International Journal of Algebra, Vol. 7, 2013, no. 8, 369 - 390
HIKARI Ltd, www.m-hikari.com

Applications of Strong Gröbner Bases over Euclidean Domains

Daniel Lichtblau

Wolfram Research
100 Trade Centre Dr
Champaign, IL 61820, USA
danl@wolfram.com

Abstract

Strong Gröbner bases over Euclidean domains and even more general rings were first defined in the 1980s. Since that time, efficient ways to compute them, and a variety of applications, have appeared. In this note we show, via simple examples, applications to solving equations in quotient rings, Hensel lifting, Hermite normal form computations, reduction of univariate polynomial lattices, and finding small generators in quadratic number rings.

Mathematics Subject Classification: 11C08, 11C20, 11D04, 11R11, 13B25, 13F07, 13P05, 13P10, 14Q99, 15A23, 15B36

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

1 Introduction

The theory and application of Gröbner bases of polynomials over fields has been developed extensively, and proven indispensible in many areas of computational mathematics. When going from a field as base ring to a more general ring, there are options for which properties of Gröbner bases one might hope to preserve. For

Euclidean domains as base rings there are both "weak" and "strong" Gröbner bases. The former have the benefit that they are easier to compute,. The cost is that reduction of polynomials modulo such a basis is now more expensive as compared to the working with strong bases. These bases are discussed in [8], [5] section 8, [9], [23], [3], [2]. These references also present various algorithms for computing either weak or strong bases. While all are based to some extent on the Buchberger algorithm over fields, there are qualifications that create substantial inefficiency as compared to the field case. An exception is [19], which in various respects is similar to the Buchberger algorithm over fields. That said, it is more specialized to the case of weak bases and also does not fully develop the redundancy criteria of the Buchberger algorithm.

In [16] we develop an efficient computation of Gröbner bases over base rings that are effectively computable Euclidean domains. After providing the relevant definitions and theory, an algorithm is presented that is very much in the spirit of the Buchberger algorithm for working over fields. In particular it uses S-polynomials, generalized division for reduction, and has criteria for eliminating pairs that are analogous to those for the field case. Our goal in this companion work is to provide a number of examples of both general and special cases. We will show how several important, and seemingly disparate, computations may be viewed as Gröbner bases over rings, and to point to practial applications along the way.

We will begin with basic examples similar to those in [16]. These recall to the reader some uses of extra variables for purposes of computing ideal intersections or working in extension rings that are not themselves Euclidean. We next show how such bases may be used to perform Hensel lifting of univariate polynomials factored over a prime modulus. We then cover computation of matrix Hermite normal forms, as well as Popov forms for matrices of univariate polynomials over a field (the polynomials themselves are now the base ring, with new variables corresponding to matrix rows). The next application we show is to bivariate modular polynomial factorization. We finish our examples with computation of small generators of ideals in quadratic number rings.

For purposes of brevity and simplicity of exposition we will omit most formal statements and proofs of correctness for the methods we present. Such proofs would use arguments based on term ordering and integer sizes, and are generally straightforward. Instead we will indicate why they work by reminding the reader of how Gröbner bases correspond to objects under study in the various applications.

All examples are run using *Mathematica* 9 [29] using a 3.1 GHz processor running on the Linux operating system. One indicates that computationa are to be performed over specific Euclidean domains (integers and univariate polynomial rings) via settings of the $CoefficientDomain$ option. We intersperse snippets of code throughout, and place larger code blocks in an appendix. That latter, in total, stills comes to fewer than 100 lines. This may give an indication of the ease of adapting such Gröbner bases to a wide variety of applications.

2 Basic Examples

We begin with an example adapted from [2]. We wish to compute a Gröbner basis for an ideal in the polynomial ring $\mathbb{Z}[\sqrt{-11}][x, y]$. Our base ring is not a unique factorization domain, hence not a Euclidean domain. We finesse this issue by adding a new variable and a corresponding defining polynomial to handle the quadratic extension. In this way we can work directly over the integers. So our ring will be $\mathbb{Z}[x, y, \alpha]/\{\alpha^2 + 11\}$. We want to treat the extension variable as though it were a coefficient. A well known way to achieve this end is to have the the new variable ordered lexicographically lower than all others. When finished we remove the defining polynomial for the extension.

```
Rest[GroebnerBasis[{2xy − αy, (1 + α)x^2 − xy, α^2 + 11},
  {x, y, α}, CoefficientDomain → Integers]]
```

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

As a second example, we will find a basis for the ideal intersection $\{3x^2 - 2y - 4, y^2 + 5y - 3xy + 2\} \cap \{16xy - 6\}$ 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 lexicographically greater than the others (specifying it as the third argument tells *GroebnerBasis* it is to be eliminated).

```
GroebnerBasis[Flatten[
  {w{3x^2 − 2y − 4, y^2 + 5y − 3xy + 2}, (1 − w){16xy − 6}}], {x, y}, w,
  CoefficientDomain → Integers, MonomialOrder → EliminationOrder]
```

