Approximate Gröbner Bases and Overdetermined Algebraic Systems

Daniel Lichtblau

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

danl@wolfram.com

ABSTRACT. We discuss computation of Gröbner bases using approximate arithmetic for coefficients. We show how certain considerations of tolerance, corresponding roughly to accuracy and precision from numeric computation, allow us to obtain good approximate solutions to problems that are overdetermined. We provide examples of solving overdetermined systems of polynomial equations. As a secondary feature we show handling of approximate polynomial GCD computations, using benchmarks from the literature.

Categories and Subject Descriptors

G.1.0 [**Numerical Analysis**]: General−−− Multiple precision arithmetic; Numerical algorithms; G.1.5 [**Numerical Analysis**]: Roots of Nonlinear Equations−−− Systems of equations; I.1.2 [**Symbolic and Algebraic Manipulation**]: Algorithms−−− Algebraic Algorithms;

General Terms

Algorithms

Keywords

Gröbner basis, algebraic systems, hybrid symbolic−numeric computation

1. INTRODUCTION

Gröbner bases provide a means for solving a myriad of problems in computational algebra. In their original form, all arithmetic was carried through exactly, on rational numbers. This was necessary in order to know when combinations of polynomial coefficients cancel. In 1993 Shirayanagi [26] indicated a means to use approximate arithmetic and still handle this "zero recognition" problem. Since that time several approaches have appeared that use approximate arith− metic [4, 5, 19, 24, 27, 29, 32]. The advantages to approximate arithmetic are several. First is that it avoids intermediate coefficient swell one often observes in exact Gröbner basis computations. A second reason is that, in many problems, one works with approximate data and does not have access to exact values. Moreover, approximation by rationals might lead to intermediate swell, and still not improve on a solution based on the original approximate input values.

Work on numeric Gröbner bases begins with [26, 27]. The method therein for controlling error is a bit similar to what we will describe in section 2. It uses bookkeeping to measure loss of precision from arithmetic operations (similar ideas are discussed in [24]). The handling of coefficients can thus be viewed as an extension of floating point arithmetic. Traverso and Zanoni [30] describe use of both a modular image and a numeric approximation for coefficients. Only when both forms give zero (approximate, in the latter case) do we regard an apparent cancellation as truly zero. A drawback is this requires exact input initially, or else a system that is not overdetermined. Other approaches include extending the notion of Gröbner bases to allow head terms with small coefficients to become non−head terms [17, 18, 29]. References [4, 30] contain some discussion of the overdetermined case. Use of syzygies to determine vanishing of coefficients is described in [4, 5], with the latter describing a possible algorithmic treatment.

A typical situation in which one might desire to work with approximate coefficients is in solving polynomial systems of equations. A common method for this utilizes Gröbner bases [2, 7, 9, 11, 20] (the `NSolve` function of *Mathematica* is based on this approach). We start by discussing in brief some of the issues associated with the Gröbner basis part of the computation, when carried out using approximate arithmetic. We go on to point out weaknesses that appear when such systems are overdetermined. We then describe precision and accuracy tolerancing, and show how they can be used to address such weaknesses.

This work provides a description and empirical study of methods that extend Gröbner bases to handle approximately consistent polynomial systems. Many of the ideas we show have been developed independently in the cited literature. Our main contribution is to show how they can be made to work quite effectively on challenging problems. We provide examples, many considered difficult, from the literature on numerical polynomial system solving and approximate GCD computation. These serve to indicate the merit of the methods we describe.

2. APPROXIMATE GRÖBNER BASES IN POLYNOMIAL SYSTEMS

We begin with the observation that there are two variants of approximate Gröbner basis computations. In one we assume that coefficients of input are exactly known, and we use approximate numbers in order to avoid either intermedi−ate swell of integers, or difficult computations with algebraic numbers. Here we are interested in a different setting: coefficients are known only approximately, and moreover we may have an overdetermined system. The rest of this section pertains to both settings. The sequel is then devoted to the case of interest. The first scenario is discussed in a companion paper to the present one [21].

Gröbner bases computation using approximate arithmetic can be subject to several problems. First, as noted above, is the issue of recognizing when a cancellation has occurred. The model of approximate arithmetic we use, significance arithmetic, turns out to be quite good at handling this. Indeed, over a decade of experience suggests this is not, in and of itself, a problem, provided we do not work with an overdetermined system [20]. The essential idea [16, 28] is that numbers carry with them an estimate of error. Standard arithmetic such as addition and multiplication of such numbers propagates error via first order approximations. 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. (Use of "approximated" to describe the error is intentional; this is effectively a first order approximation to interval arithmetic.) The upshot is that, in contrast to the approach of [26], we require no careful bookkeeping; the internal arithmetic does this automatically.

