Exact Computation Using Approximate Gröbner Bases

Daniel Lichtblau

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

danl@wolfram.com

ABSTRACT. We discuss computation of approximate Gröbner bases at high but finite precision. We show how this can be used to deduce exact results for various applications. Examples include implicitizing surfaces, finding multivariate polynomial greatest common divisors and factorizations over the rational and complex number fields.

This is an extended version of a paper for SYNASC 2010, titled "Polynomial GCD and Factorization Via Approximate Gröbner Bases", to appear in IEEE conference proceedings.

Categories and Subject Descriptors

G.1.0 [**Numerical Analysis**]: General— Multiple precision arithmetic; Numerical algorithms; I.1.2 [**Symbolic and Algebraic Manipulation**]: Algorithms--- Algebraic Algorithms;

General Terms

Algorithms

Keywords

Gröbner basis, polynomial gcd, hybrid symbolic-numeric computation

1. INTRODUCTION

Since their introduction more than 40 years ago, usage of Gröbner bases has become pervasive in the realm of computational algebra. As is well known, most common applications are in algebraic manipulation of polynomial ideals. Less known is that they also can be used in the setting of classical polynomial algebra, specifically in finding multivariate polynomial greatest common divisors and factorizations. Some approaches are presented in [1, 2, 3, 9] but they are not regarded as being competitive with more commonly used methods. The primary aim of this paper is to offer alternative methods to the classical polynomial algebra algorithms, building them on simple Gröbner basis computations. The methods we will describe and demonstrate seem to be more efficient, in practice, than those that have appeared in past.

A relatively more recent thread in the literature on Gröbner bases involves computation using approximate coefficients. This gives rise to an important issue, recognition of cancellation, without which the algorithm might

either give poor results or even fail to terminate. Shirayanagi [24] gave the first indication of how to handle this. Since that time several other approaches have appeared [15, 18, 22, 27]. In [18] there is discussed an implementation, parts of which have been in *Mathematica* since 1996 [17]. In contrast to [18], the present article uses approximation at high precision, as a means for later recovering exact results.

One point of the present work is to illustrate known methods for recovering rational or algebraic results from good approximations for Gröbner bases. For this we rely on results in section 2 that allow us to conclude such approximations can be attained. We give simple examples in section 3. Our main application will be in polynomial algebra, specifically, finding greatest common divisors and factorization. Sections 4 and 5 provide the relevant theory. They cover how to recover common divisors or factors from certain Gröbner bases that use substitution homomorphisms. For factorization, we remark that there is no step involving combination of factor images in this approach.

Typical experience is that classical methods for computing common divisors or factorizations are much faster than those methods that rely on Gröbner bases. This, however, is predicated on usage of exact Gröbner basis computations. We will propose methods involving approximate basis computation that seem to be more competitive, due to avoidance of both coefficient swell and explicit algebraic numbers. We illustrate in this setting with several examples in section 6. Specifically, we use approximate bases to obtain exact results. We then discuss failure modes. Last we summarize and indicate a number of useful directions for further work.

Acknowledgements

I thank Ilias Kotsireas for inviting me to present an early version of this work at ACA 2008 (Linz) in the session Gröbner Bases and their Applications. I also thank him and the other two session organizers, Elizabeth Arnold and Markus Rosenkranz, and honorary Session Chair Bruno Buchberger, for all the hard work that went into their session. I thank the anonymous referees of the conference version of this work for their many useful comments and questions. I also thank a referee of an earlier version of this paper for pointing out some typographical errors.

2. COMPUTATION OF APPROXIMATE GRÖBNER BASES

We briefly summarize the model of computation we use in this paper. In contrast to empirical polynomials (as encountered e.g. in [8, 18]), we assume inputs are known exactly. We also assume that we can do coefficient arithmetic at arbitrary finite precision (in practice, this might involve several hundred digits).

The model used for coefficients, and operations thereon, is significance arithmetic, as described in [14, 25]. In brief, numbers have a value and also an approximation of their maximal error. Standard arithmetic such as addition and multiplication of such numbers propagates the error via first order approximations. At low precision this tends not to be reliable. The (unknown) contributions to error from higher order terms (e.g. the product of the errors, if doing multiplication) can simply overwhelm the error estimate. At high precision, however, this model tends to be quite robust and in practice it is reliable. See [25] for further details. In this paper we will typically work at precisions of dozens to hundreds of digits. This suffices to make the computations reliable, while not being so large as to detract from the efficiency offered by using finite precision arithmetic.

Recall that in computing Gröbner bases, it is of paramount importance in polynomial reduction to recognize when sums of like terms cancel, as this is how we obtain new leading coefficients. In our model of arithmetic we regard a sum as zero when there is full cancellation of all digits, that is, the result is less than the approximated error interval.

When using significance arithmetic, over the course of many operations there will be a (typically gradual) erosion of precision. In practical usage one simply starts with a finite approximation of exact input that is sufficiently high as to allow the algorithm to run to completion without encountering loss of too much preci-

sion. What this means will be explained in more detail below. First we state a result that has appeared before in various forms. We tailor it to the needs of our arithmetic model.

Theorem 1. We are given a set F of polynomials in some set of indeterminates over the rationals, a term order, and an outcome precision s. Also we assume we use a specific algorithm to compute its reduced, minimal Gröbner basis (Buchberger's algorithm, say), and we call that basis G. Then there exists a finite precision r such that if we use that algorithm to compute a Gröbner basis for F, begin with r or more digits, and use finite precision significance arithmetic on coefficient sums, products, and reciprocals, then the result will agree with G to s digits in all coefficients.

Proof. In running the algorithm to obtain G from F, there will occur a maximal integer coefficient size M. Moreover there occurs a certain number A of arithmetic operations on coefficients, and a maximal cancellation K of digits in operations that do not yield zero (this cancellation is what causes loss of precision when repeating the process in finite precision; clearly $K \le M$). Given these parameters, and a desired precision s for the outcome, we assert the following. One can find a start precision, and repeat the algorithm using significance arithmetic, with all steps having the following properties.

(i) The precision of intermediate values is always larger than s.

(ii) Cancellations of more than K digits will correctly be regarded as resulting in zero. Smaller cancellations will retain significant digits and hence not be regarded as zero.

To see that this assertion is correct, we first observe that multiplications lose at most $M + 1$ digits of precision, and additions at most $K + 1$ (1 from roundoff and K, by assumption, from cancellation). See [25] for details on how this loss of precision is modelled to first order in significance arithmetic. The salient point is that we have a means to recognize when a full cancellation has taken place, that is, when two terms sum to zero. It happens precisely when there is cancellation of all significant digits. We need only guarantee that the initial precision be sufficiently high in order to deduce such cancellation (that is, recognize zero) correctly. This in turn implies that the finite precision algorithm, if given sufficient precision of input, will correctly emulate the exact case. From the preceding remarks we see that a starting precision of $A(M + 1) + s$ digits suffices. □

We remark that the sufficient initial precision of the above theorem is generally far larger than necessary (and also cannot be computed a priori, since it depends on information specific to running the code in exact arithmetic). While the proposition sheds no light on what will be a realistic precision in which to begin a computation, we will show examples that might be regarded as typical of actual practice.

