of a simpler polynomial.

```
poly1 = x^20 − 5 x^18 + 864 x^15 − 375 x^14 − 2160 x^13 + 1875 x^12 + 10 800 x^11 + 186 624 x^10 − 54 000 x^9 +
    46 875 x^8 + 270 000 x^7 − 234 375 x^6 − 2 700 000 x^5 − 1 953 125 x^2 + 9 765 625;
rts = x /. Solve[poly1 == 0, x];
sums = Flatten[Table[rts[[i]] + rts[[j]], {i, 19}, {j, i + 1, 20}]];
newpoly = Expand[Times @@ (x − N[sums, 200])];
newpoly = Chop[newpoly] /. a_Real → Round[a];
```

The end goal is to factor this over the integers. While it would take us too far afield to discuss the steps that use lattice reduction, we will show the Hensel lifting phase below. To this end we first factor modulo a prime.

```
mod = Prime[4000];
fax = FactorList[newpoly, Modulus → mod];
fax = First /@ Rest[fax];
```

Next we wish to make the factors correct modulo a power of the prime. The specific power is dictated by size considerations that arise in the factorization algorithm; for our example it will be 36. For reasons of efficiency it is better to iterate squarings rather than try to lift to the full power in one step, as the squaring method keeps the degree relatively small during the lifting process. We must then do more basis computations, but the improved speed per computation more than compensates for this. Hence we are, as above, doing quadratic Hensel lifting.

```
liftfactors[fax_, poly_, mod_, pow_] :=
  Module[{modpow = mod, top = Ceiling[Log[2, pow]], liftedfax = fax},
    Do[modpow = If[j == top, mod ^ pow, modpow ^ 2];
      liftedfax = Expand[liftedfax ^ 2, Modulus → modpow];
      liftedfax =
        Map[Last[GroebnerBasis[{modpow, poly, #}, CoefficientDomain → Integers]] &,
          liftedfax], {j, top}];
    liftedfax]
Timing[liftedfax = liftfactors[fax, newpoly, mod, 36];]
```

```
{5.51 Second , Null }
```

There are tactics to improve on this. One possibility, for example, might be to adapt the asymptotically fast HGCD algorithm presented in chapter 8 of [Aho, Hopcroft, and Ullman 1974]. All the same we have attained timings comparable to what was presented in [van Hoeij 2002] for this step of the algorithm using but a few lines of code to implement the Hensel lift. The rest of the factorization involves constructing and reducing a particular lattice, and takes under 2 seconds using the same machine and software as above. Note that prior to the advent of the van Hoeij algorithm this example was essentially intractable.

Some further remarks about this method of p−adic lifting are in order. First, clearly dedicated code will be faster than a general purpose Gröbner basis program. We have such code in *Mathematica*, and for the example above it is about five times faster. Tests on more strenuous problems indicate that the dedicated code is quite competitive with what seems to be the best Hensel lifting method in the literature to date, Shoup’s "tree−lift" (which is a form of divide−and−conquer algorithm) [von zur Gathen and Gerhard 1999, chapter 15, section 5]. Specifically, while it is clear that the behavior of Shoup’s method is asymptotically better than that of the method presented above (it relies on computation of quotients and remainders rather than GCDs), our experience was that for practical purposes the method in this section was actually faster for the knapsack factorization examples we tried at [Zimmerman 2003]. As these typically required lifting to many digits, this is evidence of the practicality of the method above.

In the interest of full disclosure it should be remarked that some of the examples were quite near the crossover when they reached the final lift stage. Moreover the issue of speed is of course tied to the quality of code, and it may be the case that the code underlying our Shoup implementation was insufficiently optimized. Other noteworthy differences are that the Shoup method requires about twice as much code, but, once a prime is found for which the factorization is square free, it cannot fail whereas, as per theorem 6, the p−adic GCD computation may fail at later stages. Further details regarding the factorization of these polynomials via the knapsack algorithm are presented in [Belabas, Hanrot, and Zimmerman 2001].

7 Application: Computation of matrix Hermite normal forms

Another nice application of Gröbner bases over a Euclidean domain is in computing the Hermite normal form of a matrix with elements in that domain. As there is an efficient *Mathematica* implementation of integer Hermite normal form based on [Storjohann 1994], we illustrate for the case of matrices of univariate polynomials.

Before we show an example we need code to generate a "random" polynomial matrix. For this example we will use a 3x5 matrix of polynomials in x of degree at most 2.

```
randomPolynomial[deg_Integer, var_] :=
  Table[var^j, {j, 0, deg}].RandomInteger[{−10, 10}, deg + 1]
randomMatrix[degmax_, rows_, cols_, var_] := Module[{deg},
    Table[deg = RandomInteger[{0, degmax}]; randomPolynomial[deg, var], {rows}, {cols}]]
SeedRandom[1111] mat = randomMatrix[2, 3, 5, x];
```

To set this up we need to extend `GroebnerBasis` to handle modules, using a "position over term" ordering [Adams and Loustaunau 1994]. We represent elements as vectors with respect to module basis variables. The input consists of polynomials that are linear with respect to the module variables. We then augment with relations that force all products of the module variables to be zero and find the Gröbner basis. The code below is taken from [Lichtblau 1996].

```
moduleGroebnerBasis[polys_, vars_, cvars_, opts___] :=
  Module[{newpols, rels, len = Length[cvars], gb, j, k, ruls},
    rels = Flatten[Table[cvars[[j]] * cvars[[k]], {j, len}, {k, j, len}]];
    newpols = Join[polys, rels];
    gb = GroebnerBasis[newpols, Join[cvars, vars], opts];
    rul = Map[(# → {}) &, rels];
    gb = Flatten[gb /. rul];
    Collect[gb, cvars]]
```

As the Hermite form is obtained by row operations over the base ring (that is, division is forbidden), it is equivalent to a module Gröbner basis in the case where our polynomial ring is just the base ring (that is, there are no polyno− mial variables). We convert each row of the matrix to a polynomial vector representation by making each column into a new "variable". At this point we can use the module Gröbner basis routine above. We then convert the result back to matrix form.

```
groebnerHNF[mat_?MatrixQ, domain_, mod_: 0] := Module[
    {len = Length[First[mat]], newvars, generators, mgb},
    newvars = Array[v, len];
    generators = mat.newvars;
    mgb = moduleGroebnerBasis[generators,
        {}, newvars, CoefficientDomain → domain, Modulus → mod];
    Outer[D, Reverse[mgb], newvars]]
```