A secondary issue is that, with this choice of arithmetic, precision gradually erodes over the course of a computation (as first order error estimates grow). What this means in practice is that often one must start with high precision input (say, a few hundred digits). Clearly this is well beyond the precision one can expect from input that arises as measurements of data. Again, when the problem at hand is not overdetermined, this is not a serious issue. One simply adds digits, arbitrarily, to the input coefficients. If the problem is not ill conditioned then when finished we know we have solved a nearby system. In practice one observes that residuals from such a solution, used in the original input, are typically small. If so desired, they can be further improved via local refinement methods.

Yet another problem, one particularly associated to use of significance arithmetic, is that in rare cases a decision might be made that a full cancellation took place, when in an exact computation perhaps a small but nonzero value would be obtained. This is discussed in [20]. It turns out to be a relatively unimportant issue in that it is uncommon, it is usually correctable by moving to higher precision, and generally only causes loss of huge solutions (consider the difference between solutions of $x^2 = 1$ and $\frac{1}{10^{40}} x^5 + x^2 = 1$).

We end this section with a brief historical note. As mentioned earlier, the first reported implementation of numerical Gröbner bases (of which this author is aware) is due to Shirayanagi [26] from 1993. This article discusses a bookkeep−ing approach to precision control that involves what are called "bracket coefficients". This approach is similar in spirit, if not details, to significance arithmetic. It was this article (both methodology and results) that motivated the author to implement numerical Gröbner bases in late 1993, in what would eventually become version 3 of *Mathematica*. This implementation is discussed briefly in [19]. We point out that it in effect requires either exact or at least "nice" (e.g. high precision) input. It will not handle input that is known only to a few digits of precision and also overdetermined. To handle these situations we require the tolerancing approach of the present article.

3. OVERDETERMINED SYSTEMS

We have just given a brief overview of how we can manage approximate coefficient arithmetic reliably when handling non−overdetermined (and reasonably well conditioned) systems. Indeed this suffices for many practical sorts of compu−tations. But there is a growing body of literature involving overdetermined systems. It thus becomes important to consider ways in which Gröbner bases can be extended to address them. To motivate this we begin by describing a few sources of such systems.

One place where overdetermined problems are encountered is in best−fitting of data. While local methods are typically used, there are cases where one might not have adequate information to give a starting point such that convergence will be attained. For these situations one can utilize an approximate solution to an overdetermined system, obtained with help of a Gröbner basis computation.

A related common scenario is when one uses an overdetermined system in order to rule out undesired solutions. An example is in camera pose estimation [22], where one or more extra reference points are used in order to remove from consideration undesired solutions to a possibly ill conditioned problem. The problem encountered is that, due to the approximate nature of coefficients, use of arithmetic as described in section 2 would often lead to an empty solution set. What we require, and will describe, are more tools to decide that coefficients are "small enough" to be regarded as zero.

Another source of overdetermined problems arises in trying to find "approximate" polynomial greatest common divi−sors [8, 12, 14, 15, 23, 25]. In this setting one typically wants a result that is of highest possible degree, subject to constraints on coefficient sizes in remainders (after normalizing say, by making all leading coefficients unity). Most approaches in the literature use matrix methods (i.e. singular value decompositions) or remainder sequences for this sort of computation. We will instead adapt Gröbner basis methods from the exact realm to do these.

4. ARITHMETIC CONSIDERATIONS IN SOLVING OVERDETERMINED SYSTEMS

Once we go from an exactly determined to an overdetermined systems, high precision approximate arithmetic in computing a Gröbner basis no longer alone suffices to catch cancellation of coefficients. The problem is that we need to expand the size of what we might regard to be zero, as it is now on a scale with the precision of our input.

We are thus faced with a situation where we need to "coarsen" our classification of what will be regarded as full cancellation. We note that one must be a bit careful in terminology at this point; "zeros" can refer to approximate solutions to a system of equations, or to coefficient combinations that cancelled (see [13] for discussion of approximate zeros, also referred to as "pseudozeros"). Typically our interest is in the former, and the latter appear as a byproduct to a successful navigation of the computation.

We discuss in brief notions of tolerance, accuracy, and precision. This is entirely informal; the purpose is simply to motivate our approach to zero recognition. By tolerance we typically have in mind a small threshhold, below which we regard values as zero. Precision is a relative concept, in which one considers ratio of (estimated) error to value (often this is referred to as "relative" error). Accuracy, by contrast, refers to the absolute magnitude of error.