3. EXACT RESULTS FROM APPROXIMATE GRÖBNER BASES

Given a set of polynomials with rational coefficients, we might find an approximate Gröbner basis to high precision and then attempt to recover the exact basis from the approximation. The point of such an endeavor is that it can be faster to do this, and validate the result, than to compute the exact basis by exact methods.

At this point it is helpful to see an actual example where use of approximate numbers gives a substantial improvement over exact computation. This example is mentioned in [26]. It is an implicitization of a polynomial parametric surface. To find the implicit form we compute a Gröbner basis using a term ordering that eliminates the parameters [9]. The timing shown is in seconds; an exact computation of the implicit form took several hours. Here we do the computation at 500 digits; this will suffice to recover the exact form.

Example 1

```
surfacePolys =
  {-17 - 27 s + 36 s^2 - 19 s^3 - 45 s t^2 + 81 s^2 t^2 - 54 s^3 t^2 + 30 s t^3 - 54 s^2 t^3 + 36 s^3 t^3 + 10 x,
   198 s - 369 s^2 t + 246 s^3 t - 198 t^2 + 369 s^2 t^2 - 246 s^3 t^2 + 100 y, -57 - 81 s^2 + 42 s^3 +
    99 t^2 - 81 s t^2 - 108 s^2 t^2 + 90 s^3 t^2 - 66 t^3 + 54 s t^3 + 72 s^2 t^3 - 60 s^3 t^3 + 40 z};
elims = {s, t};
vars = {x, y, z};
Timing[gbapprox = GroebnerBasis[surfacePolys, vars, elims,
    MonomialOrder → EliminationOrder, CoefficientDomain → InexactNumbers[500],
    Method → {"GroebnerWalk", "EarlyEliminate" → True}];]
{168.239, Null}
```

We recover the exact form simply by rationalizing the approximate values. This well known technique uses continued fraction approximation. Since it might fail to find rationals sufficiently "close" to the approximate numbers, we will verify explicitly that we indeed obtained rational values.

```
Timing[implicit = Rationalize[gbapprox[[1]]]; Precision[implicit]]
{0.346947, ∞}
```

One also should verify that the polynomial obtained is the correct one. This is straightforward (plug the parametric values for the variables into the implicit polynomial, and check that it expands to zero); we omit the details.

Using similar methods we can implicitize the four difficult patches of the iconic Newell teapot (these four cover its spout). To the best of my knowledge this is the only way in which their implicit forms have been found using Gröbner bases. (The remaining 28 patches can all readily be handled via Gröbner bases using exact computation; the timings vary from a fraction of a second to a few seconds).

One can do more than just recover rational numbers. Suppose, for example, we have a system of polynomial equations over the rationals, and seek exact solutions. We know from general theory that these solutions will be algebraic numbers. One approach to obtaining them is to find numeric approximations to high precision, and then deduce the algebraic numbers using, say, an algorithm based on PSLQ or lattice reduction [10, 16]. Unfortunately, preliminary results in this direction are not promising, except in cases of low degree algebraic numbers. This is because typical solutions require large integers in the algebraics, and so a tactic of recovering exact from approximate tends to be slower than a direct computation using exact methods (that possibly use approximate arithmetic in the process of computing a Gröbner basis, as was done above). But such methods of algebraic recovery will be useful to us in the context of factorization, as will be seen in a later example.

4. BIVARIATE POLYNOMIAL GCD COMPUTATION

For simplicity we restrict our attention to bivariate polynomials. Generalizing to the multivariate case is straightforward (though making it computationally efficient might pose a serious challenge). We will use a result related to material in [12]. We develop the relevant theory in a series of propositions.

Proposition 1. Suppose we have $f_1 = g\,h$ *and* $f_2 = g\,k$ *with* g, h, k *bivariate polynomials in* $\mathbb{Q}[x, y]$, *and* h, k *relatively prime. Suppose* $y_0 \in \mathbb{Q}$ *is a generic value. For our purposes of genericity we require the following conditions.*

(1) g *and* $(y - y_0)$ *are relatively prime (that is,* $(y - y_0) \nmid g$)

(2) $\deg_x(g(x, y)) = \deg_x(g(x, y_0))$, *where* $\deg_x(p)$ *denotes the degree in* x *of* p.

(3) The variety $V(h, k, y - y_0)$ *is empty (this is a generic condition from the hypothesis that* $\gcd(h, k) = 1$).

Then for all positive integers n, we have equality of ideals $(f_1, f_2, (y - y_0)^n) = (g, (y - y_0)^n)$.

Proof. Set $I_n = (f_1, f_2, (y - y_0)^n)$ and $J_n = (g, (y - y_0)^n)$. Since g divides both f_1 and f_2, clearly for all n we have $I_n \subset J_n$. For the reverse inclusion we proceed by induction. In the base case, $n = 1$, we are in the setting of univariate polynomials (since y is now equivalent to the constant y_0). Since $h_0 = h(x, y_0)$ and $k_0 = k(x, y_0)$ are univariate and relatively prime, the extended gcd provides cofactors $a(x)$ and $b(x)$ with $a\, h_0 + b\, k_0 = 1$. Thus $a(x)\, f_1(x, y_0) + b(x)\, f_2(x, y_0) = g(x, y_0)$, proving that $J_1 \subset I_1$.

Now we assume the result for n and look at the case $n + 1$. Since by assumption $I_n = J_n$, there exist (bivariate) polynomials s, t with $s\, f_1 + t\, f_2 \equiv g$ modulo $(y - y_0)^n$. Multiplying through by $(y - y_0)$ we see that $g(y - y_0) \in I_{n+1}$. The extended gcd above implies the existence of $w(x, y)$ such that $a\, h + b\, k + w\, (y - y_0) = 1$ (since by construction $a\, h + b\, k = 1 + p$ where $(y - y_0) \mid p$). Thus $g = g\,(a\, h + b\, k + w\, (y - y_0)) = a\, f_1 + b\, f_2 + (w\, g)\, (y - y_0) \in I_{n+1}$, showing the inclusion $J_{n+1} \subset I_{n+1}$. □

Proposition 2. Suppose g, h, k, y_0 satisfy the same hypotheses as in proposition 1. Suppose moreover that $\deg_x(g)$ is m and $\deg(g)$ is n, where $\deg(p)$ is the total degree of p. Also we now assume the content of g with respect to x is 1 (that is, g has no factor in y alone). For any positive integer $r > \lceil n^2/m \rceil$, let G_r be a Gröbner basis for the ideal $I_r = (f_1, f_2, (y - y_0)^r)$ with graded lexicographic term order using variable order $x \succ y$. Let $\tilde{g}$ be the minimal polynomial in G_r with respect to this term order. Then $\tilde{g}$ is an associate of g, that is, it is a gcd of (f_1, f_2).

Note that this simply means g and $\tilde{g}$ agree up to invertible factor which, in our setting, would be a nonzero rational number. The greatest common divisor is of course only unique up to such factors.