$\{12 + 30y - 50xy + 6y^2 - 80xy^2 + 48x^2y^2 - 16xy^3,$
$24 - 18x^2 + 12y - 64xy + 48x^3y - 32xy^2,$
$-60 - 36x - 90y + 160xy + 96x^2y - 24y^2 + 240xy^2 - 6y^3 + 64xy^3 + 16xy^4,$
$-24 - 12x - 36y + 70xy + 32x^2y + 90xy^2 - 16x^2y^2 + 16x^2y^3\}$

Another application is to do computations involving ideals defined over quotient rings that may contain zero divisors. As an example we will show a key step, the basis computation, for finding all solutions in the ring $\mathbb{Z}_{5072012170009}$ to the system below.

```
gb = GroebnerBasis[{5072012170009, −4984359602099 + x^2 − 3y^2 − 9xz,
  −1780431462965 + 7xy + 5y^3 + z^2, −4585397367278 + x^3 − 3y^2 + z − 12z^3},
  {x, y, z}, CoefficientDomain → Integers]
```

$\{5072012170009, 1174872829454 + 12173501962z - 1363165624472z^2 + \ldots$
$-1977589465047z^{16} - 2210999439349z^{17}\}$

A related area of application for Gröbner bases over the integers is in computations with finitely presented groups, as discussed in chapter 10 of [24].

3 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 + 18x^4 + 34x^3 + 5x^2 + 21x + 30)
  (x^4 + 24x^3 + 22x^2 + 17x + 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]
```

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

Next we wish to make the factors correct modulo a power of the prime. This correction step is referred to as Hensel lifting [27] 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 along with 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
```

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

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 real question to be answered is why these Gröbner basis computations gave the quadratic Hensel lifts of the modular factors. We address this next.

Theorem 1. *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 $s = GCD[{g_0}^2, f]$ 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 encountered 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 $g_1 = g_0 + pt_0$. Then a simple computation shows that $g_1 (g_0 - pt_0) \equiv_{p^2} {g_0}^2$. We see that $g_1 \mid f$ and $g_1 \mid {g_0}^2$ modulo p^2. Now let $s = \mathrm{GCD}\,[{g_0}^2, f]$. 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 realize 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 comes from a factorization example presented in [26]. The polynomial in question is of degree 190 and has, as its roots, all the sums of pairs of roots of a simpler polynomial, of degree 20. Code to create it is provided in the appendix.

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 of modest size.

```
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. This lift can actually be done in one step (this can be shown similar to the proof of theorem 1 above; see also multivariate factorization in [15]). 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. The code for this is in the appendix.

