Hindawi Publishing Corporation
ISRN Computational Mathematics
Volume 2013, Article ID 352806, 12 pages
http://dx.doi.org/10.1155/2013/352806

Research Article

Approximate Gröbner Bases, Overdetermined Polynomial Systems, and Approximate GCDs

Daniel Lichtblau

Wolfram Research, Kernal Development Group, 100 Trade Center Drive, Champaign, IL 61820, USA

Correspondence should be addressed to Daniel Lichtblau; danl@wolfram.com

Received 6 November 2012; Accepted 29 November 2012

Academic Editors: R. Joan Arinyo, M. Ogihara, and Y. Peng

We discuss computation of Gröbner bases using approximate arithmetic for coefficients. We show how certain considerations of tolerance, corresponding roughly to absolute and relative error 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.

1. Introduction

Gröbner bases provide a means for solving a myriad of problems in computational algebra. In its original form, the whole arithmetic was carried through exactly, on rational numbers. This was necessary in order to know when combinations of polynomial coefficients cancel. In 1993 Shirayanagi [1] indicated a means to use approximate arithmetic and still handle this "zero recognition" problem. Since that time several approaches have appeared that use approximate arithmetic [2–10]. The advantages of 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 [1, 8]. 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 [5]). The handling of coefficients can thus be viewed as an extension of floating point arithmetic. Traverso and Zanoni [11] 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 that this requires either exact input or else an approximate system that is not overdetermined. Other approaches include extending the notion of Gröbner bases to allow head terms with small coefficients to become nonhead terms [9, 12–14]. Of these, [12] is notable for allowing the use of an almost unaltered Gröbner basis algorithm, with low rank variables added as needed during the course of the computation. This family of methods attempts to stabilize the set of head terms locally, that is, in a neighborhood of a set of coefficient values that give a "nongeneric" basis skeleton. They seem to work well for handling numeric conditioning problems that might arise from having polynomial leading terms with relatively small coefficients, but it is not clear whether they can be used in the case of an overdetermined system. References [2, 11] contain some discussions of the overdetermined case. Use of syzygies to determine vanishing of coefficients is described in [2, 3], with the latter indicating a possible algorithmic treatment. The local stabilization method is further studied in [15], in the special case where there is one "small" parameter. The generalized normal form approach of [16] uses a similar method to [9, 13, 14] for handling small leading coefficients, but has access to more possibilities since the monomial ordering need not be fixed from the start. These local stabilization methods seem to work best in the case of zero dimensional ideals with the same degree (number of solutions). This is of course a very important special case.

A typical situation in which one might desire to work with approximate coefficients is in solving polynomial systems of equations. A common method for such solving sets up an eigendecomposition problem; this is done either with resultants [17], Gröbner bases [18–21], or more general normal forms [16]. We remark that all but [16, 21] were developed in the setting of exact coefficient arithmetic (since 1998 the NSolve function of *Mathematica* has used approximate coefficient arithmetic in its Gröbner basis computations). As this usage of approximate arithmetic seems to be relatively less well known than its exact counterpart, we will begin with a brief discussion of the issues associated with the Gröbner basis part of such solvers. We go on to point out problems that appear when such systems are overdetermined. We then describe two modes of tolerancing and use them in tandem to address these problems.

This work provides a description and empirical study of methods that extend Gröbner bases to handle approximately consistent polynomial systems. We define these, informally, as inconsistent systems for which there exists a "small" perturbation of input coefficients that makes them consistent. This is, in effect, the opposite situation to that of an artificial structural discontinuity of a Gröbner basis as discussed in [9, 12–14]. Here we are really trying to create and utilize such a discontinuity rather than attempting to remove it. Instead of working with a concept of "nearby" solutions (though possibly not so near in cases of ill conditioning), we are now trying to find a nearby system that has *any* solutions, in a neighborhood where generically the solution set will be empty. A very important special case, discussed in several examples, is that of finding an approximate polynomial GCD to a given pair (or set) of polynomials. We recast as an overdetermined system using computational polynomial algebra [22] and proceed from there.

Several of the ideas we present have been developed independently in the cited literature. Our main contribution is to show how they can be made to work effectively in practice. We provide several nontrivial examples from the literature on numerical polynomial system solving and approximate GCD computation to illustrate the merit of this work.

2. Overview of Gröbner Bases

Gröbner bases are a tool used universally in computational commutative algebra. Among several excellent references we single out [19, 22, 23], because they cover various aspects of equation solving via Gröbner bases. We give a brief synopsis from [4].

One first defines a term order on the exponent vectors of power products of a polynomial (these are simply products of powers of variables, e.g., x^2yz^3). An important class of term orders is "lexicographic." In this class we have a predefined ordering amongst variables; say x is largest. Then monomials involving x are regarded as "larger" than terms not involving x, and higher powers in x are larger than lower powers. In effect monomials are placed in dictionary order, wherein letters correspond to the ordering between variables. Another important class of orders are based on total degree (higher degree corresponding to larger monomials), with ties broken by another order (such as lexicographic).