Proof. We adapt an argument from [20], based on sizes of certain zero sets. Suppose there is a polynomial $g_2 \in G_r$ with leading power product strictly smaller than that of g. We take g_2 to be of minimal degree. Let I_{g_2} denote the ideal (g, g_2). Let #I denote the cardinality of the zero set, counted by multiplicity, of a finite ideal I.

We first claim we do not have $g_2 \mid g$. If that happened we would have $g = g_2 g_3$ for some g_3 a nontrivial polynomial in x (recall g is assumed primitive in x). Hence $\#(g_3, (y - y_0)^r) > 0$, and it follows that $\#(g_2, (y - y_0)^r) < \#I_r$. But $g_2 \in G_r$ implies $g_2 \in I_r$ and divisibility of g by g_2 would further imply $(g_2, (y - y_0)^r) = (g, (y - y_0)^r) = I_r$, contradicting the size inequality.

As $g_2 \nmid g$, the gcd of g_2 and g (call it g_4) must be of lower degree than g_2. Replacing $\{f_1, f_2\}$ by $\{g, g_2\}$ in proposition 1, we see that $g_4 \in I_r$. This contradicts the minimality of leading term of g_2. □

Observe that a generic linear change of coordinates (that is, a shear transformation of the form $y \to y + a\, x$) will make $\deg_x(g) = n$. This means there will be a pure term in x of maximal (total) degree. In this situation we can use a Gröbner basis for the ideal $(f_1, f_2, (y - y_0)^{n+1})$, and we can learn the value for n from the univariate polynomial $\gcd(f_1(x, y_0), f_2(x, y_0))$. There is now, moreover, a simpler proof of proposition 2. If the leading power product of g, in the specified term order, is x^n, then the gcd of the leading power products of g and $(y - y_0)^{n+1}$ is 1. Hence by the Buchberger criterion on independent leading terms, $\{g, (y - y_0)^{n+1}\}$ is a Gröbner basis. As all terms in g are of degree no larger than n, neither polynomial can be used to reduce the other. Thus this basis is reduced.

These propositions taken together give the results we will use in the sequel. We first state the general case.

Theorem 2. Suppose we have $f_1 = g\, h$ and $f_2 = g\, k$ with g, h, k bivariate polynomials in $\mathbb{Q}[x, y]$, and h, k relatively prime. Suppose $y_0 \in \mathbb{Q}$ is a value satisfying the three generic conditions from the hypotheses of

proposition 1. Suppose $\min(\deg(f_1), \deg(f_2))$ *is* n. *Then the minimal polynomial in the Gröbner basis for* $\left(f_1, f_2, (y - y_0)^{n^2+1}\right)$, *computed with respect to the graded lexicographic term order using variable order* $x > y$, *is a gcd of the pair* $\{f_1(x, y), f_2(x, y)\}$. *Letting* m *be the degree in* x *of the univariate polynomial* $\gcd(f_1(x, y_0), f_2(x, y_0))$, *we can instead use the (in general smaller) exponent* $\lceil n^2/m \rceil + 1$.

An important special case, immediate from the observation following the proof of proposition 2, is that after generic change of coordinates we may use the (in general smaller) exponent $n + 1$. Moreover we can deduce that value from the degree of the univariate greatest common divisor obtained by a generic substitution value y_0.

Theorem 3. Again using the hypotheses of Theorem 2, suppose we perform a generic linear change of coordinates. Suppose moreover that the degree of (the univariate) $\gcd(f_1(x, y_0), f_2(x, y_0))$ *is* n. *Then the smallest polynomial in the Gröbner basis for* $(f_1, f_2, (y - y_0)^{n+1})$, *computed with respect to the term order of theorem 2, is a gcd of the pair* $\{f_1, f_2\}$.

It should be mentioned that this generic change is not disasterous for the computations. While general experience with Gröbner bases suggests that sets of sparse polynomial are more efficiently handled than their dense counterparts, we have two things in our favor. One is that we are in a bivariate setting, hence the number of monomials for a given total degree grows only quadratically. The other is that we will not suffer from explosive coefficient growth (either from the basis change itself, or in the process of computing the required Gröbner basis), since we work in finite precision. Nevertheless, it remains to be determined whether the benefits of reducing the necessary lifting degree outweighs the possible loss of efficiency from making the input dense. Empirical evidence suggests that it does, but this is by no means definitive.

5. BIVARIATE FACTORIZATION

We use the following setup throughout this section. We are given a square free bivariate polynomial $f(x, y)$ over the rationals, and a value $y_0 \in \mathbb{Q}$ such that $(y - y_0) \nmid f$. We assume f has no factors in y alone. Let x_0 be a root of $f(x, y_0)$, that is, $f(x_0, y_0) = 0$. Again we assume y_0 is generic, specifically $f(x, y_0)$ has the same degree in x as $f(x, y)$, and it remains square free. We will note other generic requirements as we proceed. Let $g(x, y)$ be the irreducible factor of $f(x, y)$ in $\mathbb{K}[x, y]$ for which $g(x_0, y_0) = 0$. By $\mathbb{K}$ we allow for working over the rationals $\mathbb{Q}$ or their algebraic closure $\overline{\mathbb{Q}}$ or (a numeric approximation to) the complex numbers $\mathbb{C}$. Let g_0 denote $g(x, y_0)$; here we impose the generic condition on y_0 that g and g_0 have the same degree in x. For each positive integer n we define $I_n = (f, g_0^n, (y - y_0)^n)$.

Theorem 4. For all n *we have* $g \in I_n$.

Outline of proof: First observe that g and g_0 agree up to a term with factor $(y - y_0)$. That is, $g = g_0 + s\,(y - y_0)$ for some polynomial $s(x, y)$. So the result holds for $n = 1$. It suffices to show that it holds for all powers of 2, since these ideals are descending (that is, $I_n \supset I_{n+k}$ for $k > 0$). Moreover we may assume $g_0 \neq g$ since otherwise the result follows from proposition 1.

We proceed by introducing a family of "conjugate" factors to powers of g. We define $\overline{g} = (g_0 - s\,(y - y_0))$ and next observe that $g\,\overline{g} = g_0^2 - s^2(y - y_0)^2 \equiv g_0^2$ modulo $(y - y_0)^2$. Thus $(f, g_0^2, (y - y_0)^2) = \left(f, g\,\overline{g}, (y - y_0)^2\right)$. Irreducibility of g implies it is relatively prime to $\overline{g}$. We next claim that $\overline{g}$ is relatively prime to f (we show this below). So the greatest common divisor of f and $g\,\overline{g}$ is g. By Proposition 1 we conclude $g \in I_2$. (It is here that we impose a further assumption on genericity of y_0, per requirement (3) of that proposition.)

To go from $n = 2$ to $n = 4$ and thence to higher powers of 2, observe that we can repeat this multiplication tactic. The last paragraph tells us we can write $g = t\,g_0^2 + u\,(y - y_0)^2 + v\,f$ for some polynomials t, u, and v. This

time we take as cofactor $\overline{g} = t\, g_0^2 - (u(y - y_0)^2 + v\, f)$ and observe that $g\,\overline{g} \equiv t^2 g_0^4$ modulo $\{f, (y - y_0)^4\}$. Thus $g\,\overline{g} \in I_4$. We deduce as before that $g \in I_4$. We omit the formal induction details.