Recall that the key operations in Gröbner basis computations are forming of S−polynomials and reduction thereof [1, 3, 6, 10]. Our main concerns are twofold. We do not want to retain leading coefficients that, in an exact computation, would have vanished. And we do not want polynomials that should have vanished in their entirety. In practice, each of these can happen if we do not employ some tactics for recognizing cancellation. We emphasize that the second situation is not a special case of the first; this indeed gets to the heart of the double−tolerance approach we will describe. Loss of a leading coefficient is analogous to a precision issue: one coefficient is notably smaller than most or all others. In contrast, an entire polynomial might be regarded as a full cancellation in a situation where all coefficients are comparable in size, but small in an absolute sense when compared with normalized inputs that gave rise to them.

Our approach to handling these cases is simple. We will have a "precision" tolerance and an "accuracy" tolerance. All manipulations involve addition of pairs of polynomials. Prior to that we find the average magnitude of the coefficients in these polynomials (we'll call this IPCA, for "input polynomial coefficient average"). If, after addition, a resulting coefficient is less than the precision tolerance times IPCA, we regard it as zero and remove it. If instead all coefficients are smaller than the accuracy tolerance times IPCA, then we regard the entire resulting polynomial as zero. We employ the "precision" mode to remove coefficients that are small relative to other coefficients. We use the "accuracy" notion to justify removing an entire polynomial when all coefficients are small in absolute magnitude (this does, however, assume some sort of normalization is in place for the polynomials that gave rise to it).

As a practical matter working with these tolerances can be troublesome. For example there are many problems where, even after scaling of variables, coefficient sizes will be orders of magnitude apart. Thus a precision tolerance can remove coefficients that are actually needed. Cases where no precison can discern between coefficients to keep and ones to discard are, for this method, ill conditioned.

The accuracy tolerance is typically less prone to misuse, at least in the sorts of examples we show. That said, some of the examples required trial−and−error selection of tolerances in order to attain good results. On a brighter note, many do not, and one can often base a sensible setting on the precision of the input. Typical values for the sort of problems in the examples, with machine numbers for input, tend to be around 10^{-8} for precision and 10^{-3} for the accuracy tolerance.

We mention that Kondratyev and coauthors [17, 18, 29] have a different way of handling the problem of small leading coefficients. Their "stabilized Gröbner bases" retain such terms but bypass them for purposes of forming S−polynomials. Also Traverso and Zanoni [30] describe a hybrid arithmetic in terms of what they call t_1 and t_2 tolerances. The second appears to serve the same purpose as the precision tolerance discussed above, and the first is similar to the accuracy tolerance. Moreover, Sasaki and Kako [24] used ideas similar to our accuracy tolerancing for the detection of zero polynomials.

5. EXAMPLES

All examples were run using the version of *Mathematica* under development at the time of this writing. Auxiliary code is provided in an appendix, as are inputs for several of the longer examples. Where we use pairs of tolerance values, the first denotes precision and the second accuracy. When one appears alone, it is interpreted as a precision tolerance.

EXACTLY DETERMINED SYSTEMS

We begin with some classical numeric systems that are not overdetermined, in order to indicate that no special handling is needed (at least for the Gröbner basis phase of the computations). These provides a sort of baseline for contrast with later computations. First we will show the Cassou−Noguès system. We require high precision for the eigensystem phase of the solver, hence the nondefault `WorkingPrecision` specification.

```
Timing[Length[solnsCassou = NSolve[polysCassou == 0, WorkingPrecision → 200]]]
{0.236015, 10}
```

We check that the residuals are indeed small.

```
Max[Abs[polysCassou /. solnsCassou]]
```

$0. \times 10^{-145}$

Observe that the resulting residuals, while small, are many times larger than the precision. This simply indicates that precision loss occurred in parts of the computation. We show another example, one that is considerably slower: the Caprasse system. It is troublesome because it has multiplicity of several roots, and moreover the multiplication (endomorphism) matrices utilized in the solver are derogatory.

```
Timing[Length[solnsCaprasse = NSolve[polysCaprasse]]]
{14.7329, 56}
```

```
Max[Abs[polysCaprasse /. solnsCaprasse]]
```

2.84217×10^{-14}

Last we show a small perturbation of this troublesome system. This moves the system to one that is nearly but not exactly derogatory. The numerical solver again obtains good results in reasonable time.

```
Timing[Length[solnsCaprasseModified = NSolve[polysCaprasseModified]]]
{3.53622, 56}
```

```
Max[Abs[polysCaprasseModified /. solnsCaprasseModified]]
```

2.80977×10^{-10}

We now describe a system that is exactly determined, but shows quite interesting behavior if not handled with toleranc−ing. It comes from [31]. As the input is long we will not show it, but simply describe the problem. It has nine polynomi−als in nine variables, and describes configurations of a certain type of Stewart platform. The coefficients are machine double precision complex numbers.