Now we obtain our module basis over $\mathbb{Z}_{8933}[x]$. We work over a prime field in order to restrict the size of the coefficients.

```
hnf = groebnerHNF[mat, Polynomials[x], 8933]
```

$$\{\{1, 0, 4832 + 3665 x + 3652 x^2 + 3695 x^3,$$
$$8283 + 8735 x + 74 x^2 + 3405 x^3 + 6787 x^4 + 7042 x^5,$$
$$3056 + 4811 x + 3887 x^2 + 7902 x^3 + 174 x^4\},$$
$$\{0, 1, 4183 + 7075 x + 5100 x^2 + 4074 x^3,$$
$$505 + 155 x + 3912 x^2 + 3307 x^3 + 8617 x^4 + 5441 x^5,$$
$$7548 + 1222 x + 947 x^2 + 2787 x^3 + 5820 x^4\},$$
$$\{0, 0, 2434 + 3140 x + 1796 x^2 + 2494 x^3 + x^4,$$
$$1761 + 2265 x + 2999 x^2 + 2492 x^3 + 7414 x^4 + 123 x^5 + 7656 x^6,$$
$$2127 + 6380 x + 8631 x^2 + 221 x^3 + 6177 x^4 + 5106 x^5\}\}$$

Note that it is here where the coefficient ring is specified. We could instead generate a random integer matrix and work over the integers to find the Hermite form, although as mentioned above that is not a terribly efficient way to obtain it.

```
m2 = RandomInteger[{−100, 100}, {10, 15}];
Timing[hnf2 = groebnerHNF[m2, Integers];]
```

```
{0.23 Second, Null}
```

Indeed, what we did above is by no means the most efficient way to obtain the Hermite form of a matrix of polynomials. Several tactics for obtaining good computational efficiency are discussed in [Storjohann 1994]. At the expense of a fair amount of code one could adapt some of them to work in this Gröbner basis method. Some experimentation indicates that coefficient swell can be a serious problem when working with polynomials over the rationals and so the above method appears to be much more effective when working with polynomials over a prime field.

We adapt the technology in the previous example to solve linear polynomial diophantine systems. To solve such a system we transpose the matrix, prepend the right hand side vector, augment on the right with an identity matrix, and take the Hermite normal form. We find the row corresponding to the right hand side, check that it was multiplied, if at all, by a unit. When this is the case the solution vector can be taken from the rest of that row (which corresponds to multiples of columns of the original matrix that were needed to zero the right hand side) multiplied by the negative reciprocal of that unit. Null vectors come from later rows in the Hermite normal form and we return those as well. Note that this is readily adapted to handle a system of modular congruences. We simply treat the modulus in each congruence as something to be multiplied by a new variable, hence each gets a new row. As we are not interested in the specific multiple, we do not enlarge the identity matrix by which we augment, but instead add zero rows to join to the new rows necessitated by these moduli.

This method of diophantine solving may be found e.g. in [Blankenship 1966]. While a recent method works over the fraction field [Mulders and Storjohann 1999; Malaschonok 2001] and tends to be more efficient, this application of the Hermite normal form is all the same quite nice and very simple to code.

The tactic of augmenting with an identity matrix, well known e.g. for matrix inversion, is a form of "tag variable" manipulation in Gröbner basis technology. It can be used, for example, to record syzygies or conversion matrices using nothing beyond a standard `GroebnerBasis` function. The method appears in [Caboara and Traverso 1998] and was also discussed in [Lichtblau 1998] (the relevant conferences were indeed only days apart).

```
systemSolve[mat_?MatrixQ, rhs_?VectorQ, dom_, mod_: 0, moduli_: {}] /;
    Length[rhs] == Length[mat] :=
  Module[{newmat, modrows, hnf, j = 1, len = Length[mat], zeros, solvec, nullvecs},
    newmat = Prepend[Transpose[mat], rhs];
    newmat = Transpose[Join[Transpose[newmat], IdentityMatrix[Length[newmat]]]];
    If[moduli ≠ {},
      modrows =
        Table[If[j == k, moduli[[j]], 0], {j, Length[moduli]}, {k, Length[newmat[[1]]]}];
      newmat = Join[newmat, modrows]];
    hnf = groebnerHNF[newmat, dom, mod];
    zeros = Table[0, {len}];
    While[j ≤ Length[hnf] && Take[hnf[[j]], len] =!= zeros, j++];
    solvec = Drop[hnf[[j]], len + 1] / −hnf[[j, len + 1]];
    nullvecs = Map[Drop[#, len + 1] &, Drop[hnf, j]];
    {solvec, nullvecs}]
```

For this example we use a 3x5 matrix of polynomials in x of degree at most 3. Again we will work modulo 8933.

```
randomSystem[degmax_, rows_, cols_, var_] :=
  {randomMatrix[degmax, rows, cols, var], Table[randomPolynomial[degmax, var], {rows}]}
SeedRandom[11 111];
mod = 8933;
{mat, rhs} = randomSystem[3, 4, 6, x];
Timing[{sol, nulls} = systemSolve[mat, rhs, Polynomials[x], mod];]

{0.057991 Second , Null }
```

We check the result. The matrix times the solution vector must give the right hand side, and the matrix times the null vectors must give zeroes.

```
zeroTensor[t_] := Max[Abs[t]] == 0
{zeroTensor[Expand[mat.sol − rhs, Modulus → mod]],
zeroTensor[Expand[mat.Transpose[nulls], Modulus → mod]]}

{True , True }
```