To complete the proof we need to show that $\overline{g}$ and f are relatively prime. Observe that we may treat y_0 as a variable, and when viewed that way g_0 is an analytic function in y_0. (It is of course a polynomial in x, but if we treat y_0 as a variable then it is well known that the coefficients of this polynomial are analytic functions of y_0.) Since g_0 is not constant in y_0, whereas g is constant (with respect to y_0) , we see that $\overline{g}$ is likewise not constant in y_0. Now suppose that for each y_0 there is a polynomial $h(x, y)$ (a priori with coefficients dependent on y_0) such that $h \mid f$ and $h \mid \overline{g}$. As f has but finitely many factors, it is easy to show that for a dense set of values y_0, h is the same factor. In other words, we may assume h is also constant with respect to y_0. Recalling the definition of $\overline{g}$, we have $h \mid (g - 2s(y - y_0))$ (where s is function of (x, y, y_0)). As this holds for any y_0, it holds for $y_0 = y$, and so $h \mid g$. Thus h must be constant since $\overline{g}$ is relatively prime to g. This finishes the proof that $\overline{g}$ is relatively prime to f. □

Corollary 1. If $\deg_x(g) = \deg(g) = n$ *then g is the polynomial of minimal degree in* I_{n+1}.

Note that a generic linear change of coordinates guarantees the hypotheses of this corollary will be satisfied. We also remark that these ideals in effect provide lifts of factors that are correct modulo powers of $(y - y_0)$. This corollary thus gives an effective lifting bound.

Proof: This follows from the counting argument of the preceding section for greatest common divisiors in certain ideals, coupled with the proof of Theorem 4 which shows that we find g in exactly that setting. □

Corollary 2: In the setting of absolute factorization over $\mathbb{C}$, *the minimal degree factor of f that vanishes at the point* (x_0, y_0) *is in the ideal* $(f, (x - x_0)^n, (y - y_0)^n)$ *for all n.*

Proof: Simply note that we can replace g_0 by the irreducible factor of $g(x, y_0)$ vanishing at (x_0, y_0) and still have the desired ideal inclusion of theorem 4. Clearly $(x - x_0)$ is that factor of g_0. Now apply Theorem 4. □

Indeed, a minor variation of this corollary holds for base field of $\mathbb{Q}$ as well, but it is of less use in that case due to corollary 1, along with the fact that working with $(x - x_0)$ entails a need to lift further than if we work with g_0 (which, when the base field is $\mathbb{Q}$, is a product of $(x - x_0)$ and its algebraic conjugates). That is to say, $(x - x_0)$ is in general a proper divisor of g_0. We remark that the irreducible factor modulo $(y - y_0)$ of g, that we obtain computationally, might in principle be a proper factor of g_0. This does not affect the lifting bounds in our analysis above. We also note that for generic y_0 our factor will indeed be g_0 when we work over the rationals. This follows from Hilbert's irreducibility theorem [29].

From a practical standpoint we get too many low degree polynomials in our ideals unless we go to sufficiently high powers of $(x - x_0)$ and $(y - y_0)$. One must then ask what degree suffices for lifting. This is provided in the next theorem.

Theorem 5. Suppose $\deg(f) = n$, *and* $f(x_0, y_0) = 0$. *Let* $m = n^2 + 1$. *In the setting of factoring over* $\mathbb{C}$, *the minimal degree factor of f that vanishes at the point* (x_0, y_0) *is the minimal polynomial in a Gröbner basis for* $(f, (x - x_0)^m, (y - y_0)^m)$, *where the term order is graded reverse lexicographic with* $x \succ y$.

This can be proved with a zero set size argument along the lines of the proof of Proposition 2. We omit the details.

We point out that there is a large body of literature in the realm of absolute factorization. For some of the existing methods see [4, 5, 6, 7, 8, 11, 13, 21, 23] and references therein. In particular [5] provides substantial

detail about a few of these methods. We do not claim that our approach is as efficient as some of these others. Its main advantage is that it requires but little code. Moreover as it rests on fundamental technology in symbolic computation (i.e. Gröbner bases), general improvements in that area can have a beneficial impact on our method.

6. BIVARIATE GCD AND FACTORIZATION EXAMPLES

We now show several examples that serve to illustrate the methods. We use precisions based on trial and error; this aspect to the methods is not automated. For simplicity of exposition we also avail ourselves of a minor short cut. To wit, we perform no change of variables. It turns out that all the examples either already are of full degree in the unsubstituted variable, or still work at degree bounds assumed by that case. One should realize that this also confers a slight speed advantage because coefficients remain of modest size (thus slightly reducing the precision needed in the approximate basis computations), and moreover any sparseness we might have in the computations will not be destroyed.

All timings are in seconds. These were run on a 3.2 GHz processor, using version 7 of *Mathematica* [28] running under Linux.

GCD

We first show an example from [8] that factors over the rationals. We remark that their variant had some numerical noise (they were illustrating an approximate factorization method), whereas we work with the exact input.

Example 2

```
g = -84 + 41 x + 23 y + 99 x^2 y^5 - 61 x^2 y^4 - 50 x^2 y^3 - 12 x^2 y^2 - 18 x^2 y -
    26 x y^7 - 62 x y^6 + x y^5 - 47 x y^4 - 91 x y^3 - 47 x y^2 + 66 x^3 y - 55 x^7 y - 35 x^6 y^2 +
    97 x^6 y + 79 x^5 y^3 + 56 x^5 y^2 + 49 x^5 y + 57 x^4 y^4 - 59 x^4 y^3 + 45 x^4 y^2 - 8 x^4 y +
    92 x^3 y^5 + 77 x^3 y^2 + 54 x^3 + 53 y^6 + 31 x^2 - 90 y^7 - 58 y^8 - 85 x^8 - 37 x^7 - 86 y^2 +
    50 x^6 + 83 y^3 + 63 x^5 + 94 y^4 - 93 x^4 - y^5 - 5 x^2 y^6 - 61 x y + 43 x^3 y^4 - 62 x^3 y^3;
h = -76 - 53 x + 88 y + 66 x^2 y^5 - 29 x^2 y^4 - 91 x^2 y^3 - 53 x^2 y^2 - 19 x^2 y + 68 x y^6 -
    72 x y^5 - 87 x y^4 + 79 x y^3 + 43 x y^2 + 80 x^3 y - 50 x^6 y - 53 x^5 y^2 + 85 x^5 y +
    78 x^4 y^3 + 17 x^4 y^2 + 72 x^4 y + 30 x^3 y^2 + 72 x^3 - 23 y^6 - 47 x^2 - 61 y^7 + 19 x^7 -
    42 y^2 + 88 x^6 - 34 y^3 + 49 x^5 + 31 y^4 - 99 x^4 - 37 y^5 - 66 x y - 85 x^3 y^4 - 86 x^3 y^3;
f = Expand[g h];
```