```
Timing[liftedfax = liftfactors[fax, newpoly, mod, 36]; ]
{1.790000, Null}
```

We have attained a speed comparable to what was presented in [26] 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. We remark 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, it is clear that 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 it 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) [27], 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 [30]. As these typically required lifting to many digits, this is evidence of the practicality of the method above.

4 Computation and Use 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. 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. The code to create a random matrix of such polynomials is in the appendix.

SeedRandom[1111];
mat = randomMatrix[2, 3, 5, x]
$\{\{-1, -1+4x, 2+7x+9x^2, 4-6x-2x^2, 8-9x-3x^2\},$
$\{7-9x, -9-9x, -9+5x, -7+8x+9x^2, -2\},$
$\{-3+x+3x^2, 4, -4-9x, -7, -3\}\}$

To set this up we need to extend *Mathematica*'s *GroebnerBasis* to handle modules, using a "position over term" (POT) ordering [2]. 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, adapted from [12], may be found in the appendix.

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 other polynomial

variables). We convert each row of the matrix to a polynomial vector representation by making each column into a new "variable". Now we use the module Gröbner basis routine described above. Since we use a POT term order, the reduction works to first find a minimal element (according to the Euclidean valuation) in the first column, then in the second, and so forth. When finished we convert the result back to matrix form. We summarize this discussion as a simple theorem.

Theorem 2. *One can represent a diophantine system over a Euclidean domain (e.g. integers or univariate polynomials over a field) as a module over the domain, and via a module Gröbner basis computation one can then obtain the Hermite form of the matrix.*

We mention a minor departure from the standard version of the Hermite normal form. There is an issue of normalization of nonzero entries that lie above pivots (when we regard the lattice as being generated by rows, that is, compute the row-based Hermite form). The Gröbner basis method just described will do this differently in many cases from the forms described in the literature. This is both easily remedied after the fact if so desired, and also is of no consequence in terms of practical usage. We ignore the distinction and work with the result proved by the Gröbner basis.

As an example now we obtain our module basis over $\mathbb{Z}_{8933}[x]$ for the matrix above. We work over a prime field in order to restrict the size of the coefficients.

hnf = groebnerHNF[mat, Polynomials[x], 8933]
$\{\{1, 0, 1917 + 1070x + 3059x^2 + 8245x^3 + 8000x^4, 3765 + 4710x + 4646$
$x^2 + 7963x^3 + 5748x^4, 324 + 2015x + 8444x^2 + 6027x^3 + 311x^4\},$
$\{0, 1, 1679 + 4306x + 7219x^2 + 1156x^3 + 2499x^4, 4895 + 745x + 2205x^2 +$
$4054x^3 + 3533x^4, 4867 + 1191x + 1666x^2 + 3704x^3 + 8100x^4\},$
$\{0, 0, 6543 + 7867x + 6763x^2 + 1615x^3 + 4524x^4 + x^5, 1286 + 7316x + 1030x^2 +$
$955x^3 + 2536x^4 + 4963x^5, 3970 + 1175x + 3348x^2 + 7720x^3 + 7939x^4 + 5955x^5\}\}$

What we do above is by no means the only way to obtain the Hermite form of a matrix of polynomials. Several tactics for obtaining good computational efficiency are discussed in [25]. At the expense of more code one could adapt some of them to work within this Gröbner basis method. We also point out that method shown above for computing a matrix Hermite decomposition can readily be extended to compute a Smith decomposition, and in practice it seems to be fairly efficient.

We now 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. This method of diophantine solving may be found e.g. in [4]. A more efficient variant is presented in [20]. 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.

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 [6] and was also discussed in [13] (the relevant conferences were indeed but days apart).

For this example we use a 4×6 matrix of polynomials in x of degree at most 3. Again we will work modulo 8933. We first set up a random polynomial system.

SeedRandom[11111];
mod = 8933;
{mat, rhs} = randomSystem[3, 4, 6, x]

$$\{\{-9-2x, -1+3x, 8+8x+2x^2-2x^3, -4+4x+x^2-6x^3, -8+10x-7x^2-x^3, 10\},$$
$$\{-4-5x-9x^2, 2, 5-3x, 6+6x, -10, -9x^2\},$$
$$\{9-7x+4x^2, 10-8x, 9, -2-9x-6x^2, -3-9x, -5+6x\},$$
$$\{-1, -10-x, 9+9x+6x^2, 4, -9-9x, -1\}\}$$

$$\{-1-6x-2x^2-2x^3, -5+9x-10x^2+6x^3, -5+2x+10x^2-6x^3, 4+4x-3x^3\}$$

We compute a particular solution as well as a basis for the null vectors (solutions to the homogeneous system).

{sol, nulls} = systemSolve[mat, rhs, Polynomials[x], mod]

$$\{0, -6911-8325x-1007x^2-3258x^3-5553x^4-461x^5-2962x^6-2697x^7-5078x^8,$$
$$-1595-5252x-6226x^2-4661x^3-4977x^4-8450x^5-6487x^6-3824x^7,$$
$$-2138-3927x-285x^2-3502x^3-2954x^4-7104x^5-4269x^6-5140x^7,$$
$$-3462-306x-1529x^2-8826x^3-4560x^4-3639x^5-8726x^6-6177x^7,$$
$$-3040-6860x-7638x^2-7785x^3-1317x^4-3941x^5-4273x^6\},$$

$$\{\{1, 1278+8648x+1881x^2+7702x^3+5712x^4+4601x^5+855x^6+3166x^7+7530x^8, 5957+$$
$$5383x+7939x^2+2088x^3+1863x^4+6071x^5+1016x^6+1255x^7, 2282+4590x+$$
$$3441x^2+5607x^3+6567x^4+7241x^5+5280x^6+7826x^7, 1923+7650x+8533x^2+$$
$$179x^3+6723x^4+3135x^5+6022x^6+4132x^7, 5854+103x+5232x^2+5526x^3+387x^4+$$
$$8186x^5+517x^6\}, \{0, 116+4911x+8497x^2+4984x^3+158x^4+7000x^5+3210x^6+3726x^7+$$
$$4x^8+x^9, 4999+1437x+2565x^2+3994x^3+3624x^4+8754x^5+2977x^6+758x^7+$$
$$1489x^8, 2529+6147x+8417x^2+6672x^3+2816x^4+6216x^5+5228x^6+7431x^7+$$
$$5954x^8, 6720+3620x+5556x^2+1650x^3+1274x^4+342x^5+2973x^6+1489x^7+$$
$$5963x^8, 2400+2757x+7612x^2+6882x^3+8738x^4+4064x^5+7836x^6+8436x^7\}\}$$

It is straightforward to 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 from [7] for the analogous case of an integer system. We have six modular congruences in six variables that we wish to satisfy, with coefficient matrix, right hand side, and moduli as below. In this case our Euclidean domain is of course the integers. The code in the appendix is sufficiently generic as to handle this base ring as well.