We now show an example for the integer case that comes from [Dolzmann and Sturm 2001]. We have a system of six modular congruences in six variables that we wish to satisfy, with coefficient matrix, right hand side, and moduli as below.

```
mat = {{70, 0, 6, 89, 0, 7}, {87, 93, 78, 73, 0, 0}, {0, 87, 0, 0, 41, 0},
    {0, 12, 37, 69, 0, 15}, {75, 0, 90, 65, 14, 0}, {0, 0, 0, 0, 91, 96}};
rhs = {−30, −53, −3, −53, −41, −55};
moduli = {280, 5665, 110, 1545, 3125, 1925};
Timing[{soln, nulls} = systemSolve[mat, rhs, Integers, 0, moduli]]

{0.13 Second , {{0 , −2 , 4 , 12 802 , −29 779 , −34 696 },
  {{5 , 0 , 0 , −18 165 , 4400 , 333 025 },
   {0 , −5 , 0 , −16 135 , 26 475 , 445 025 }, {0 , 0 , 15 , 17 755 ,
     −26 950 , 540 925 }, {0 , 0 , 0 , 39 655 , 4950 , −594 825 },
   {0 , 0 , 0 , 0 , 68 750 , 0 }, {0 , 0 , 0 , 0 , 0 , −1 586 200 }}}}
```

We check that the solution indeed satisfies the congruences, and that matrix times null vectors gives zero vectors modulo the congruence moduli.

```
{zeroTensor[Mod[mat.soln − rhs, moduli]],
  zeroTensor[Mod[mat.Transpose[nulls], moduli]]}
```

```
{True ,   True }
```

In addition to being faster (though slow in comparison to what one can do with specialized Hermite normal form algorithm over the integers as in [Storjohann 1994]), the Hermite form method we use has the advantage that it gives a smaller solution, with components of 5 digits as compared to 12 in [Dolzmann and Sturm 2001]. Moreover it provides the null vectors, and we can attempt to add multiples of them to the solution in order to obtain a solution that is smaller still. We do this by forming a matrix comprised of the solution and null vectors. We augment by prepending one column containing zeroes in the null vector rows and a suitably chosen integer to act as an "anchor" in the row containing the original solution vector. We then apply fast lattice reduction [Lenstra, Lenstra, and Lovácz 1982]. The purpose of the anchor is to prevent the solution vector from being multiplied by anything other than a unit, and we check after reduction whether this succeeded. If so, the new solution is obtained from the remaining entries in the row containing the anchor (there may be more than one such, in which case the first will be smallest). The code below does all this, returning the original solution if it fails in the attempt to find something that involves only a unit multiple of that original solution vector.

```
smallSolution[sol_?VectorQ, nulls_?MatrixQ] :=
  Module[{max, dim = Length[nulls] + 1, weight, auglat, lat, k, soln},
    lat = Prepend[nulls, sol];
    max = Max[Flatten[Abs[lat]]];
    weight = dim max^2;
    auglat  = Map[Prepend[#, 0] &, lat] ;
    auglat[[1, 1]] = weight;
    lat = LatticeReduce[auglat];
    For[k = 1, lat[[k, 1]] == 0, k++];
    soln = lat[[k]];
    Which[
      soln[[1]] == weight, Drop[soln, 1],
      soln[[1]] == −weight, −Drop[soln, 1],
      True, sol]]
```

We obtain our new solution and again check that it satisfies the desired congruences.

```
Timing[newsol = smallSolution[soln, nulls]]
zeroTensor[Mod[mat.newsol − rhs, moduli]]
```

```
{0.01  Second ,   {565 ,   358 ,   −326 ,   227 ,   21 ,   −221 }}

True
```

This method, when used with more powerful technology for computing the Hermite form, will readily handle much larger problems, and moreover works well over the Gaussian integers. In the example below we use code in `systemSolve2` that is substantially identical to that shown above in `systemSolve`. We use `Developer‘HermiteNormalForm[[2]]` (extracting the second element because the first is the transformation matrix) instead of groebnerHNF as the former is specialized for working over (rational or Gaussian) integers.

```
SeedRandom[1111];
mat = RandomInteger[{−100, 100}, {20, 25}] + ⅈ RandomInteger[{−100, 100}, {20, 25}];
rhs = RandomInteger[{−100, 100}, 20] + ⅈ RandomInteger[{−100, 100}, 20];
```

```
Timing[{soln, nulls} = systemSolve2[mat, rhs];]
{zeroTensor[mat.soln − rhs], zeroTensor[mat.Transpose[nulls]]}
```

```
{0.475928  Second ,   Null  }

{True ,   True }
```

```
Timing[smallsoln = smallSolution[soln, nulls];]
zeroTensor[mat.smallsoln − rhs]
```

```
{0.308954  Second ,   Null }

True
```

We check that the new solution is indeed much smaller than the original.

```
{Max[Abs[N[soln]]], Max[Abs[N[smallsoln]]]}
```

$\{3.98003 \times 10^{47}, \quad 3.91379 \times 10^{9}\}$

So the initial solution had elements with up to 48 digits whereas those in the small solution do not exceed 10 digits.

We remark that a related method for finding a small solution is presented in [Matthews 2001]. It also uses Hermite normal form computation to obtain a solution vector as part of the transformation matrix, but attempts to enforce small size in that matrix via an implementation based on lattice reduction. A simpler form of what we showed above (anchor set to 1) has been referred to as the "embedding" technique in [Nguyen 1999]. It is not clear where it originated (the version shown above was first coded in 1995) and it seems to have been independently discovered multiple times. It should also be noted that the code shown above can be improved. In the Hermite solver there is no need to augment with an identity matrix because the transformation matrix will contain the necessary information. In finding small solutions, one will typically want to iterate the above method, reducing the size of the weight as the solution vectors get progressively smaller. Moreover it often works better if the null vectors are first reduced.

8 Application: Reduction of polynomial lattices

In this section we take leave of Gröbner bases over Euclidean domains and instead work in the more familiar terri− tory of computations over fields. The reason is that the Hermite normal form algorithm shown above, with but minor alteration, gives us a means of finding reduced lattices for univariate polynomial matrices; reduction here is in the sense of [Lenstra 1985]. The idea is to use the polynomial variable as an ideal variable (as opposed to retaining it in the coefficient structure), and compute a degree−based basis for the module. The code below will do exactly this.

```
polynomialLatticeReduce[mat_?MatrixQ, mod_: 0] := Module[
   {len = Length[First[mat]], newvars, generators, mgb},
   newvars = Array[v, len];
   generators = mat.newvars;
   mgb = moduleGroebnerBasis[generators, Variables[mat], newvars, CoefficientDomain →
          Rationals, Modulus → mod, MonomialOrder → DegreeReverseLexicographic];
   Outer[D, Reverse[mgb], newvars]]
```