We begin by forming another polynomial, e, as the product of g and a random bivariate k of similar degree.

```
randomPoly[deg_, vars_] := Module[{n = Length[vars], t, poly, terms},
  poly = RandomInteger[{-10, 10}, {deg + 1}].t^Range[0, deg];
  Expand[poly /. t^j_ :> (RandomInteger[{-5, 5}, {Length[vars]}].vars)^j]]
SeedRandom[1111];
k = randomPoly[7, {x, y}];
e = Expand[g k];
```

We now compute an approximate Gröbner basis for the ideal consisting of $\{f, e, (y - y_0)^d\}$ for a randomly selected y_0 and $d = \min(\deg(f), \deg(e)) + 1$. Shown below is the timing, in seconds.

```
First[Timing[gby = GroebnerBasis[{f, e, (y - RandomInteger[{20, 40}])^9},
      {x, y}, MonomialOrder → DegreeReverseLexicographic,
      CoefficientDomain → InexactNumbers[200]];]]
0.168975
```

We check that the recovered common factor agrees with g up to multiplicative factor.

```
Together[Numerator[Together[Rationalize[First[gby]]]] / g]
-1
```

Factorization

The code below is based on the theory presented for bivariate factorization over the rationals. For efficiency we use quadratic lifting rather than trying to do the full lift in one step. We do this by squaring polynomials in the preceding basis and augmenting with the polynomial whose factorization we seek. In practice this makes a tremendous difference in speed. Moreover it allows us to work with lower initial precision than would otherwise be the case, because overall precision loss is less when done in quadratic stages rather than all at once.

At each step we print the time in seconds it took for the Gröbner basis computation of that lift, and the precision of the result. We give up if the initial precision is insufficient to obtain the approximate numeric bases for the full number of lifting steps required for this task. In this case we use the best result we have prior to this failure, in the hope that it might have been sufficiently lifted to form a correct factor. An improvement we do not implement would be to perform trial division tests along the way.

```
factorBivariate[f_, x_, y_, y0_, prec_] := Catch[Module[
   {rts, gb, n, fax, fac},
   fax = Select[FactorList[f /. y → y0], ! NumberQ[#[[1]]] &];
   fac = fax[[1, 1]];
   gb[0] = N[{fac, y - y0}, prec];
   n = Ceiling[Log[2, Exponent[fac, x] + 1]];
   Print["lift ", n, " times"];
   Do[Print[{First[Timing[gb[j] =
          GroebnerBasis[Flatten[{f, gb[j - 1]^2}],
           {x, y}, MonomialOrder → DegreeReverseLexicographic,
           CoefficientDomain → InexactNumbers];]], Precision[gb[j]]}];
    If[Head[gb[j]] === GroebnerBasis, Throw[Numerator[
       Together[Rationalize[First[gb[j - 1]]]]]]];
    , {j, n}];
   fac = First[gb[n]];
   Clear[gb];
   Numerator[Together[Rationalize[fac]]]]]
```

Example 3

We will continue with the previous example, this time factoring f. We first find roots in x upon setting y to a fixed value (we use 5 in this case). We will work with the first such root to develop a full bivariate factor. We begin with 500 digits of precision.

```
Timing[factor = factorBivariate[f, x, y, 5, 500]]
lift 3 times
{0.016997, 469.763}
{0.052992, 421.616}
{0.212968, 331.525}
{0.288956, -76 - 53 x - 47 x^2 + 72 x^3 - 99 x^4 + 49 x^5 + 88 x^6 + 19 x^7 + 88 y -
  66 x y - 19 x^2 y + 80 x^3 y + 72 x^4 y + 85 x^5 y - 50 x^6 y - 42 y^2 + 43 x y^2 - 53 x^2 y^2 +
  30 x^3 y^2 + 17 x^4 y^2 - 53 x^5 y^2 - 34 y^3 + 79 x y^3 - 91 x^2 y^3 - 86 x^3 y^3 + 78 x^4 y^3 + 31 y^4 -
  87 x y^4 - 29 x^2 y^4 - 85 x^3 y^4 - 37 y^5 - 72 x y^5 + 66 x^2 y^5 - 23 y^6 + 68 x y^6 - 61 y^7}
```

Observe that we indeed recovered a correct factor, specifically $h(x, y)$.

```
Together[factor / h]
1
```

In the code above we use yet another short cut, which is to assume the univariate factor lifts to a multivariate factor (that is, no recombination is needed). That this is reasonable is due to Hilbert's irreducibility theorem, which shows that most integer substitutions for one variable preserve irreducibility over the rationals [29]. Were our factor not to lift in such a way, the algorithm would still work except we might need to lift to a higher degree; this is the same situation as that described in the remark following Corollary 2 in the previous section. An alternative would be to lift to half the total degree of the input, and be willing to try all univariate factors of degree less or equal to that. The point of either short cut is to reduce the lifting degree, as the higher ones clearly make for more strenuous Gröbner basis computations both in terms of polynomial degree and likelihood of encountering inadequate precision (the raising of which would incur a time penalty on all steps).

The next example is from [11].

Example 4

```
p1 = 48 x y^2 + 12 x^2 y^2 + 11 x^2 + 6 x^3 - 11 x^7 y^2 - 63 x^3 y^2 + 54 x^2 y^3 - 23 x^10 y^3 +
     58 x^4 y^4 + 18 x^9 y^2 - 42 x^5 y^5 - 80 x^11 y^2 + 10 y^10 x^4 + 14 x^4 y + 47 y^15 + 18 x^12 y^2 +
     36 y^13 - 23 x^3 y^12 - 68 x^8 y^3 - 41 x^4 y^7 + 72 x^3 y^11 + 22 x y^13 - 76 x^2 y^13 - 54 y^10 -
     18 y^9 + 79 x^7 y^3 - 49 x^7 y^4 + 9 x^5 - 75 x^6 y - 46 x^3 y^5 - 17 x^4 y^5 + 3 x^2 y^7 + 21 y^5;
p2 = -40 x^2 y^2 + 55 x y^2 + 36 x^3 y - 82 x^6 y^2 - 73 x^9 y - 27 x^5 y^5 + 18 x^7 y^2 + 31 x^2 y^3 + 54 y^7 y^3 -
     16 x^3 y^5 - 96 y^4 + 13 y^3 + 40 x^2 - 99 y^2 - 20 y^6 + 44 x^9 - 65 y^10 + 39 x^6 + 66 x y^5 - 34 x^3 y^3 +
     99 x^6 y^4 - 9 x^3 y^4 - 83 x^3 y^7 + 76 x^4 y^3 + 28 x^8 - 89 y^8 + 71 x y^8 + 38 x y^9 - 40 x^2 y^6 - 12 x^2 y^5;
prod = Expand[p1 p2];
Timing[factor = factorBivariate[prod, x, y, 2, 600]]
```

```
lift 4 times
{0.034994, 539.508}
{0.103985, 470.659}
{0.537917, 359.285}
{2.50162, 128.572}
{3.19152, -40 x^2 - 39 x^6 - 28 x^8 - 44 x^9 - 36 x^3 y + 73 x^9 y + 99 y^2 - 55 x y^2 + 40 x^2 y^2 + 82 x^6 y^2 -
  18 x^7 y^2 - 13 y^3 - 31 x^2 y^3 + 34 x^3 y^3 - 76 x^4 y^3 + 96 y^4 + 9 x^3 y^4 - 99 x^6 y^4 - 66 x y^5 +
  12 x^2 y^5 + 16 x^3 y^5 + 27 x^5 y^5 + 20 y^6 + 40 x^2 y^6 + 83 x^3 y^7 + 89 y^8 - 71 x y^8 - 38 x y^9 + 11 y^10}
```