```
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};
{soln, nulls} = systemSolve[mat, rhs, Integers, 0, moduli]
```

{0, −2, 4, 12802, −29779, −34696},
{{5, 0, 0, −18165, 4400, 333025}, {0, −5, 0, −16135, 26475, 445025},
{0, 0, 15, 17755, −26950, 540925}, {0, 0, 0, −39655, −4950, 594825},
{0, 0, 0, 0, −68750, 0}, {0, 0, 0, 0, 0, 1586200}}

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 [25]), 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 [7]. Moreover it provides the null vectors, and we can proceed to add multiples of them to the solution in order to obtain a solution that is smaller still. To this end, we will digress briefly from the topic of Gröbner bases over Euclidean rings, in order to describe how to find small integer solutions to a linear diophantine system. While this is an important computation in its own right (e.g. for integer linear programming, see [1], [14]), we also show later how the corresponding polynomial case might be handled using Gröbner bases.

To find a small solution to an integer linear equation we form 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 lattice reduction [10]. 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). Code in the appendix does this, returning the original solution if it fails in the attempt to find something smaller.

Using this we now obtain our new solution for the above example.

```
newsol = smallSolution[soln, nulls]
```
$\{565, 358, -326, 227, 21, -221\}$

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 a minor modification of *systemSolve*. It uses the *Mathematica* built in function *HermiteDecomposition*, instead of *groebnerHNF*, as the former is specialized for working over (rational or Gaussian) integers.

```
SeedRandom[1111];
mat = RandomInteger[{-100, 100}, {30, 35}] + iRandomInteger[{-100, 100}, {30, 35}];
rhs = RandomInteger[{-100, 100}, 30] + iRandomInteger[{-100, 100}, 30];
Timing[{soln, nulls} = systemSolve2[mat, rhs];]
```
$\{0.710000, \text{Null}\}$
```
Timing[smallsoln = smallSolution[soln, nulls];]
zeroTensor[mat.smallsoln - rhs]
```
$\{0.190000, \text{Null}\}$
True

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