With default settings, `NSolve` will find 80 solutions. This is twice the number claimed at [31], and moreover half of the solutions give large residuals and are themselves large. This makes one suspect they are erroneous. However, raising the precision of the input and solving then gives 80 solutions, now all with modest residuals; this tells us there is indeed a "nearby" system for which all 80 solutions are valid.

The crux is that the input describes a numerically unstable situation, wherein coefficients need to satisfy certain hidden constraints in order to properly specify the type of platform in question. In making numerical coefficients, they become perturbed slightly and now we have a system with more solutions. Those that give large residuals, at machine precision, are in fact not wanted; they are the artifacts of having approximated the polynomial coefficients.

The tolerancing that repairs this is quite straightforward. We use a precision value of 10^{-10} and an accuracy of 10^{-3}. The overall result: we get the desired 40 solutions, and a factor of 6 speed improvement because extra work is needed internally to get the "large" solutions to have acceptable residuals..

OVERDETERMINED AND ILL CONDITIONED SYSTEMS

We start with an example presented in [8]. We seek approximate singular points on a curve given implicitly as the zero set of a certain polynomial. This is simply a matter of finding points for which the polynomial and its two first deriva−tives all (approximately) vanish.

```
poly = 4.0 y^4 + 17.0 x^2 y^2 + 13.07 x y^2 -
   19.572938 y^2 + 4.0 x^4 + 5.228 x^3 - 18.29175 x^2 - 5.228 x + 15.29175 ;
```

$$\texttt{NSolve}\left[\{\texttt{poly}, \partial_x \texttt{poly}, \partial_y \texttt{poly}\}, \{x, y\}, \texttt{Tolerance} \to \frac{1}{10^2}\right]$$

```
{{x -> 1.18339, y -> 0.}}
```

We now show an overdetermined camera pose problem from [22]. Here we need to raise precision artificially so that the `GroebnerBasis` step can run to completion (when precision of any coefficients becomes too low, it gives up). We postprocess by chopping off smallish imaginary parts.

```
coords = {{1, 2, 1.49071, 4}, {1, 3, .400000, 8}, {1, 4, .894427, 4},
   {2, 3, 1.49071, 4}, {2, 4, .666667, 8}, {3, 4, .894427, 4}};
vars = Array[x, 4];
polys = MapThread[x[#1]^2 + x[#2]^2 - #3 x[#1] x[#2] - #4 &, Transpose[coords]]
{-4 + x[1]^2 - 1.49071 x[1] x[2] + x[2]^2, -8 + x[1]^2 - 0.4 x[1] x[3] + x[3]^2,
 -4 + x[1]^2 - 0.894427 x[1] x[4] + x[4]^2, -4 + x[2]^2 - 1.49071 x[2] x[3] + x[3]^2,
 -8 + x[2]^2 - 0.666667 x[2] x[4] + x[4]^2, -4 + x[3]^2 - 0.894427 x[3] x[4] + x[4]^2}

Chop[soln = NSolve[polys, vars, Tolerance → {10^(-3), 0}, WorkingPrecision → 8],
 10^(-3)]
{{x[1] → 2.23606, x[2] → 2.99999, x[3] → 2.23607, x[4] → 0.999999},
 {x[1] → 2.23606, x[2] → 2.99999, x[3] → 2.23607, x[4] → 0.999999},
 {x[1] → -2.23606, x[2] → -2.99999, x[3] → -2.23607, x[4] → -0.999999},
 {x[1] → -2.23606, x[2] → -2.99999, x[3] → -2.23607, x[4] → -0.999999}}
```

We check that the worst residual is not terribly large.

```
Max[Abs[polys /. soln]]
0.000064876
```

UNIVARIATE APPROXIMATE GCD

There is a vast literature on ways to compute approximate polynomial GCDs. Most involve reformulations as linear algebra problems, and make use of numeric algorithms well suited to computing matrix rank reliably in the presence of approximation input. For background on such methods, see [8, 12, 14, 15] and references therein. We do not propose that the methods to be shown below are faster or more reliable. But they are to an extent automated (once the working precision and tolerances are selected), use very simple code, and give reasonable results quite quickly.

For univariate polynomials it is well known that we can extract a GCD via simple Gröbner basis computation. This is in effect a form of polynomial remainder sequence, and thus bears similarity to the univariate case of the method discussed by Sasaki and Sasaki in [25].