We see that this recovers (up to sign) the factor *p2*.

```
Together[factor / p2]
-1
```

Example 5

Our final example is a modification of one appearing in [7]. The goal is to find the absolute factorization of a certain bivariate polynomial. Our modification subtracts $x^5 y + x^4 + x^3$ from the polynomial in the reference, so that it will factor nontrivially over some extension field (which is to be determined in the process).

```
poly = -3 - 16 x - 20 x^2 - 11 x^3 - 2 x^4 - 4 x^5 - 8 x^7 - 3 x^9 + 7 y - 16 x^2 y -
       16 x^3 y + 5 x^4 y - 4 x^5 y + 8 x^6 y - 4 x^7 y + 3 y^2 + 4 x y^2 - x^2 y^2 - 11 x^3 y^2 +
       3 x^4 y^2 - x^5 y^2 + 8 y^3 + 8 x y^3 + 5 x^2 y^3 - 5 x^3 y^3 + 6 x^4 y^3 + 8 x^6 y^3 + 6 y^4 +
       4 x y^4 + 4 x^2 y^4 - 10 x^3 y^4 + 3 x^4 y^4 + 3 y^5 + x^2 y^5 + 3 y^6 - 5 x^3 y^6 + 3 y^7 + y^9;
```

One can check that this polynomial is irreducible over the rationals. To proceed with the absolute factorization, we first substitute a value for one variable (we use *x*), and solve numerically to high precision for the roots of the resulting univariate. We show a low precision approximation to the first root in the list.

```
x0 = 11 / 7;
roots = y /. NSolve[poly /. x → x0, WorkingPrecision → 400];
root1 = First[roots];
N[root1]
```
```
-1.31386 - 1.39338 i
```

We now attempt to reconstruct the factor containing the first of our roots. We will just do a direct lift to degree 10, that is, modulo $\{(x - x0)^{10}, (y - root1)^{10}\}$. This suffices because any nontrivial factor must have a degree that divides 9 (since it gives the full polynomial when multiplied by its equal-degree conjugates). The only candidates are 1 and 3, so by prior theory a lift to degree $3^2 + 1 = 10$ suffices.

```
Timing[fac = First[ GroebnerBasis[{poly, (y - root1)^10, (x - x0)^10}, {y, x},
 MonomialOrder -> DegreeReverseLexicographic,
 CoefficientDomain -> InexactNumbers]];]
```
```
{0.542917, Null}
```

Here is a low precision version of our candidate factor.

```
Chop[N[fac]]
```
```
(1.68233 + 2.32308 i) + (0.682328 + 2.32308 i) x -
  (2.23279 - 0.792552 i) x^3 + 1. y + (0.341164 + 1.16154 i) x y + 1. y^3
```

Before we proceed to deduce the exact form of this result, we should check that it is a viable candidate factor. Specifically, it must divide our polynomial. We may verify this using generalized division. In *Mathematica* this is accomplished with PolynomialReduce.

```
PolynomialReduce[poly, fac, {y, x},
 CoefficientDomain -> InexactNumbers,
 MonomialOrder -> DegreeReverseLexicographic][[2]]
```
```
0
```

This is promising. We now take the numeric coefficients of our factor and attempt to recast them as algebraic numbers. For this we use RootApproximant along with a small amount of code to convert representations to all use a common algebraic number (RootApproximant uses functionality based on [10, 16]). We show the timing in seconds.

```
Timing[
 dtl = Chop[GroebnerBasis'DistributedTermsList[fac, {y, x}]];
 newdtl = MapAt[RootApproximant, dtl, Thread[{1, Range[Length[dtl[[1]]]], 2}]];
 rtlist = Cases[newdtl, Root[__], Infinity];
 firstrt = First[rtlist];
 newroots = Join[{firstrt}, ToNumberField[Rest[rtlist], firstrt]];
 newroots = newroots /. AlgebraicNumber[aa_, bb_] :>
    AlgebraicNumberPolynomial[AlgebraicNumber[aa, bb], aa];
 newdtl = newdtl /. Thread[rtlist → newroots];
 algfactor = GroebnerBasis'FromDistributedTermsList[newdtl]]
```
```
{0.388941, 1 + y + y^3 + 2 Root[1 + #1 + #1^3 &, 3] + 2 x Root[1 + #1 + #1^3 &, 3] +
   x y Root[1 + #1 + #1^3 &, 3] + x^3 (-1 + Root[1 + #1 + #1^3 &, 3]^2)}
```

We can check the result as follows. Irreducibility of *poly* over the rationals implies that all algebraic factors are conjugates of one another. We can explicitly form the product of these conjugates, then check that their product recovers the polynomial up to an invertible (that is, rational) factor. We show this below.

```
algprod = Product[algfactor /. Root[a_, 3, b___] :> Root[a, j, b], {j, 3}]
```

$$\begin{aligned}
&\left(1 + y + y^3 + 2\,\text{Root}\left[1 + \#1 + \#1^3 \&,\, 1\right] + 2\,x\,\text{Root}\left[1 + \#1 + \#1^3 \&,\, 1\right] + \right.\\
&\quad \left. x\,y\,\text{Root}\left[1 + \#1 + \#1^3 \&,\, 1\right] + x^3\left(-1 + \text{Root}\left[1 + \#1 + \#1^3 \&,\, 1\right]^2\right)\right)\\
&\left(1 + y + y^3 + 2\,\text{Root}\left[1 + \#1 + \#1^3 \&,\, 2\right] + 2\,x\,\text{Root}\left[1 + \#1 + \#1^3 \&,\, 2\right] + \right.\\
&\quad \left. x\,y\,\text{Root}\left[1 + \#1 + \#1^3 \&,\, 2\right] + x^3\left(-1 + \text{Root}\left[1 + \#1 + \#1^3 \&,\, 2\right]^2\right)\right)\\
&\left(1 + y + y^3 + 2\,\text{Root}\left[1 + \#1 + \#1^3 \&,\, 3\right] + 2\,x\,\text{Root}\left[1 + \#1 + \#1^3 \&,\, 3\right] + \right.\\
&\quad \left. x\,y\,\text{Root}\left[1 + \#1 + \#1^3 \&,\, 3\right] + x^3\left(-1 + \text{Root}\left[1 + \#1 + \#1^3 \&,\, 3\right]^2\right)\right)
\end{aligned}$$

Next we collect all terms in the polynomial, operating on the coefficients to obtain reduced forms. These will be algebraics of minimal degree; what we hope to get for all coefficients are rationals. We do, because the result cancels perfectly with *poly*.

```
Together[Expand[RootReduce[Collect[Expand[algprod], {x, y}]]] / poly]
```

```
1
```

We remark that there are other good ways to recover exact algebraic coefficients, if one first finds the set of all approximate factors. One such is presented in [6], where an analysis of needed precision is also provided.