We will again generate a random matrix, this time will all entries of fixed degree.

```
randomMatrix[degmax_, rows_, cols_, var_] := Table[
    randomPolynomial[degmax, var], {rows}, {cols}]
SeedRandom[1111]
mat = randomMatrix[4, 3, 5, x]
```

$$\{\{7+9x+8x^2+8x^3-4x^4,$$
$$-10-4x+x^2+4x^3-9x^4,\ -9-10x-7x^2-10x^3-10x^4,$$
$$-2x-9x^2+6x^3-4x^4,\ -7+7x-7x^2+3x^3+10x^4\},$$
$$\{7-6x^2+4x^4,\ 3+x-4x^2-8x^3-x^4,\ 8-2x+8x^2-5x^3+5x^4,$$
$$-5-6x+5x^2+4x^3-8x^4,\ -6-4x+2x^2+3x^3-6x^4\},$$
$$\{-10+9x+7x^2+x^3+4x^4,\ -7-9x-x^2-5x^3-8x^4,$$
$$7+5x^2-x^3+6x^4,\ 5-6x+8x^2+2x^3-10x^4,\ 7+3x-4x^3-6x^4\}\}$$

We begin by computing the Hermite form, as this is in some sense as "far" as possible from "reduced" (as deter− mined by orthogonality defect from [Lenstra 1985]).

```
Timing[hnf = groebnerHNF[mat, Polynomials[x], 8933]]
```

$$\begin{aligned}
&\{0.03\ \text{Second},\\
&\{\{1, 0, 6315 + 2935\,x + 8883\,x^{2} + 5385\,x^{3} + 4550\,x^{4} + 808\,x^{5} + 2173\,x^{6} +\\
&\quad 7678\,x^{7} + 6828\,x^{8} + 6950\,x^{9} + 7034\,x^{10} + 5482\,x^{11},\\
&\quad 4265 + 3896\,x + 7717\,x^{2} + 8361\,x^{3} + 5693\,x^{4} + 456\,x^{5} + 1435\,x^{6} +\\
&\quad 1512\,x^{7} + 735\,x^{8} + 2310\,x^{9} + 4026\,x^{10} + 5613\,x^{11},\\
&\quad 1823 + 6912\,x + 1669\,x^{2} + 7758\,x^{3} + 6345\,x^{4} + 5105\,x^{5} + 5740\,x^{6} +\\
&\quad 3596\,x^{7} + 4251\,x^{8} + 697\,x^{9} + 6128\,x^{10} + 5919\,x^{11}\},\\
&\{0, 1, 7877 + 2252\,x + 8201\,x^{2} + 6977\,x^{3} + 2172\,x^{4} + 5467\,x^{5} +\\
&\quad 600\,x^{6} + 5158\,x^{7} + 1063\,x^{8} + 6803\,x^{9} + 3165\,x^{10} + 4686\,x^{11},\\
&\quad 7076 + 5399\,x + 3436\,x^{2} + 1847\,x^{3} + 2955\,x^{4} + 8048\,x^{5} + 7613\,x^{6} +\\
&\quad 3262\,x^{7} + 4634\,x^{8} + 2128\,x^{9} + 3772\,x^{10} + 6069\,x^{11},\\
&\quad 8016 + 5639\,x + 590\,x^{2} + 1676\,x^{3} + 8248\,x^{4} + 2610\,x^{5} + 5491\,x^{6} +\\
&\quad 4410\,x^{7} + 5683\,x^{8} + 7750\,x^{9} + 7900\,x^{10} + 1253\,x^{11}\},\\
&\{0, 0, 6959 + 1101\,x + 957\,x^{2} + 7105\,x^{3} + 2752\,x^{4} + 3926\,x^{5} +\\
&\quad 448\,x^{6} + 5348\,x^{7} + 3571\,x^{8} + 2178\,x^{9} + 8427\,x^{10} + 7895\,x^{11} + x^{12},\\
&\quad 6450 + 3168\,x + 5859\,x^{2} + 4822\,x^{3} + 1719\,x^{4} + 8668\,x^{5} + 3480\,x^{6} +\\
&\quad 1789\,x^{7} + 4848\,x^{8} + 6061\,x^{9} + 2487\,x^{10} + 8928\,x^{11} + 1391\,x^{12},\\
&\quad 3374 + 5261\,x + 6349\,x^{2} + 1682\,x^{3} + 8563\,x^{4} + 7676\,x^{5} + 4861\,x^{6} +\\
&\quad 8733\,x^{7} + 245\,x^{8} + 2641\,x^{9} + 5566\,x^{10} + 4165\,x^{11} + 2183\,x^{12}\}\}\}
\end{aligned}$$

```
Timing[redlat = polynomialLatticeReduce[hnf, 8933]]
```

$\{0.04\ \text{Second},$
$\{\{2888 + 7592\,x + 8382\,x^2 + 7495\,x^3 + x^4,\ 7000 + 6901\,x + 1933\,x^2 + 840\,x^3,$
$151 + 6746\,x + 4817\,x^2 + 7687\,x^3,\ 2228 + 4662\,x + 7543\,x^2 +$
$4470\,x^3 + 7242\,x^4,\ 935 + 3521\,x + 1489\,x^2 + 8540\,x^3 + 8833\,x^4\},$
$\{7545 + 3572\,x + 8733\,x^2 + 5161\,x^3,\ 4170 + 5758\,x + 4764\,x^2 + 4367\,x^3 + x^4,$
$6551 + 8139\,x + 3375\,x^2 + 1985\,x^3,\ 8932 + 5757\,x + 4367\,x^2 + 199\,x^4,$
$2778 + 6153\,x + 2978\,x^2 + 6353\,x^3 + 1588\,x^4\},$
$\{8133 + 7147\,x + 7546\,x^2 + 396\,x^3,\ 2381 + 4564\,x + 6552\,x^2 + 3773\,x^3,$
$1191 + 3377\,x + 5756\,x^2 + 4966\,x^3 + x^4,\ 3 + 4567\,x + 3773\,x^2 +$
$8931\,x^3 + 1391\,x^4,\ 1593 + 7346\,x + 2978\,x^2 + 8732\,x^3 + 2183\,x^4\}\}\}$

Notice that we have a sort of reversal of roles from [Basiri and Faugere 2003]. In that paper they use reduction of a polynomial matrix to compute a Gröbner basis whereas we do quite the opposite. These are not mutually exclusive, however, and in principle the method of reduction above could lie beneath their algorithm (in other words, we are bootstrapping rather than coding in circles).