Here is an example from [8] . With a precision tolerance of two digits we recover a nontrivial approximate GCD.

```
p1 = x^14 + 3.00001 x^10 - 7.99998 x^7 - 25.00002 x^6 + 3.00001 x^13 +
   9.00006 x^9 - 3.00001 x^5 - 2.00001 x^8 - 6.00005 x^4 + 16.00004 x + 2.00001;
p2 = x^13 - 3.00003 x^9 - 2.99999 x^6 + 2.99999 x^12 - 9.00006 x^8 -
   8.99997 x^5 - 1.99998 x^7 + 5.99999 x^3 + 5.99994;
First[N[GroebnerBasis[setCoefficientPrecision[{p1, p2}, 50],
   x, CoefficientDomain → InexactNumbers , Tolerance → 1/10^2]]]
-2.00015 + 3.00024 x^5 + 1. x^6
```

We see it corresponds closely to the GCD of the "obvious" polynomial pair formed by rounding coefficients.

```
First[GroebnerBasis[{p1, p2} /. a_Real :→ Round[a], x]]
-2 + 3 x^5 + x^6
```

Now we show an example from [12], wherein we look for approximate multiple factors by taking the GCD of a polynomial with its derivative. Using coarse tolerancing we get a common factor of degree 6, in agreement with that reference.

```
poly = x^9 - (5.833333 + 2.333333 ⅈ) x^8 + (12.888889 + 11.7222222 ⅈ) x^7 +
   (-13.416667 - 24.694444 ⅈ) x^6 + (5.293210 + 28.703704 ⅈ) x^5 +
   (2.389403 - 20.183642 ⅈ) x^4 + (-3.790123 + 8.750857 ⅈ) x^3 +
   (1.880630 - 2.247914 ⅈ) x^2 + (-.452884 + .299535 ⅈ) x + (.045217 - .013868 ⅈ);
```

```
Chop[N[First[GroebnerBasis[setCoefficientPrecision[{poly, D[poly, x]}, 40],
    x, CoefficientDomain → InexactNumbers , Tolerance→ {1/10^4, 1/10^2}]]]]]
(0.013033486039440238 + 0.24739461042947114 ⅈ) -
 (0.3199779084192973 + 1.7517086950261698 ⅈ) x +
 (1.9659824714996788 + 5.134785008354451 ⅈ) x^2 -
 (5.221808113608173 + 7.66519441636273 ⅈ) x^3 +
 (6.888250445595586 + 5.721144296886046 ⅈ) x^4 -
 (4.33302767357858 + 1.6663679299044327 ⅈ) x^5 + 1. x^6
```

MULTIVARIATE APPROXIMATE GCD

Multivariate polynomial approximate GCDs algorithms are presented in [8, 12, 14, 15, 23, 33]. They tend to use matrix methods or polynomial sequences,. We instead take an elimination ideal method from [1], using approximate Gröbner bases as the main computational engine to get the (approximate) LCM. We follow with generalized division to extract the GCD. Note that we make no effort to locally improve the result, e.g. by Newton's method.

We first show example exF07 from [14]. This is relatively straightforward insofar as the input, if rationalized, has a nontrivial (exact) GCD. The actual inputs are a bit long to display, but are available as the set `cleanF7_list` at the URL in the references.

```
Timing[fgcd = floatPolynomialGCD[exF07polys[[1]], exF07polys[[2]], {1/10^8, 1/10^4}]]
{0.816051,
 2. + 6. x + 10. x^2 + 8. x^3 - 2. x^4 + 7.64022 × 10^-11 y + 8. x y + 2.54509 × 10^-13 x^2 y -
   8. x^3 y - 8. y^2 - 8. x y^2 - 10. x^2 y^2 + 8. y^3 - 4. x y^3 - 6. y^4 + 6. z - 10. x z - 10. x^2 z +
   2.09215 × 10^-14 x^3 z + 10. y z + 8. x y z + 4. x^2 y z - 4. y^2 z + 2. x y^2 z - 8. y^3 z + 6. z^2 -
   4. x z^2 + 2. x^2 z^2 - 10. y z^2 - 10. x y z^2 - 10. y^2 z^2 - 2. z^3 + 4. x z^3 - 6. y z^3 - 2. z^4}
```

Here is an example from [23].

```
c[x_, u_, n_] := (x + Sum[u[j]^j, {j, n}] + 1)^2

f2[x_, u_, n_] := (x^2 - Sum[u[j], {j, n}] - .5)^2

g2[x_, u_, n_] := (x^2 + Sum[u[j], {j, n}] + .5)^2
```

We create a pair of polynomials with proscribed GCD. We readily recover it using approximate arithmetic.

```
f[5] = Expand[f2[x, u, 5] * c[x, u, 5]];
g[5] = Expand[g2[x, u, 5] * c[x, u, 5]];
Timing[floatPolynomialGCD[f[5], g[5], {1 / 10^6, 1 / 10^2}]]
{8.27652, 1. + 2. x + 1. x^2 + 2. u[1] + 2. x u[1] + 1. u[1]^2 + 2. u[2]^2 +
   2. x u[2]^2 + 2. u[1] u[2]^2 + 1. u[2]^4 + 2. u[3]^3 + 2. x u[3]^3 + 2. u[1] u[3]^3 +
   2. u[2]^2 u[3]^3 + 1. u[3]^6 + 2. u[4]^4 + 2. x u[4]^4 + 2. u[1] u[4]^4 +
   2. u[2]^2 u[4]^4 + 2. u[3]^3 u[4]^4 + 1. u[4]^8 + 2. u[5]^5 + 2. x u[5]^5 +
   2. u[1] u[5]^5 + 2. u[2]^2 u[5]^5 + 2. u[3]^3 u[5]^5 + 2. u[4]^4 u[5]^5 + 1. u[5]^10}
```