```
{Max[Abs[N[soln]]], Max[Abs[N[smallsoln]]]}
```
$\{3.31318 \times 10^{73}, 6.08455 \times 10^{14}\}$

So the initial solution had elements with up to 74 digits whereas those in the small solution do not exceed 15 digits.

A related method for finding a small solution is presented in [17]. 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 (with anchor set to 1) has been referred to as the "embedding" technique in [22]. It is not clear where it originated (the code in the appendix dates to 1995) and it seems to have been independently discovered a few times. We also remark that the method used above can be strengthened in some cases to obtain still smaller solutions. Specifically, one can iterate, reducing the size of the weight as the solution vectors get progressively smaller, and this sometimes gives further size reductions.

5 Reduction of Polynomial Lattices

We now provide a univariate polynomial analog to the lattice reduction method for obtaining "small" generators. Reduction here is in the sense of [11], wherein one minimizes the largest degree of polynomials appearing in the matrix. This gives what is also called the weak Popov form of the matrix [21]. The idea is to compute a degree-based basis for the module, now using a "term over position" (TOP) order for the module Gröbner basis computation. This enforces that highest degree polynomials, regardless of column, will be higher in the term order than ones of lower degree, and hence will force degrees to be minimal. We again summarize this informal discussion as a theorem.

Theorem 3. *One can represent a diophantine system by a matrix of univariate polynomials over a field as a module over this domain, and via a module Gröbner basis computation one can then obtain the Popov form of the matrix.*

As one might expect from the description of this method, the actual implementation (see appendix) is virtually identical to that for computing the Hermite form, differing primarily in the module term ordering. There is also a slight difference in that we actually compute our Gröbner basis over the base field of the polynomial ring, now treating the polynomial variable as a bona fide variable rather than a coefficient. This is necessitated by our requirement of a TOP term order, wherein we must rank the module variables lower than the ring variable.

As an example we will again generate a random matrix, this time with all entries of fixed degree.

SeedRandom[1111]

mat = randomMatrix[4, 3, 5, *x*]

$\{\{-8+5x-3x^2-8x^3-4x^4, 2+4x+2x^2-2x^3+7x^4, -1-10x+9x^2-6x^3-3x^4, -2+5x-2x^2+4x^3+6x^4, 4+5x^2-8x^3-5x^4\}, \{4+5x-3x^2-6x^3-6x^4, -7-9x+8x^2+9x^3+2x^4, 10+x+5x^2+8x^3-8x^4, 4+5x-2x^2+x^3-9x^4, -10-4x-3x^2+3x^3+10x^4\}, \{-2x^2+4x^3+x^4, 7+10x-5x^2-3x^3-10x^4, -6-8x-6x^2+7x^3-3x^4, -9-4x-5x^2-6x^3-8x^4, -6+10x+4x^2+7x^3-9x^4\}\}$

We begin by computing the Hermite form, as this is in some sense as "far" as possible from "reduced" (as measured by orthogonality defect from [11]).

hnf = groebnerHNF[mat, Polynomials[*x*]]

$\{\{-3463744008314029656300, 0, 14191097841916761325395451418815988215855358x+6840308700051010164924x^2+7151499989765769609279x^3-3466256061833707815106x^4+\ldots\}, \{\ldots\}, \{0, 0, 244-299x-614x^2+2838x^3-217x^4-2050x^5+2192x^6+669x^7-598x^8-2215x^9+10x^{10}+37x^{11}-12x^{12}, -264+159x+856x^2-476x^3-914x^4+292x^5+651x^6-2849x^7-1789x^8+132x^9+1876x^{10}+1131x^{11}+373x^{12}, -736+40x+1464x^2-1948x^3-1790x^4-889x^5+2722x^6+2172x^7+621x^8+1078x^9-644x^{10}-1092x^{11}-926x^{12}\}\}$

redlat = polynomialLatticeReduce[hnf]

$\{\{596-125x+143x^2+130x^3, -585-869x+318x^2+515x^3, 592+885x-153x^2+374x^3+12x^4, 554+11x+220x^2+5x^3-373x^4, -358-472x-525x^2+325x^3+926x^4\}, \{268-55x+61x^2+62x^3, -255-379x+138x^2+229x^3-12x^4, 260+387x-75x^2+178x^3, 238+x+92x^2-5x^3-179x^4, -170-200x-231x^2+155x^3+406x^4\}, \{-892+175x-205x^2-182x^3+12x^4, 879+1303x-486x^2-781x^3, -896-1311x+219x^2-574x^3, -826-25x-320x^2-7x^3+575x^4, 554+704x+783x^2-491x^3-1390x^4\}\}$

It should 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. We will not pursue that here.

6 Bivariate Modular Polynomial Factorization

We now put together techniques from the preceding sections for the purpose of factoring a bivariate polynomial modulo a prime. For an example we will generate a pair of random polynomials such that there are terms of highest total degree in each variable separately; this brings 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 conveniences 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 bulletproof algorithm but rather to illustrate the method on a relatively simple, yet nontrivial, example.

We first generate a pair of pseudorandom bivariate polynomials and take their product.

mod = 19;
totdeg = 6;
SeedRandom[11112222];
poly1 = randomBivariatePoly $\left[\frac{\text{totdeg}}{2}, \text{mod}, x, y\right]$;
poly2 = randomBivariatePoly $\left[\frac{\text{totdeg}}{2}, \text{mod}, x, y\right]$;
poly = Expand[poly1poly2, Modulus → mod]

$8+7x+9x^2+15x^3+2x^4+2x^5+12x^6+18y+5xy+6x^2y+11x^3y+2x^4y+13y^2+5xy^2+15x^2y^2+15x^4y^2+10y^3+9xy^3+8x^2y^3+12x^3y^3+3y^4+9xy^4+5x^2y^4+13xy^5+7y^6$

We will evaluate at $x = 11$ and factor, removing the constant term. This produces all univariate factors of the evaluated polynomial.

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

$\{7+y, 8+y, 13+y, 15+y, 7+10y+y^2\}$

As in the Hensel lifting section we will use a simple Gröbner basis computation to lift a factor modulo a power of the ideal $(x - 11)$ that is sufficient to reclaim factors of degree 3 in x. A proof that this method gives the correct Hensel lifting could be constructed along the lines of that for the case when the base field is the complex numbers; see [15].

```
subst = (x - val);
pow = 24;
substpower = subst^pow;
liftedfactor = Last[GroebnerBasis[{poly, substpower, fax[[1]]^pow},y,
  Modulus -> mod, CoefficientDomain -> Polynomials[x]]]
```

$$11 + 8x + 6x^2 + 9x^3 + 6x^4 + 12x^5 + 2x^6 + 10x^7 + 17x^8 + 9x^9 + 13x^{10} + 16x^{11} + 4x^{12} + 15x^{13} + 17x^{14} + 5x^{15} + 17x^{16} + 16x^{17} + 7x^{18} + 17x^{19} + 9x^{20} + 8x^{21} + 5x^{22} + 15x^{23} + y$$

See [11] for bounds on how high one must lift in order to be guaranteed of finding a correct factor. We used degree of 24 which is sufficient for the example at hand. As in that reference we now set up a lattice of univariate polynomials in x, using shifts of both the lifted polynomial and the power of the prime ideal over which we initially factored.

```
deg = Exponent[liftedfactor, y];
lattice1 = Table[If[i == j, substpower, 0], {i, deg}, {j, totdeg}];
coeffs = PadRight[CoefficientList[liftedfactor, y], totdeg];
lattice2 = Table[RotateRight[coeffs, j], {j, 0, totdeg - 1 - deg}];
lattice = Join[lattice1, lattice2];
```

The minimal lattice row will provide a factor [11]. It might be a nontrivial multiple of the correct factor so we use a GCD extraction to get that factor.