One next has a notion of reduction by rewriting a given polynomial. This can happen if the leading term of one divides the leading term of another; we can "reduce" that second one by subtracting an appropriate multiple of the first so that the resulting polynomial has a smaller leading monomial. Using any variant of the algorithm developed by Buchberger (as in the references above) one then rewrites the given set of polynomials to obtain new set called a Gröbner basis. A key step in this process, well explained in the literature, is to methodically generate new polynomials that cannot be reduced by the old ones.

This new set generates the same polynomial ideal and hence has the same solution set. If computed with respect to a lexicographic term ordering, one has in effect triangulated the system, in a form analogous to a row reduced system of linear equations. If computed by a degree-based term order one has in the basis at least one ideal member of smallest total degree. Each of these can be useful in various situations. We avail ourselves of both in this paper.

An issue with these bases is that often they are strenuous to compute. One cause can be a form of size increase wherein coefficients grow quite large compared to inputs. Use of approximate numbers of a specified maximal size helps to combat this. Moreover there are situations in which inputs are only known approximately to begin with; in such cases it makes little sense to do exact computations if that can be avoided. In [21, 24] we describe a mode of arithmetic that allows us to work in substantially the same way as with exact arithmetic, but without the burden of size growth. From here on we will assume that the underlying coefficient arithmetic is done with approximate numbers and proceed to describe variants that assist in certain important situations.

3. Approximate Gröbner Bases and 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 intermediate swell of integers or difficult computations with algebraic numbers.

The first scenario is in some sense quite nice insofar as theorems can be proven regarding the quality of result based on input precision. This is covered in some detail in [24]; where it is shown that computation of approximate Gröbner bases from exact input can be regarded as a solved problem (at least to the extent that any computation of such bases might be thus regarded). The reasons for this, in brief, are as follows. First suppose such a system is at a structural singularity in the sense of [9, 12–15]; that is, a system for which perturbations of the coefficients will change the structure of the basis. Then use of sufficiently high but finite precision arithmetic, with precision tracking, will uncover this fact; in practice the needed precision is typically modest [24]. If it is away from such a singularity then again a precision tracking coefficient

arithmetic suffices (and generally at less precision than might be needed at a singularity). This leaves the case where it is near a singularity, or possibly at a singularity to the precision of the input (i.e., the implied error in input coefficients includes a structurally singular system). This tends to be the hardest case in practice. By artificially raising input precision if it was finite, or using fairly high finite precision values for exact coefficients, we again can use precision tracking arithmetic. In principle the result could show either the generic form of basis or the structurally singular form, but with sufficiently high precision in the computation the former will happen in practice, since we are (almost never) actually at a structural singularity once we raise the input precision. In either case, though, it can be used to extract useful information, for example, roots to the system.

In Section 5 we will see a nontrivial pair of examples involving the Caprasse system and a perturbation thereof. Theoretical underpinning and practical considerations for this are discussed in [24, Section 2]. We observe that all three cases indicated above allow for the possibility that input is of finite and possibly of low precision. In such cases we can artificially raise the precision to perform the basis computation. Conceptually this means we are solving a nearby problem, indeed, one that is a representative from the set of all systems implied by the error in the coefficients. The only assumption is that zero really means zero; that is, we do not allow for implied interval inputs that contain zero. Moreover we make no claim that all sets in the family of systems will have close solution sets. That depends on the intrinsic conditioning of the problem and is outside the scope of this work.

While the case of exact input is certainly of interest, here we are primarily 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.

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 poses no issue, provided we do not work with an overdetermined system [21]. The essential idea [7, 25, 26] 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 implementations described in [1, 5–8], we require no careful bookkeeping; the arithmetic does this automatically.

A secondary issue is that, with this choice of arithmetic, precision gradually erodes over the course of a computation (as the arithmetic 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 [21]. 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 numerically huge and generally nonuseful solutions (consider the difference between solutions of $x^2 = 1$ and $(1/10^{40})\, x^5 + x^2 = 1$).

We end this section with a historical note. As mentioned earlier, the first reported implementation of numerical Gröbner bases (of which this author is aware) is due to Shirayanagi [1] from 1993. This article discusses a bookkeeping 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 [4]. We point out that, if input is overdetermined, it in effect requires either exact or at least "nice" (e.g., high precision) input. It will not handle input that is both known only to a few digits and also overdetermined. This case is taken up in the next sections.

4. Overdetermined Systems

We have just given a brief overview of how we can manage approximate coefficient arithmetic reliably when handling nonoverdetermined (and reasonably well conditioned) systems. Indeed this suffices for many practical sorts of computations. 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 [27], 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 tools to enable a decision that coefficients are "small enough" to discard.

Another source of overdetermined problems arises in trying to find "approximate" polynomial greatest common divisors [28–33]. 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 exact Gröbner basis methods to handle this approximate numeric setting.