Here we see that, with some amount of noise thrown in, we can still recover a reasonable approximate GCD.

```
fnoise[5] = Expand[f2[x, u, 5] * (c[x, u, 5] + .001) + .002];
gnoise[5] = Expand[g2[x, u, 5] * (c[x, u, 5] - .004) - .007];
Timing[floatPolynomialGCD[fnoise[5], gnoise[5], {10^(-2), 10^(-1)}]]
```

```
{8.85655, 0.997 + 2. x + 1. x^2 + 2. u[1] + 2. x u[1] + 1. u[1]^2 + 2. u[2]^2 +
  2. x u[2]^2 + 2. u[1] u[2]^2 + 1. u[2]^4 + 2. u[3]^3 + 2. x u[3]^3 + 2. u[1] u[3]^3 +
  2. u[2]^2 u[3]^3 + 1. u[3]^6 + 2. u[4]^4 + 2. x u[4]^4 + 2. u[1] u[4]^4 +
  2. u[2]^2 u[4]^4 + 2. u[3]^3 u[4]^4 + 1. u[4]^8 + 2. u[5]^5 + 2. x u[5]^5 +
  2. u[1] u[5]^5 + 2. u[2]^2 u[5]^5 + 2. u[3]^3 u[5]^5 + 2. u[4]^4 u[5]^5 + 1. u[5]^10}
```

6. SUMMARY

We have demonstrated how precision and accuracy ideas from numerical computation can be adapted to the setting of numerical Gröbner bases. While by no means flawless, we see from numerous examples that these approaches hold promise for handling overdetermined systems of algebraic equations. These computational methods also apply to other problems from hybrid symbolic−numeric computation, such as finding approximate polynomial GCDs.

While most examples covered seem to work efficiently and give reasonable results, it remains an open question as to how competitive these methods are in regard to speed and quality of results, as compared to other approaches. An advantage to Gröbner bases is that polynomial algebra is carried out in a sparse setting; many methods based on linear algebra require dense matrix manipulation. The examples presented offer evidence that, when working with input of modest degree, Gröbner bases methods are viable. That the coding is simple makes them all the more attractive.

An open area for further work is in determining, in some automated fashion (perhaps based on problem type), what are reasonable tolerances for a specific problem. A possible approach would be to set up an outer level optimization, wherein one strives to maximize a degree of a candidate GCD, or the (finite) number of solutions to an overdetermined system, and has for parameters these tolerances. This is another place where SVD−based matrix approaches have an advantage: a "natural" tolerance is generally revealed from the largest ratio in consecutuve singular values (possibly excepting cases where a jump is from a very small singular value to zero). At present all Gröbner basis methods need some prespecification of tolerance.

Another avenue for future work is to adapt methods from [21] to handle overdetermined systems at modest precision. Those methods for polynomial GCD, say, tend to be faster than what we indicate in this paper. But we have not yet succeeded in making them work for fuzzy systems where a GCD or factorization is only correct up to some modest tolerance.

It is also an open question whether symbolic "epsilon" powers can be used to improve the methods of this paper. The idea, roughly, is to replace coefficients that are deemed "small" (according to some precision tolerance, say) by suitable powers of a variable that is local in the term ordering sense (hence monomials having powers of this variable are smaller than any monomial not containing it, including constants). Variants of this idea are discussed in [17, 24, 29, 30, 32].

Based on experimentation and comparison of timings with other methods reported, we state a tentative conclusion. The methods of this paper are viable and effective when the problem at hand is unperturbed from an exactly solvable variant. They often give good results when the problem is overdetermined, provided the noise is modest relative to an exactly solvable nearby problem, and the scale of coefficients does not vary too much. In other situations it is not clear whether our methods can be adapted so readily.