It should also be noted that, as was shown with the integer case above, this lattice reduction might be put to use to find "small" (that is, low degree) solutions to diophantine polynomial systems with nontrivial null spaces.

9 Application: Bivariate modular polynomial factorization

We now put together techniques from the preceding applications sections for the purpose of factoring a bivariate polynomial modulo a prime. We will generate a pair of random polynomials such that there are terms of highest total degree in each variable separately; this convenience involves no actual loss of generality, as one can always attain this for one variable by a linear change of coordinates. We make a few other useful choices so as not to run afoul of necessary conditions e.g. degree changing on substitution of a value for one variable. Again, these are all conve−niences insofar as one can work in an extension field in one variable, in essence performing a substitution of an algebraic element outside the base field. The purpose of this section is not to derive a foolproof algorithm but rather to illustrate the method on a relatively simple albeit nontrivial example.

```
randpoly[deg_, mod_, x_, y_] :=
```
$$\sum_{i=0}^{\text{deg}} \sum_{j=0}^{\text{deg}-i} \text{RandomInteger}[\{\text{If}[i + j == \text{deg} \,\&\&\, i\, j == 0, 1, 0], \text{mod} - 1\}]\, x^i\, y^j$$

```
mod = 19; SeedRandom[1111]; totdeg = 6; poly1 = randpoly[totdeg/2, mod, x, y];
  poly2 = randpoly[totdeg/2, mod, x, y]; poly = Expand[poly1 poly2, Modulus → mod]
```

$$18\,x + 6\,x^2 + 18\,x^3 + 16\,x^4 + 12\,x^5 + 8\,x^6 + 13\,y + 12\,x y + 17\,x^2\,y + 11\,x^3\,y + 7\,x^4\,y + 16\,x^5\,y + 16\,y^2 + 17\,x y^2 + 6\,x^2\,y^2 + 2\,x^3\,y^2 + 14\,x^4\,y^2 + 14\,y^3 + 18\,x y^3 + 5\,x^2\,y^3 + 2\,x^3\,y^3 + y^4 + 4\,x y^4 + 16\,x^2\,y^4 + 18\,y^5 + 18\,x y^5 + 7\,y^6$$

We will evaluate at $x = 11$ and factor, removing the constant term.

```
fax = Map[First, Drop[FactorList[poly /. x → 11, Modulus → mod], 1]]
```

$$\left\{8 + y,\ \ 14 + 3\,y + y^2,\ \ 14 + 13\,y + 9\,y^2 + y^3\right\}$$

As in the Hensel lifting section we will lift a factor modulo a power of the ideal $(x - 11)$ that is sufficient to reclaim factors of degree 3 in x.

```
subst = (x − 11);
pow = 12;
substpower = subst ^ pow;
liftedfactor = Last[GroebnerBasis[{poly, substpower, fax[[1]] ^ pow},
    y, Modulus → mod, CoefficientDomain → Polynomials[x]]]
```

$$3 + 14\,x + 2\,x^2 + 15\,x^3 + 2\,x^4 + 15\,x^5 + 18\,x^6 + 18\,x^7 + 18\,x^8 + 13\,x^9 + 5\,x^{10} + x^{11} + y$$

We remark that one must exercise care at this last step, insofar as a minor reformulation may accidentally recover the entire factor, thus rendering moot the rest of the example. To wit:

```
liftedfactor2 = First[GroebnerBasis[{poly, (x − 11) ^ 10, fax[[1]] ^ 10},
    {x, y}, Modulus → mod, MonomialOrder → DegreeReverseLexicographic]]
```

$$12\,x + 4\,x^2 + x^3 + 15\,y + 2\,x y + 11\,x^2\,y + 5\,x y^2 + 5\,y^3$$

```
PolynomialMod[5 * poly2, mod] == liftedfactor2

True
```

This, however, is another method and will not be discussed further in the present paper. It at least serves to show us the goal for the method at hand.

As in [Lenstra 1985] we now set up a lattice in univariate polynomials in x.

```
deg = Exponent[liftedfactor, y];
lattice1 = Table[If[i == j, substpower, 0], {i, deg}, {j, totdeg - 2}];
coeffs = PadRight[CoefficientList[liftedfactor, y], totdeg - 2];
lattice2 = Table[RotateRight[coeffs, j], {j, 0, totdeg - 3 - deg}];
lattice = Join[lattice1, lattice2]
```

$$\{\{(-11+x)^{12}, 0, 0, 0\},$$
$$\{3+14x+2x^2+15x^3+2x^4+15x^5+18x^6+18x^7+18x^8+13x^9+5x^{10}+x^{11}, 1, 0, 0\},$$
$$\{0, 3+14x+2x^2+15x^3+2x^4+15x^5+18x^6+18x^7+18x^8+13x^9+5x^{10}+x^{11}, 1, 0\},$$
$$\{0, 0, 3+14x+2x^2+15x^3+2x^4+15x^5+18x^6+18x^7+18x^8+13x^9+5x^{10}+x^{11}, 1\}\}$$

```
First[redlat = polynomialLatticeReduce[lattice, mod]].y^Range[0, totdeg/2]
```

$$12x+4x^2+x^3+\left(15+2x+11x^2\right)y+5xy^2+5y^3$$

We recognize from this that we have recovered one of the true modular factors of our original polynomial.

10 Application: Computing small generators of ideals in quadratic number rings

A ring of integers extended by a square root is an important object in number theory. Say d satisfies $d^2 = D$, where D is a squarefree integer. Two elements of the quadratic integer ring $\mathbb{Z}[d]$, say $x = r + s\,d$ and $y = u + v\,d$, comprise the basis of an ideal. We will compute small generators for that ideal. We can moreover recover Bezout relations. That is, we find a pair of multipliers $\{m, n\} \in \mathbb{Z}[d]$ such that $m\,x + n\,y = g$ for each such generator g.