We also observe that the method shown above, while perhaps not as efficient as those of [4, 21], is nonetheless faster than *Mathematica*'s current built in capabilities. Moreover these latter require that one know in advance the extension field. We show this explicitly below (the method will only split off one full factor, but that suffices to recover its conjugates).

```
Timing[fax = FactorList[poly, Extension→ Root[1 + #1 + #1^3 &, 3]];]
```

```
{0.678896, Null}
```

We also note that while this might not provide a multivariate absolute factorization method that scales well as the number of variables grows, it can at least quickly determine irreducibility (with high probability). This would be effected simply by deciding whether there are numeric approximate factors after substitution for all but two variables. Lifting of all variables and exact recovery of nontrivial factors, if any, would not be needed in such a scenario; we only need lift in one variable, and that only approximately, in order to determine whether a given input is irreducible. A practical, nontrivial example from constraint geometry may be found in [19]. The polynomial in question is trivariate, of total degree 16 and with all exponents even. It is dense subject to those constraints, and has coefficients ranging from two to ten digits in size.

7. FAILURE MODES AND VERIFICATION OF RESULTS

There are three failure modes for our use of finite precision. One is to run out of precision in the Gröbner basis phase, and abort computation. Another is to compute a Gröbner basis that has the wrong structure, and thus either fail to find an approximately correct factor or else find a putative factor that is entirely wrong. A last type of failure is to find a factor that is approximately correct, but does not have sufficient precision from which to recover an exact form thereof.

The first form of failure is typically handled by increasing the initial precision a few times, until either we obtain a result or else reach some upper bound. In this case we have a "do not know the result" situation, which is at least preferable to an incorrect result.

Now suppose we have obtained a putative approximate factor or gcd. A first question is to decide whether it is in fact a viable approximation to an exact one. This determination can typically be made by polynomial division, checking that coefficients in any remainder are suitably small. If we have a reasonable approximation, we can often improve it by local numerical optimization methods. One uses the coefficients of the approximated factor as a starting point in order to improve the precision. The objective function could be, for example, a sum of squares of coefficients in the remainder on division.

If we have an approximate factor or gcd that, on division, is clearly seen to be not viable, this also can be treated as an inconclusive case. We now remark on how this case can arise. In the process of computing an approximate Gröbner basis, we might err in a cancellation of terms, such that we decide a leading coefficient is zero. This is the only way to get a result that is structurally incorrect, that is, has the wrong polynomial skeleton. We do not have a theoretical argument to quantify how rare this might be. We can, however, say a bit about practical experience. In over a decade of operation, the numeric Gröbner basis capabilities of *Mathematica*'s `NSolve` have never shown this type of failure in problems that arise in-house, from users of the software, or from testing done on benchmark problems from the literature.

As with any numeric method one can likely create a problem for which it will fail if run at too low precision. It seems that such problems do not arise with any frequency in practice. We can offer a tentative explanation for this. Recall theorem 1: For a give problem and a given output precision, there exists an input precision such that computing a Gröbner basis at that initial precision, using significance arithmetic, will give a result that is correct to the specified output precision. While it is impractically high for realistic problems it seems that a more modest precision typically suffices. And when it does not suffice we almost always lose too much precision and abort. In order to obtain a skeletally incorrect result we would have to have a catastrophic cancellation. This means that what should be a nonzero term (e.g. in an exact or at least sufficiently high precision computation) appeared to be zero. In order for cancellation at all significant places to occur we effectively require polynomials with coefficients that are commensurately far apart in scale. When the precision is in, say, the hundreds of digits, we simply do not see such polynomials in practical settings. As precision in the computation degrades it is of course more likely that we might encounter such a scale difference. But there is only a narrow window for this error to manifest as an incorrect cancellation; at sufficiently low precision (a few digits, say) the algorithm will go into the first mentioned failure mode, and abort. The conclusion is that we are much more likely to abort than to give an incorrect result.

We also observe that, should such a problematic case arise, the most likely manifestation in our setting would be a nontrivial result that is demonstrably wrong. This is because a leading term catastrophic cancellation is more likely to yield polynomials in the result that are too small—in term order—rather than too large. Hence we obtain a nontrivial putative result that fails to verify, rather than a claim that there is no nontrivial result.

These various observations support the view that an unverifiable wrong result (say, a proper divisor that is not greatest, or a factorization that is further factorizable) will be quite rare.

8. SUMMARY AND FURTHER DIRECTIONS

We have indicated several ways in which approximate Gröbner bases might be used to advantage to obtain exact, verifiable results to problems in computational algebra. Applications include finding polynomial greatest common divisors and multivariate polynomial factorization. For this latter we indicate how to proceed both for ordinary factorization over the rationals, and for absolute factorization over the algebraic closure of the rationals.

While our methods seem to be slower than those of e.g. [4, 6] and subsequent refinements thereof, they are still useful in practice. An added advantage is that they are straightforward to implement. Indeed, the *Mathematica* code used for the examples was, in total, but a few dozen a few lines. The quadratic lifting time seems to approximately double with each step. This is in accord with what one might expect using standard lift methods. There are at least two caveats. One is that we have no proof that the lifting complexity will behave this way in general. Another is that use of finite precision and significance arithmetic means our approach can fail and abort computation should we lose too much precision.

There are many open areas that seem worthy of further investigation. We indicate several and also mention some preliminary findings.