7. CODE APPENDIX

Below is code used in computations in this paper.

```
setCoefficientPrecision[a_?NumberQ, prec_] :=
 If[Abs[a] < 10^(-prec), 0, SetPrecision[a, prec]]
setCoefficientPrecision[a_?NumberQ * b_?(! NumberQ[#] &), prec_] :=
 setCoefficientPrecision[a, prec] * b
setCoefficientPrecision[(a_Plus | a_Times | a_List), prec_] :=
 Map[setCoefficientPrecision[#, prec] &, a]
setCoefficientPrecision[a_, _] := a
```

```
floatPolynomialLCM[poly1_, poly2_, tol_] := Module[
  {vars, mat, v, cvars, newpolys, rels, gb, rul}, vars = Variables[{poly1, poly2}];
  mat = {{1, 1, 1}, {poly1, 0, 0}, {0, poly2, 0}};
  cvars = Array[v, 3];
  newpolys = mat.cvars;
  rels = Flatten[Union[Outer[Times, cvars, cvars]]];
  newpolys = Join[newpolys, rels];
  gb = GroebnerBasis[newpolys, Prepend[vars, Last[cvars]],
    Most[cvars], MonomialOrder → EliminationOrder , Tolerance→ tol,
    CoefficientDomain → InexactNumbers[Precision[newpolys]], Sort → True];
  rul = Map[(# → {}) &, rels];
  gb = Flatten[gb /. rul];
  First[gb] /. Last[cvars] → 1]

floatPolynomialGCD[p1_, p2_, tol_] :=
 Expand[PolynomialReduce[p1 * p2, floatPolynomialLCM[p1, p2, tol],
    CoefficientDomain → InexactNumbers][[1, 1]]]
```

Here are the longer examples we used.

`polysCassou =`

$$\{15 b^2 c d^2 + 6 b^2 c^3 + 21 b^2 c^2 d - 144 b c - 8 b c^2 e - 28 b c d e - 648 b d + b d^2 e + 9 b^2 d^3 - 120,$$
$$30 b^2 c^3 d - 32 c d e^2 - 720 b c d - 24 b c^3 e - 432 b c^2 + 576 c e - 576 d e + 16 b c d^2 e + 16 d^2 e^2 + 16 c^2 e^2 + 9 b^2 c^4 + 5184 + 39 b^2 c^2 d^2 + 18 b^2 c d^3 - 432 b d^2 + 24 b d^3 e - 16 b c^2 d e - 240 c,$$
$$216 b c d - 162 b d^2 - 81 b c^2 + 5184 + 1008 c e - 1008 d e + 15 b c^2 d e - 15 b c^3 e - 80 c d e^2 + 40 d^2 e^2 + 40 c^2 e^2, 261 + 4 b c d - 3 b d^2 - 4 b c^2 + 22 c e - 22 d e\};$$

`polysCaprasse =`

$$\{-2 x + 2 t x y - z + y^2 z, 2 + 4 x^2 - 10 t y + 4 t x^2 y - 10 y^2 + 2 t y^3 + 4 x z - x^3 z + 4 x y^2 z,$$
$$-x + t^2 x - 2 z + 2 t y z, 2 - 10 t^2 - 10 t y + 2 t^3 y + 4 x z + 4 t^2 x z + 4 z^2 + 4 t y z^2 - x z^3\};$$

`polysCaprasseModified =`

$$\left\{-2 x + 2 t x y - \frac{100\,001 z}{100\,000} + y^2 z,\right.$$
$$\frac{2\,000\,001}{1\,000\,000} + 4 x^2 - 10 t y + 4 t x^2 y - 10 y^2 + 2 t y^3 + 4 x z - x^3 z + 4 x y^2 z,$$
$$\left.-x + t^2 x - 2 z + 2 t y z, 2 - 10 t^2 - 10 t y + 2 t^3 y + 4 x z + 4 t^2 x z + 4 z^2 + 4 t y z^2 - x z^3\right\};$$

8. REFERENCES

Adams Loustaunau

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

Auzinger Stetter

[2] W. Auzinger and H. Stetter (1988). An elimination algorithm for the computation of all zeros of a system of multivariate polynomial equations. International Series of Numerical Mathematics **86**, R. P. Agarwal, Y. M. Chow, and S. J. Wilson editors, 11−31. Birkhäuser Verlag.

Becker Weispfenning Kredel

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

Bodrato Zanoni

[4] M. Bodrato and A. Zanoni (2004). Numerical Gröbner bases and syzygies: an interval approach. Proceedings of the 6th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 04), 77−89.

Bodrato Zanoni 2

[5] M. Bodrato and A. Zanoni (2006). Intervals, syzygies, numerical Gröbner bases: a mixed study approach. Proceedings of the 9th International Workshop on Computer Algebra in Scientific Computing (CASC 06), Springer LCNS **4194**, 64−76.

Buchberger

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

Corless

[7] R. Corless (1996). Editor's Corner: Gröbner bases and matrix eigenproblems. ACM SIGSAM Bulletin: Communications in Computer Algebra **30**(4),26−32.