Here we use code from previous sections to provide multipliers for the Bezout relations. We compute a module basis with first column comprised of our given x and y, and a 2×2 identity matrix to the right of that column. We further−more have a 3×3 matrix beneath this, comprised of the reducing quadratic on the diagonal and zero elsewhere. The Hermite form of this matrix, computed via `groebnerHNF`, will have as first row the greatest common divisor and the Bezout relation multipliers. For full generality, we handle the case where $d^2 \equiv_4 1$ by instead using the defining polynomial $\left((2\,d-1)^2 - D\right)/4$; this allows us the full range of elements in the corresponding quadratic ring. The division by 4, which would be superfluous were our coefficient domain a field, is necessary for attaining a monic defining polynomial.

There is an added wrinkle. The Bezout relation multipliers computed as above can be quite large. But we can find a smaller set, exactly as we found small integer solutions to diophantine systems. We simply treat the quadratic integers as integer pairs, flatten our vectors of these, and invoke `smallSolution`. Then we translate consecutive pairs of the resulting integer vector back to quadratic integers. Code for this is below.

```
quadraticIntegerToIntegerVector[n1_Integer, alg_] := {n1, 0}
quadraticIntegerToIntegerVector[n1_. + n2_.∗alg_, alg_] := {n1, n2}
quadraticVectorToIntegerVector[vec_, alg_] :=
  Flatten[Map[quadraticIntegerToIntegerVector[#, alg] &, vec]]

smallSolutionQuadratic[vec_, nulls_, alg_] := Module[
    {soln, nulls2},
    soln = quadraticVectorToIntegerVector[vec, alg];
    nulls2 = Map[quadraticVectorToIntegerVector[#, alg] &, nulls];
    soln = smallSolution[soln, nulls2];
    Partition[soln, 2] /. {a_Integer, b_Integer} :> a + alg∗b
  ]
```

Here is the Bezout relation code.

```
bezout[d_, m1_Integer, m2_. + n2_.∗d_, tsqr_] :=
  Module[{theta, polys}, polys = {m1, m2 + n2∗theta};
    polyBezout[polys, theta, d, tsqr]]
bezout[d_, m2_. + n2_.∗d_, m1_Integer, tsqr_] :=
  Module[{theta, polys}, polys = {m2 + n2∗theta, m1};
    polyBezout[polys, theta, d, tsqr]]
bezout[d_, m1_. + n1_. d_, m2_. + n2_.∗d_, tsqr_] :=
  Module[{theta, polys}, polys = {m1 + n1∗theta, m2 + n2∗theta};
    polyBezout[polys, theta, d, tsqr]]

polyBezout[polys_, theta_, d_, tsqr_] :=
  Module[{defpoly, mat, gb, gcd, solns, soln, nulls, relations, subs},
    defpoly = If[Mod[tsqr, 4] == 1, subs = theta → (1 + Sqrt[tsqr])/2;
        Expand[((2∗theta − 1)^2 − tsqr)/4], subs = theta → Sqrt[tsqr]; theta^2 − tsqr];
    mat = Join[Transpose[Join[{polys}, IdentityMatrix[2]]], defpoly∗IdentityMatrix[3]];
    gb = groebnerHNF[mat, Integers];
    relations = Select[gb, #[[1]] =!= 0 && FreeQ[#[[1]], theta^_] &];
    solns = Map[Rest, relations];
    nulls = Map[Rest, Cases[gb, {0, __}]];
    nulls = DeleteCases[nulls, vec_ /; ! FreeQ[vec, theta^_]];
    solns = Map[smallSolutionQuadratic[#, nulls, theta] &, solns];
    Partition[Riffle[Map[First, relations], solns] /. subs, 2]]
```

We show a quick example. We'll work over $\mathbb{Z}[\sqrt{-19}]$ (so $d = (1+\sqrt{-19})/2$, with inputs $51 + 43\,d$ and $26 - 55\,d$.

```
bezrels = bezout[d, 51 + 43 d, 26 − 55 d, −19]
```

$$\left\{\left\{1\ ,\ \left\{115\ -2\ \left(1\ +i\sqrt{19}\ \right),\ \ 101\ -17\ \left(1\ +i\sqrt{19}\ \right)\right\}\right\}\right\}$$

We now check that result by expanding to see we recover the claimed gcd.

```
Expand[bezrels[[1, 2]].{51 + 43 (1 + Sqrt[−19]) / 2, 26 − 55 (1 + Sqrt[−19]) / 2} − bezrels[[1, 1]]]
```

```
0
```

We now show a larger example.

```
n = 50;
randsqrt = √RandomInteger[10^n] /. a_Integer √b_ → √b
randqints = {RandomInteger[10^n, 2].{1, d}, RandomInteger[10^n, 2].{1, d}}

√76 645 210 216 068 275 341 252 449 427 250 942 042 565 641 503 899
{41 230 119 139 644 742 056 691 832 704 420 484 800 325 317 097 349  +
    66 637 694 434 836 005 939 093 652 315 806 699 696 521 110 837 633  d,
  29 376 606 053 454 810 686 236 077 314 995 219 308 578 051 470 644  +
    68 102 540 162 537 581 922 579 541 354 979 200 541 266 074 057 948  d}

Timing[bezrels = bezout[d, Sequence @@ randqints, randsqrt^2]]

{0.013998 ,  {{3  + √76 645 210 216 068 275 341 252 449 427 250 942 042 565 641 503 899  ,
     {62 910 542 074 600 312 551 765 960 503 628 547 531 246 233 643 820 636 292 754 943 253 400 ⋱
          066 320 117 637 426 498 520 034 683 710  –
        31 136 862 355 224 409 625 896 210 047 184 165 981 450 145 584 808 340 049 705 245 902 612 ⋱
            582 352 889 642 233 951 439 580 078 105
          √76 645 210 216 068 275 341 252 449 427 250 942 042 565 641 503 899  ,
      –55 848 984 550 900 338 117 246 064 954 825 208 020 230 922 492 945 275 734 518 302 470 297 ⋱
            813 449 754 151 001 095 645 110 160 958  +
        30 467 126 693 584 986 302 978 558 640 228 834 191 641 712 044 290 348 730 028 225 317 883 ⋱
            677 987 057 268 184 365 890 747 403 850
          √76 645 210 216 068 275 341 252 449 427 250 942 042 565 641 503 899  }},  {30 ,
     {–60 305 611 400 261 823 127 930 569 882 412 860 470 856 519 992 038 292 070 000 965 250 335 ⋱
            473 959 556 066 003 160 005 410 829 435  +
        29 847 581 335 270 731 904 742 643 882 576 386 427 147 936 374 019 427 518 903 539 178 803 ⋱
            045 332 043 535 642 926 090 923 373 923
          √76 645 210 216 068 275 341 252 449 427 250 942 042 565 641 503 899  ,
      53 536 451 099 594 498 510 472 658 242 785 071 456 611 736 153 058 906 900 507 263 017 699 ⋱
          342 307 286 854 677 383 630 536 832 643  –
        29 205 577 352 793 031 992 930 235 827 933 444 488 853 979 241 234 764 625 005 127 503 982 ⋱
            830 800 303 475 548 423 181 673 182 794
          √76 645 210 216 068 275 341 252 449 427 250 942 042 565 641 503 899  }}}}
```