- Automate selection of precision or at least develop some useful heuristic to assess in advance an amount that will likely succeed for the computation at hand. This is currently a wide open topic.
- Modify these methods so that they might be used to tackle problems of an approximate nature. Inputs might only be known to a few digits, and we seek solutions to "nearby" problems that have nontrivial results (e.g. common divisors larger than 1, or more than one factor). Methods such as those shown in [18] can find approximate gcds in this setting. But they tend to be slower than the methods of this paper and they rely on heuristic choices of tolerance. A small amount of testing to date has indicated that some of the approximate Gröbner basis ideas from [18] might carry over to the setting of the present work. This is a topic that needs further experimentation.
- Apply these methods to the setting of finite fields, in such a way that they remain reasonably fast even when the characteristic is small. The problem, in this case, is that one might need to work in an algebraic extension. This can be costly for Gröbner basis computations. When such an extension is needed it is by no means clear that this approach can be made practical (it involves another variable and another polynomial, and this will typically make the computation of the needed Gröbner bases slower). In cases where no extension is required (that is to say, hypotheses of theorems can be satisfied with appropriate substitutions from the finite field itself), one would expect behavior better than that indicated in this paper. The reason is that the coefficient arithmetic is now bounded by the modulus size and moreover there is no possibility of loss of precision.
- In the reverse direction, it might be possible to work over a large finite field and then lift to a result over the rationals. This would allow one to avoid approximate arithmetic although there is then the issue of avoiding primes that are unlucky for the given input.
- During lifting steps many of the Gröbner basis inputs seem to be close, in some sense, to bases with respect to a lexicographic term order. It would be interesting to see if conversion methods such as the Gröbner walk are effective for the computation that takes us to the desired term order. Preliminary tests show that the walk can be faster than what we did in our examples, but that it is subject to more extreme loss of precision. This, presumably, is from the conversion steps that occur at each cone traversal during the walk. It is an open problem to determine if this loss of precision can be avoided or at least curtailed.
- One possibility for alleviating precision loss in intermediate lift steps is to recompute, from the approximate bases, their exact counterparts. When precision is sufficient for this task, we might then numericize anew at the original precision. Our preliminary finding is that this tactic does help, but only to a small extent when using the Gröbner walk. All the same, this provides some improvement in that it allows for a lower initial precision, regardless of what method is used to compute the Gröbner bases for the lifting steps. Also one could try heuristics that gauge the precision loss at a given lifting step, and ratchet to accordingly higher precision in preparing for the next step. We must also observe that this tactic is far more likely to succeed when working over rationals than when doing absolute factorization. The reason is as follows. Given the same initial input, one will typically require far less precision to recover rationals for the bases produced in lift steps, than that which would be needed for algebraic number recovery.
- As a referee observed, using a shear transformation to obtain a monic polynomial will destroy sparsity of input. We can avoid such transformations, in some of the algorithms presented above, at the cost of requiring higher degrees for lifting. It would be useful to understand the tradeoffs in efficiency between retaining sparsity vs. having lower lift bounds. It would be particularly useful to find ways to avoid the linear coordinate change and still have optimal lifting degree bounds.
- We discussed various modes of failure. It would be useful, though probably quite difficult, to have a better theoretical understanding of the interplay between initial precision and erroneous results.
- The examples of this paper, together with their timings, provide a proof of concept that the methods presented are viable. It would of course be nice to further improve on speed (above and beyond ideas noted in the preceding items). A recent line of inquiry, in regard to the absolute irreducibility testing discussed in the previous section, is to understand situations in which we might have an early termination during the quadratic lift process. This is under current investigation.

9. REFERENCES

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

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

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

[4] G. Chèze (2004). Absolute polynomial factorization in two variables and the knapsack problem. Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation (ISSAC 2004), 87-94. J. Gutierrez, editor. ACM Press.

[5] G. Chèze and A. Galligo (2005). Four lectures on polynomial absolute factorization. Solving Polynomial Equations, Volume 14 of Algorithms and Computation in Mathematics, pp. 339-392. Springer Berlin Heidelberg.

[6] G. Chèze and A. Galligo (2006). From an approximate to an exact absolute polynomial factorization. Journal of Symbolic Computation 41:682-696.

[7] R. Corless, A. Galligo, I. Kotsireas, and S. Watt (2002). A geometric-numeric algorithm for absolute factorization of multivariate polynomials. Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation (ISSAC 2002), 37-45. T. Mora, editor. ACM Press.

[8] R. Corless, A. Giesbrecht, M. van Hoeij, I. Kotsireas, and S. Watt (2001). Towards factoring bivariate approximate polynomials. Proceedings of the 2001 International Symposium on Symbolic and Algebraic Computation (ISSAC 2001), 85-92. B. Mourrain, editor. ACM Press.

[9] 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.

[10] H. Ferguson, D. Bailey, and S. Arno (1999). Analysis of PSLQ, an integer relation finding algorithm. Mathematics of Computation 68(225):351-369.

[11] A. Galligo and S. Watt (1997). A numerical absolute primality test for bivariate polynomials. Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation (ISSAC 1997), 217-224. W. Küchlin, editor. ACM Press.

[12] P. Gianni and B. Trager (1985). Gcd's and factoring multivariate polynomials using Gröbner bases. Proceedings of the 1985 European Conference on Computer Algebra (EUROCAL 1985), B. Caviness, editor. Lecture Notes in Computer Science 204, 409-410. Springer-Verlag.

[13] E. Kaltofen (1990). Computing the irreducible real factors and components of an algebraic curve. Applicable Algebra in Engineering, Communication and Computing 1:135-148.

[14] J. Keiper (1992). Numerical computation with *Mathematica* (tutorial notes). http://library.wolfram.com/infocenter/Conferences/4687/

[15] A. Kondratyev, H. Stetter, and F. Winkler (2004). Numerical computation of Gröbner bases. Proceedings of the 7th Workshop on Computer Algebra in Scientific Computing (CASC 2004), St.Petersburg, 295-306, V. G. Ghanza, E. W. Mayr, E. V. Vorozhtov (eds.), Technische Univ. München.

[16] A. K. Lenstra (1984). Polynomial factorization by root approximation. Proceedings of the International Symposium on Symbolic and Algebraic Computation (EUROSAM 1984), J. Fitch, editor. Lecture Notes in Computer Science 174, 272-276, Springer-Verlag.

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

[18] D. Lichtblau (2008). Approximate Gröbner bases and overdetermined algebraic systems. Manuscript. http://library.wolfram.com/infocenter/Conferences/7536/

[19] D. Lichtblau (2010). Midpoint locus of a triangle in a corner. ADG 2010, Munich. http://lsiit.u-strasbg.fr/adg2010/upload/2/29/ADG2010_Daniel_TiC_talk.pdf

[20] M. Noro and K. Yokoyama (2002). Yet another practical implementation of polynomial factorization over finite fields. Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation (ISSAC 2002), 200-206. T. Mora, editor. ACM Press.

[21] T. Sasaki (2001). Approximate multivariate polynomial factorization based on zero-sum relations. Proceedings of the 2001 International Symposium on Symbolic and Algebraic Computation (ISSAC 2001), 284-291. B. Mourrain, editor. ACM Press.

[22] T. Sasaki and F. Kako (2007). Computing floating-point Gröbner bases stably. Proceedings of the 2007 International Workshop on Symbolic–numeric Computation (SNC 07), 180-189. ACM Press.

[23] T. Sasaki, M. Suzuki, M. Kolář, and M. Sasaki (1991). Approximate factorization of multivariate polynomials and absolute irreducibility testing. Japan Journal of Industrial and Applied Mathematics 8(3):357-375.

[24] K. Shirayanagi (1996). Floating point Gröbner bases. Mathematics and Computers in Simulation 42:509-528. Elsevier Science Ltd.

[25] M. Sofroniou and G. Spaletta (2005). Precise numerical computation. Journal of Logic and Algebraic Programming 64(1):113-134.

[26] Q–N Tran (2007). A new class of term orders for elimination. Journal of Symbolic Computation **42**:533-548.

[27] C. Traverso and A. Zanoni (2002). Numerical stability and stabilization of Groebner basis computation. Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation (ISSAC 02), 262-269. T. Mora, editor. ACM Press.

[28] Wolfram Research, Inc. (2008). Champaign, Illinois. *Mathematica* 7. http://www.wolfram.com

[29] R. Zippel (1993). Effective Polynomial Computation. Kluwer Academic Publishers.