```
candidate = First[redlat = polynomialLatticeReduce[lattice, mod]].
    y^Range[0,Length[lattice[[1]]]-1]
```

$$(8 + 9x + 14x^2 + x^3)\, y^2 + (16 + x + 2x^2)\, y^3 + (4 + 14x)y^4 + 4y^5$$

```
fac = PolynomialGCD[candidate, poly, Modulus -> mod]
```

$$8 + 9x + 14x^2 + x^3 + 16y + xy + 2x^2y + 4y^2 + 14xy^2 + 4y^3$$

We check that we have indeed recovered one of the true modular factors of our original polynomial.

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

True

We remark that one can do the lifting step using a different term order and in so doing recover the factor without resorting to a lattice reduction step. We show this below.

```
liftedfactor2 = First[GroebnerBasis[{poly, (x - val)^10, fax[[1]]^10}, {x, y},
  Modulus -> mod, MonomialOrder -> DegreeReverseLexicographic]]
```

$$8 + 9x + 14x^2 + x^3 + 16y + xy + 2x^2y + 4y^2 + 14xy^2 + 4y^3$$

The theoretical underpinnings to this approach are given, along with efficiency improvements, in [15]. One advantage, as might be noted above, is that often the lifting degree requirement is smaller.

7 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 $x = r + sd$ and $y = u + vd$, 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 $mx + ny = g$ 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 furthermore 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 $((2d-1)^2 - D)/4$; 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 all this may be found in the appendix.

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

$\mathbf{bezrels = bezout[}d\mathbf{, 51 + 43}d\mathbf{, 26 - 55}d\mathbf{, -19]}$
$\{\{1, \{115 - 2\left(1 + i\sqrt{19}\right), 101 - 17\left(1 + i\sqrt{19}\right)\}\}\}$

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

$\mathbf{Expand\left[bezrels[[1,2]].\{51 + 43\left(1+\sqrt{-19}\right)/2, 26 - 55\left(1+\sqrt{-19}\right)/2\} - bezrels[[1,1]]\right]}$
0

We now generate and work with a significantly larger example.

$\mathbf{n = 50;}$
$\mathbf{randsqrt = \sqrt{RandomInteger\left[10^n\right]}/.a\_Integer\sqrt{b\_} \to \sqrt{b}}$
$\mathbf{randqints = \{RandomInteger\left[10^n, 2\right].\{1, d\}, RandomInteger\left[10^n, 2\right].\{1, d\}\}}$

$\sqrt{766452102160682753412524494272509420425656415038 99}$
$\{41230119139644742056691832704420484800325317097349+$
$666376944348360059390936523158066996965211108376 33d,$
$29376606053454810686236077314995219308578051470644+$
$68102540162537581922579541354979200541266074057948d\}$

Timing[bezrels = bezout[d, Sequence@@randqints, randsqrt^2];]
$\{0.013998, Null\}$

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 [28], [18] for further details. We show below a simple approach that works in many situations. We work with $d = \sqrt{53719}$.

sqrt = Sqrt[53719];
qints = {73609 + 15577d, 2991 + 6417d};
bezrels = bezout[d, Sequence@@qints, sqrt^2]
$\{\{1 + \sqrt{53719}, \{14462895 - 3265382\sqrt{53719}, -1344274 + 7923502\sqrt{53719}\}\},$
$\{6, \{1128711 - 254907\sqrt{53719}, -104180 + 618536\sqrt{53719}\}\}\}$

We have reduced the ideal sum to one generated by $\left(1 + \sqrt{53719}, 6\right)$. There is in fact a GCD, though, because the ring $\mathbb{Z}\left[\sqrt{53719}\right]$ is a principal ideal domain (this follows from the fact that $\sqrt{53719}$ has class number of 1). Checking norms shows that any common factor of 6 and $1 + \sqrt{53719}$ will have a norm of 6 or -6. Thus we have a Pell type of equation to solve: find integers $\{a, b\}$ such that $(6a + b)^2 - 53719b^2 = \pm 6$. Well known facts about such equations tell us that any solution will have $(6a + b)/b$ as a convergent to the continued fraction expansion of $\sqrt{53719}$. We remark that for this method to work, we require that the right hand side, ± 6 in this case, have absolute value less than $\sqrt{53719}$.

cf = ContinuedFraction $\left[\sqrt{53719}\right]$;
frax = Convergents[cf];
solns1 = Table[Solve[
{(6 * a + b)^2 − 53719 * b^2 == 6, (6 * a + b)/b == frax[[j]]}, {a, b}], {j, Length[frax]}];
solns2 = Table[Solve[
{(6 * a + b)^2 − 53719 * 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}]

```
{{-342894841094108692200361834013658743982799930240348698449771068892658461568428877661316160283
2,-89150972140308509130356945221982374902233029263834884255241520937582267870954684338850528243},
{342894841094108692200361834013658743982799930240348698449771068892658461568428877661316160283
2,891509721403085091303569452219823749022330292638348842552415209375822678709546843388505282
43}}
```

We have found a GCD.