Corless Gianni Trager Watt

[8] R. Corless, P. Gianni, B. Trager, and S. Watt (1995). The singular value decomposition for polynomial systems. Proceedings of the 1995 International Symposium on Symbolic and Algebraic Computation (ISSAC 95), 195−207. A. H. M. Levelt, editor. ACM Press.

Cox

[9] D. Cox (1998). Introduction to Gröbner bases. In *Proceedings of Symposia in Applied Mathematics* **53,** D. Cox and B. Sturmfels, editors. 1−24. ACM Press.

Cox Little OShea

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.

Gianni Mora

[11] P. Gianni and T. Mora (1988). Algebraic Solutions of systems of polynomial equations using Gröbner bases. In Applied Algebra, Algebraic Algorithms, and Error−Correcting Codes (AAECC 5). L. Hugeut, A. Poli, editors. Springer LCNS **356,** 247−257.

Hribernig Stetter

[12] V. Hribernig and H. Stetter (1997). Detection and validation of clusters of polynomial zeros. Journal of Symbolic Computation **24**:667−681.

Huang Stetter Wu Zhi

[13] Y. Huang, H. Stetter, W. Wu, and L. Zhi (2000). Pseudofactors of multivariate polynomials. Proceedings of the 2000 International Symposium on Symbolic and Algebraic Computation (ISSAC 00), 161−168. C. Traverso, editor. ACM Press.

Kaltofen Yang Zhi

[14] E. Kaltofen, Z. Yeng, and L. Zhi (2006). Approximate greatest common divisors of several polynomials with linearly constrained coefficients and singular polynomials. Proceedings of the 2006 International Symposium on Symbolic and Algebraic Computation (ISSAC 06), 169−176. J.−G. Dumas, editor. ACM Press. Examples available from:
http://www4.ncsu.edu/~kaltofen/software/manystln/

Karmarkar Lakshman

[15] N. Karmarkar and Y. Lakshman (1996). Approximate polynomial greatest common divisors and nearest singular polynomials. Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation (ISSAC 96), 35−39. Y. N. Lakshman, editor. ACM Press.

Keiper

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

Kondratyev

[17] A. Kondratyev (2003). Numerical Computation of Gröbner Bases. Doctoral dissertation, Johannes Kepler Universität, Linz. F. Winkler and H. Stetter, advisors.

KondratyevStetterWinkler

[18] A. Kondratyev, H. Stetter, and F. Winkler (2004). Numerical computation of Gröbner bases. in: 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.

Lichtblau 1

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

Lichtblau 2

[20] D. Lichtblau (2000). Solving finite algebraic systems using numeric Gröbner bases and eigenvalues. In Proceedings of the World Conference on Systemics, Cybernetics, and Informatics (SCI 2000), Volume 10, 555−560.
http://library.wolfram.com/infocenter/Conferences/7514/

Lichtblau 3

[21] D. Lichtblau (2008). Exact Computation Using Approximate Gröbner Bases. Manuscript.
http://library.wolfram.com/infocenter/Conferences/7537/

Reid Tang Zhi

[22] G. Reid, J. Tang, and L. Zhi (2003). A complete symbolic−numeric linear method for camera pose determination. Proceedings of the 2003 International Symposium on Symbolic and Algebraic Computation (ISSAC 03), 215−223. R. Sendra, editor. ACM Press.

Sanuki

[23] M. Sanuki (2007). Computing approximate GCD of multivariate polynomials. Proceedings of the 2005 International Workshop on Symbolic−numeric Computation (SNC 05), 55−68. Trends in Mathematics, Birkhäuser.

Sasaki Kako

[24] 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.

Sasaki Sasaki

[25] T. Sasaki and F. Sasaki (1997). Polynomial remainder sequence and approximate GCD. ACM SIGSAM Bulletin: Communications in Computer Algebra **31**(3):4−10.

Shirayanagi

[26] K. Shirayanagi (1993). An algorithm to compute floating point Groebner bases. Mathematical Computation with Maple V, Ideas and Applications 95−106. T. Lee, editor. Birkhauser Boston.

Shirayanagi 2

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

Sofroniou Spaletta

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

Stetter

[29] H. Stetter (1997). Stabilization of polynomial systems solving with Groebner bases. Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation (ISSAC 97), 117−124. W. Küchlin, editor. ACM Press.

Traverso Zanoni

[30] 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.

Verschelde

[31] J. Verschelde (2000−present). Web site of polynomial systems.
http://www.math.uic.edu/~jan/Demo/movestew.html

Zanoni

A. Zanoni (2003). Numerical Gröbner Bases. Doctoral dissertation, Università di Firenze, Italy. C. Traverso, advisor.

Zeng Dayton

[33] Z. Zeng and B. Dayton (2004). The approximate GCD of inexact polynomials. Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation (ISSAC 04), 320−327. J. Gutierrez, editor. ACM Press.