In the special case where the ideal is {1}, (e.g. {*x*, *y*} generates the entire ring), then we actually have obtained an extended gcd.More generally it is easy to show that when there is a gcd and it is a rational integer, or else there is no rational integer in the ideal, then the above code finds that gcd and the corresponding Bezout relation. In other cases one would need to do further work to either recover a gcd or else (in the case where the class number of the quadratic ring is not 1) show that no gcd exists. See [Weilert 2005] or [Miled andOuertani 2010] for further details. We show below a simple approach that works in many situations. We work with $d = \sqrt{53\,719}$.

```
sqrt = Sqrt[53 719];
qints = {73 609 + 15 577 d, 2991 + 6417 d};
```

```
bezrels = bezout[d, Sequence @@ qints, sqrt^2]
```

```
{{1  + √53 719 ,
   {14 462 895  − 3 265 382 √53 719 ,   −1 344 274  + 7 923 502 √53 719 }},
  {6 ,  {1 128 711  − 254 907 √53 719 ,   −104 180  + 618 536 √53 719 }}}
```

We have reduced the ideal sum to one generated by $(1 + \sqrt{53\,719}, 6)$. There is in fact a gcd, though, because the ring $\mathbb{Z}[\sqrt{53\,719}]$ is a principal ideal domain (this follows from the fact that $\sqrt{53\,719}$ has class number of 1). Checking norms shows that any common factor of 6 and $1 + \sqrt{53\,719}$ will have a norm of 6 or −6. So we have a Pell type of equation to solve: find integers $\{a, b\}$ such that $(6\,a + b)^2 - 53\,719\,b^2 = \pm 6$. Well known facts about such equations tell us that any solution will have $(6\,a + b)/b$ as a convergent to the continued fraction expansion of $\sqrt{53\,719}$. We remark that for this method to work, we require that the right hand side, ± 6 in this case, have absolute value less than $\sqrt{53\,719}$.

```
cf = ContinuedFraction[√53 719 ];
frax = Convergents[cf];
solns1 = Table[Solve[{(6*a + b)^2 − 53 719*b^2 == 6, (6*a + b)/b == frax[[j]]}, {a, b}],
     {j, Length[frax]}];
solns2 = Table[Solve[{(6*a + b)^2 − 53 719*b^2 == −6, (6*a + b)/b == frax[[j]]}, {a, b}],
     {j, Length[frax]}];
```

```
Cases[Flatten[{a, b} /. solns1, 1], {x_Integer, y_Integer}]
Cases[Flatten[{a, b} /. solns2, 1], {x_Integer, y_Integer}]
```

```
{}
```

```
{{−3 428 948 410 941 086 922 003 618 340 136 587 439 827 999 302 403 486 984 497 710 688 926 584 615 ⋱
      684 288 776 613 161 602 832 ,
   −89 150 972 140 308 509 130 356 945 221 982 374 902 233 029 263 834 884 255 241 520 937 582 267 ⋱
      870 954 684 338 850 528 243 },
  {3 428 948 410 941 086 922 003 618 340 136 587 439 827 999 302 403 486 984 497 710 688 926 584 615 ⋱
     684 288 776 613 161 602 832 ,
   89 150 972 140 308 509 130 356 945 221 982 374 902 233 029 263 834 884 255 241 520 937 582 267 ⋱
     870 954 684 338 850 528 243 }}
```

We have found a gcd.

```
gcd = Expand[
   {3 428 948 410 941 086 922 003 618 340 136 587 439 827 999 302 403 486 984 497 710 688 926 584 ⋱
       615 684 288 776 613 161 602 832,
      89 150 972 140 308 509 130 356 945 221 982 374 902 233 029 263 834 884 255 241 520 937 582 ⋱
       267 870 954 684 338 850 528 243}.{6, 1 + √53 719 }]
```

```
20 662 841 437 786 830 041 152 066 986 041 507 013 870 228 843 684 756 791 241 505 654 497 089 961 ⋱
   976 687 344 017 820 145 235   +
  89 150 972 140 308 509 130 356 945 221 982 374 902 233 029 263 834 884 255 241 520 937 582 267 870 ⋱
     954 684 338 850 528 243   √53 719
```

We confirm that the norm is indeed −6.

```
Expand[gcd * (gcd /. √53719 -> -√53719)]
```

```
-6
```

11 Summary

We have presented an algorithm for computing a strong Gröbner basis over a Euclidean domain that is essentially identical to Buchberger's method for the case where the base ring is a field. In particular we have retained the S−polynomial reduction approach as well as the Buchberger criteria for elimination of redundant S−polynomials. Several basic examples were presented to illustrate diverse applications of this technology e.g. working over quotient rings, solving nonlinear systems over rings. As more specialized applications, we showed how to use these bases to compute Hensel lifts in a univariate polynomial ring and find matrix Hermite normal forms; closely related code gives us reduction of univariate polynomial lattices. While these last applications are quite specialized in the sense that efficient methods are available that do not require the full power of Gröbner bases, it is all the same nice to have the methods shown above for computing them. One reason is that the code is simple and fairly flexible should modifications be desired. Another is that these methods, while not as fast as the best known, perform reasonably well on many problems that are of practical size. Perhaps most interesting is that several fundamental ideas from com− puter algebra, such as Hensel lifting, matrix canonical forms, and lattice reduction, as well as interplay between these, may be cast as computations involving Gröbner bases over Euclidean domains or close relatives thereof.