```
gcd = Expand[
  {342894841094108692200361834013658743982799930240348
    698449771068892658461568428877661316160283 2,
   891509721403085091303569452219823749022330292638348
    842552415209375822678709546843388505282 43}.
  {6, 1 + Sqrt[53719]}]
```

```
20662841437786830041152066986041507013870228843684756791241505654497089961976687344017820145235+
89150972140308509130356945221982374902233029263834884255241520937582267870954684338850528243 Sqrt[53719]
```

We confirm that the norm is indeed -6.

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

```
-6
```

8 Summary

We have presented several applications of Gröbner bases over Euclidean domains. The first ones showed how to work over quotient rings of Euclidean rings. Next we provided an algorithm, using Gröbner bases over Euclidean domains, to perform Hensel lifting of polynomial factors modulo powers of a prime ideal. We proceeded to compute matrix Hermite and weak Popov normal forms, and showed an application of these to multivariate modular factorization. We finished with an example showing nontrivial computations in quadratic number rings.

While many of these applications are specialized, in the sense that good methods are available that do not require Gröbner bases, it is all the same nice to have the methods given herein. One reason is that the code is simple and also flexible should modifications be desired. Another is that the methods presented here perform reasonably well on many problems that are of nontrivial size. Perhaps most interesting is that several fundamental computations in computer algebra, such as Hensel lifting, matrix canonical forms, and polynomial lattice reduction, as well as interplay

between these, may be cast as either Gröbner bases computations over Euclidean domains or close relatives thereof.

ACKNOWLEDGEMENTS.

I thank Michael Trott for drawing my attention to the article by Dolzmann and Sturm, as well as referees for beneficial comments on prior drafts of this work.

9 Code Appendix

Here we create the polynomial of degree 190 that we use in section 3.

```
poly1 = x^20 - 5x^18 + 864x^15 - 375x^14 - 2160x^13 + 1875x^12 + 10800x^11 + 186624x^10 -
  54000x^9 + 46875x^8 + 270000x^7 - 234375x^6 - 2700000x^5 - 1953125x^2 + 9765625;
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];
```

Here is the code for quadratic Hensel lifting of polynomial factors from $\mathbb{Z}_p[x]$. It is an important step in finding factors in $\mathbb{Z}[x]$.

```
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]
```

Here is code to create a random matrix of univariate polynomials over the integers.

```
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}]]
randomSystem[degmax_, rows_, cols_, var_]:=
  {randomMatrix[degmax, rows, cols, var],
    Table[randomPolynomial[degmax, var], {rows}]}
```

Next is our workhorse for computing Gröbner bases over modules.

```
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]]
```

We use a module Gröbner basis to compute the Hermite normal form of a matrix over a Euclidean domain.

```
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]]
```

Given a linear system of equations with univariate polynomials for coefficients, the code below will return a univariate polynomial solution vector if one exists. It also computes a basis of univariate polynomial null vectors for the system.

```
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}]
```

This next code finds small solutions to integer linear equations, given a particular solution and a basis for the integer null space.

```
smallSolution[sol_?VectorQ, nulls_?MatrixQ]:=Module[
  {max, dim = Length[nulls] + 1, weight, auglat, lat, k, soln},
  lat = Prepend[LatticeReduce[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]]
```

The following computes a matrix weak Popov normal form. This is also referred to as a lattice reduction for the case of matrices comprised of univariate polynomials over a field.

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

Here is code for generating a pseudorandom bivariate polynomial of a given total degree with coefficients in a given prime field.

```
randomBivariatePoly[deg_, mod_, x_, y_]:=
  ∑_{i=0}^{deg} ∑_{j=0}^{deg−i} RandomInteger[{If[i + j == deg &&ij == 0, 1, 0], mod − 1}] x^i y^j
```

With modest preprocessing we can use *smallSolution* to obtain "small" elements in quadratic rings.

```
quadraticIntegerToIntegerVector[n1_Integer, alg_]:={n1, 0}
quadraticIntegerToIntegerVector[n1_. + n2_. * alg_, alg_]:={n1, n2}