12 Acknowledgements

I thank one anonymous refereee for pointing out several typographical errors in an earlier draft of this paper, another for drawing my attention to the related work by Norton and Sălăgean, Ana Sălăgean for providing a preprint of their work, and Michael Trott for drawing my attention to the preprint by Dolzmann and Sturm.

13 References

[1] W. Adams and P. Loustaunau (1994). *An Introduction to Gröbner Bases*. Graduate Studies in Mathematics **3**. American Mathematical Society.

[2] A. Aho, J. Hopcroft, and J. Ullman (1974). *The Design and Analysis of Computer Algorithms*. Addison−Wesley Publishing Company.

[3] A. Basiri and J−C Faugère (2003). Changing the ordering of Gröbner bases with LLL: case of two variables. In: Proceedings of the 2003 International Symposium on Symbolic and Algebraic Computation. J. R. Sendra, ed. 23−29. ACM Press.

[4] T. Becker, W. Weispfenning, and H. Kredel (1993). *Gröbner Bases*: *A Computational Approach to Computer Algebra*. Graduate Texts in Mathematics **141**. Springer−Verlag.

[5] K. Belabas, G. Hanrot, and P. Zimmerman (2001). Tuning and generalizing van Hoeij's algorithm. INRIA Research report 4124, February 2001. Available in electronic form at:
http://www.loria.fr/~zimmerma/papers/

[6] W. Blankenship (1965). Algorithm 288: Solution of simultaneous linear diophantine equations. Communications of the ACM **9**(7):514.

[7] B. Buchberger (1985). Gröbner bases: An algorithmic method in polynomial ideal theory. In *Multidimensional Systems Theory*, chap 6. N. K. Bose, ed. D. Reidel Publishing Company.

[8] M. Caboaro and C. Traverso (1998). Efficient algorithms for ideal operations (extended abstract). In: Proceedings of the 1998 International Symposium on Symbolic and Algebraic Computation. O. Gloor, ed. 147−152. ACM Press.

[9] S. Collart, M. Kalkbrenner, and D. Mall (1997). Converting bases with the Gröbner walk.. Journal of Symbolic Computation **24**(3−4):465−469.

D. Cox, J. Little, and D. O'Shea (1992). *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Computer Algebra*. Undergraduate Texts in Mathematics. Springer−Verlag.

[11] A. Dolzmann and T. Sturm (2001). Solving Systems of Linear Congruences. Technical Report MIP−0101, FMI, Universität Passau, D−94030 Passau, Germany, February 2001. Available in electronic form at:
http://www.fmi.uni−passau.de/~sturm/publications/techreports/

[12] J. von zur Gathen and J. Gerhard (1999). *Modern Computer Algebra*. Cambridge University Press.

[13] D. Grayson (1996). Private communication.

[14] M. van Hoeij (2002). Factoring polynomials and the knapsack problem. Journal of Number Theory **95**:167−181.

[15] A. Kandri−Rody and D. Kapur (1984). Computing the Gröbner basis of polynomial ideals over the integers. Proceedings of the Third MACSYMA Users' Conference, Schenectady, NY. 436−451.

[16] A. Kandri−Rody and D. Kapur (1988). Computing a Gröbner basis of a polynomial ideal over a Euclidean domain. Journal of Symbolic Computation **7**:55−69.

[17] A. Lenstra(1985). Factoring multivariate polynomials over finite fields. Journal of Computer and System Sciences **30**:235−248.

[18] A. Lenstra, H. Lenstra, and L. Lovácz (1982). Factoring polynomials with rational coefficients. Mathematische Annalen **261**:515−534.

[19] D. Lichtblau (1996). Gröbner bases in Mathematica 3.0. The *Mathematica* Journal **6**(4): 81−88.

[20] D. Lichtblau (1998). Talk title: "Practical computations with Gröbner bases". IMACS−ACA 1998, Prague. Abstract available in electronic form at:
http://www.math.unm.edu/ACA/1998/sessions.html

[21] G. Malaschonok (2001). Fast methods of a linear system solving in commutative domains. Preprint.

[22] K. R. Matthews (2001). Short solutions of $A X = B$ using a LLL−based Hermite normal form algorithm. Preprint.

[23] A. Miled and A. Ouertani (2010). Extended gcd of quadratic integers. arXiv:1002.4487v1 [cs.DM]. Also see:
http://demonstrations.wolfram.com/ExtendedGCDOfQuadraticIntegers/

[24] H. M. Möller (1988). On the construction of Gröbner bases using syzygies. Journal of Symbolic Computation **6**: 345−359.

[25] T. Mulders and A. Storjohann (1999). Diophantine linear system solving. In: Proceedings of the 1999 International Symposium on Symbolic and Algebraic Computation. Sam Dooley, ed. 181−188. ACM Press.

[26] P. Nguyen (1999). Cryptanalysis of the Goldreich−Goldwasser−Halevi cryptosystem from Crypto '97. Advances in Cryptology, Proceedings of CRYPTO 1999, Santa Barbara, CA. Available in electronic form at:
http://www.di.ens.fr/~pnguyen/pub.html#Ng99

[27] G. Norton and A. Sălăgean (2001). Strong Gröbner bases and cyclic codes over a finite−chain ring. Workshop on Coding and Cryptography, Paris 2001. Preprint.

[28] L. Pan (1989). On the D−bases of polynomial ideals over principal ideal domains. Journal of Symbolic Computation **7**: 81−88.

[29] C. Sims (1994). *Computations with Finitely Presented Groups*. Cambridge University Press.

[30] A. Storjohann (1994). Computation of Hermite and Smith Normal Forms of Matrices. Master's thesis, Department of Computer Science, University of Waterloo.

[31] A. Weilert (2005). Two efficient algorithms for the computation of ideal sums in quadratic orders. Mathematics of Computation 75(254):941−981.

[32] S. Wolfram (1999). *The Mathematica Book* (4th edition). Wolfram Media/Cambridge University Press.

[33] P. Zimmerman (2003). Polynomial Factorization Challenges: a collection of polynomials difficult to factor. Available in electronic form at:
http://www.loria.fr/~zimmerma/mupad/