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 + n2theta, m1};
  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[((2theta - 1)^2 - tsqr)/4],
    subs = theta → Sqrt[tsqr]; theta^2 - tsqr];
  mat = Join[Transpose[Join[{polys}, IdentityMatrix[2]]], defpolyIdentityMatrix[3]];
  gb = groebnerHNF[mat, Integers];
  relations = Select[gb, #[[1]]=!=0&&FreeQ[#[[1]], theta^-1]&];
  solns = Map[Rest, relations];
  nulls = Map[Rest, Cases[gb, {0, _}]];
  nulls = DeleteCases[nulls, vec_/;!FreeQ[vec, theta^-1]];
  solns = Map[smallSolutionQuadratic[#, nulls, theta]&, solns];
  Partition[Riffle[Map[First, relations], solns]/.subs, 2]]
```

References

[1] K. Aardal, C. A. J. Hurkens, and A. K. Lenstra. Solving a system of linear diophantine equations with lower and upper bounds on the variables. *Mathematics of Operations Research*, 25(3):427–442, 2000.

[2] W. Adams and P. Loustaunau. *An Introduction to Gröbner Bases*. American Mathematical Society, Providence, RI, USA, 1994.

[3] T. Becker, H. Kredel, and V. Weispfenning. *Gröbner Bases: a Computational Approach to Commutative Algebra*. Springer-Verlag, London, UK, 1993.

[4] W. Blankenship. Algorithm 288: Solution of simultaneous linear diophantine equations. *Communications of the ACM*, 9(7):514, 1966.

[5] B. Buchberger. *Gröbner-Bases: An Algorithmic Method in Polynomial Ideal Theory.*, chapter 6, pages 184–232. Reidel Publishing Company, Dodrecht - Boston - Lancaster, 1985.

[6] M. Caboara and C. Traverso. Efficient algorithms for ideal operations (extended abstract). In *In International Symposium on Symbolic and Algebraic Computation (ISSAC 99*, pages 147–152, 1998.

[7] A. Dolzmann and T. Sturm. Parametric systems of linear congruences. In *Computer Algebra in Scientific Computing. Proceedings of the CASC 2000*, pages 149–166. Springer, 2001.

[8] A. Kandri-Rody and D. Kapur. Algorithms for computing Groebner bases of polynomial ideals over various euclidean rings. In *EUROSAM*, pages 195–206, 1984.

[9] A. Kandri-Rody and D. Kapur. Computing a Gröbner basis of a polynomial ideal over a euclidean domain. *Journal of Symbolic Computation*, 6(1):37–57, 1988.

[10] A. Lenstra, H. L. Jr., and L. Lovasz. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.

[11] A. K. Lenstra. Factoring multivariate polynomials over finite fields. *Journal of Computer and System Sciences*, 30(2):235–248, 1985.

[12] D. Lichtblau. Gröbner bases in Mathematica 3.0. *The Mathematica Journal*, 6(4):81–88, 1996. http://library.wolfram.com/infocenter/Articles/2179/.

[13] D. Lichtblau. Practical computations with Gröbner bases. http://www.math.unm.edu/ACA/1998/sessions.html, 1998.

[14] D. Lichtblau. Making change and finding repfigits: balancing a knapsack. In *ICMS*, pages 182–193, 2006.

[15] D. Lichtblau. Polynomial GCD and factorization via approximate Gröbner bases. In *Proceedings of the 2010 12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, SYNASC '10, pages 29–36, Washington, DC, USA, 2010. IEEE Computer Society.

[16] D. Lichtblau. Effective computation of strong Gröbner bases over euclidean domains. to appear, Illinois Journal of Mathematics, 2013.

[17] K. Matthews. Short solutions of A X=B using a LLL-based Hermite normal form algorithm. Manuscript, 2001.

[18] A. Miled and A. Ouertani. Extended gcd of quadratic integers. http://arxiv.org/abs/1002.4487, 2010.

[19] H. M. Möller. On the construction of Gröbner bases using syzygies. *J. Symb. Comput.*, 6(2-3):345–359, 1988.

[20] T. Mulders and A. Storjohann. Diophantine linear system solving. In *In International Symposium on Symbolic and Algebraic Computation (ISSAC 99*, pages 181–188. ACM Press, 1999.

[21] T. Mulders and A. Storjohann. On lattice reduction for polynomial matrices. *Journal of Symbolic Computation*, 35(4):377–401, 2003.

[22] P. Nguyen. Cryptanalysis of the Goldreich-Goldwasser-Halevi cryptosystem from Crypto '97. In *Proceedings of Advances in Cryptology - CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 288–304. Springer-Verlag, 1999.

[23] L. Pan. On the D-bases of polynomial ideals over principal ideal domains. *J. Symb. Comput.*, 7(1):55–69, 1989.

[24] C. Sims. *Computation with Finitely Presented Groups*. Cambridge University Press, 1994.

[25] A. Storjohann. *Computation of Hermite and Smith Normal Forms of Matrices*. University of Waterloo, 1994. Master's Thesis, University of Waterloo Deptartment of Computer Science.

[26] M. van Hoeij. Factoring polynomials and the knapsack problem. *Journal of Number Theory*, 95(2):167–189, 2002.

[27] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, New York, NY, USA, 2003.

[28] A. Weilert. Two efficient algorithms for the computation of ideal sums in quadratic orders. *Mathematics of Computation*, 75(254):941–981, 2006.

[29] I. Wolfram Research. Mathematica 9, 2012.

[30] P. Zimmermann. Polynomial factorization challenges: a collection of polynomials difficult to factor, 2003. http://www.loria.fr/ zimmerma/mupad/.

Received: March 9, 2013