5. 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 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 [34] 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 as applied to both absolute and relative accuracy. This is entirely informal; the purpose is simply to motivate our approach to zero recognition. By tolerance we typically have in mind a small threshold, below which we regard values as zero.

Recall that the key operations in Gröbner basis computations are forming of S-polynomials and reduction thereof [22, 23, 35, 36]. Our main concerns are twofold. We do not want to retain leading coefficients that, in an exact computation, would have vanished. Also we do not want to retain 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 in essence a relative error issue; one coefficient is notably smaller than most or all others. For this situation we will require a relative error tolerancing. In contrast, an entire polynomial might be regarded as a full cancellation in a situation where all coefficients are small, in an absolute sense, when compared with those of the normalized inputs that gave rise to them. This situation will be dealt with via absolute error tolerancing.

As noted above, we will utilize both relative and absolute tolerance values. Recall that in Gröbner basis computations all manipulations involving coefficient arithmetic arise from addition of pairs of polynomials. Prior to performing such an operation we compute the average magnitude of the coefficients in these polynomials; we will refer to this value as IPCA, for "input polynomial coefficient average". If, after addition, a resulting coefficient is less than the relative error tolerance times this IPCA, we regard it as zero and remove it. If in fact all coefficients are smaller than the absolute tolerance times the IPCA; then we regard the entire resulting polynomial as zero. In short, we employ the relative error mode to remove coefficients that are small relative to other coefficients, and we use the absolute error to justify removing an entire polynomial when all coefficients are small in absolute magnitude. We again note that this latter assumes some sort of normalization is in place for the polynomials that gave rise to the removed polynomial sum, since now comparison is not with other coefficients of the same polynomial, but rather with a pair of different polynomials whose sum generated the one under scrutiny.

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

The absolute tolerance is typically less prone to misuse (at least in the types of examples we will present). While some of the examples did in fact require trial-and-error selection of tolerances, many do not, and experience indicates one can often base a sensible setting on the precision of the input. Typical values for the sort of problems in the proceeding examples, with machine numbers for input, tend to be around 10^{-8} and 10^{-3} for the relative and absolute tolerances, respectively.

We mention that Kondratyev and coauthors [9, 13, 14] 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 ([12] also accomplishes this, though in a slightly different way). Also Traverso and Zanoni [11] 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 relative tolerance discussed above, and the first is similar to the absolute tolerance. Moreover, Sasaki and Kako [5, 6] used ideas similar to our absolute tolerancing for the detection of zero polynomials.

6. Exactly Determined Systems

The next few sections are organized around examples. All computations were with version 8.0.4 of *Mathematica* [37] on a 3 GHz desktop machine under the Ubuntu 11.04 Linux operating system. Auxiliary code is provided in an appendix, as are explicit inputs for most examples. When we use

tolerancing in specific examples of later sections, pairs of values denote relative and absolute tolerances, respectively. When one appears alone it is used as a relative tolerance.

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 provide a baseline in contrast with later computations. As they are exactly determined rather than overdetermined, in these initial examples we use no tolerancing.

Example 1. First we will show the Cassou-Noguès system [38]. We use high precision for the eigensystem phase of the solver, hence the nondefault WorkingPrecision specification (even using machine doubles for the eigensystem step we still get solutions with 8 or so correct digits, but this is not sufficient to obtain small residuals).

```
Timing[Length[solnsCassou = NSolve[polys
Cassou == 0, WorkingPrecision → 200]]]
{0.2, 16}
```

We check that the residuals are indeed small.

```
Max[Abs[polysCassou/.solnsCassou]]
0. × 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. As the computation is relatively fast, and the actual basis computation takes but a small fraction of the total time spent, it does not appear that precision loss is from the overall number of arithmetic operations. We suspect rather that this loss is due to the appearance of approximate clone polynomials (as defined in [5, 6]) at intermediate steps of the computation. In [7] the authors also refer to this phenomenon as accidental cancellation. In their simplest form clones arise when the same polynomial is used to reduce two others, and its tail becomes a large part, in the sense of coefficient norms, of the two reducta. The idea is that we now have two polynomials that are comparable to monomial multiples of the tail of one polynomial; hence they are close to one another.

Example 2. We next have an example that is considerably slower: the Caprasse system. It is troublesome because several roots have multiplicity, and moreover the multiplication (endomorphism) matrices utilized in the solver are derogatory (this circumstance is known to make trouble for the eigendecomposition method).

```
Timing[Length[solnsCaprasse = NSolve[polys
Caprasse, WorkingPrecision → 200]]]
{9.13, 56}
Max[Abs[polysCaprasse/.solnsCaprasse]]
8. × 10^-185
```

Example 3. Now 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,WorkingPrecision → 200]]]
{4.29, 56}
Max[Abs[polysCaprasseModified/.solnsCaprasse
Modified]]
2.59914108496 × 10^-138
```

While we do not show the Gröbner bases computed, we again remark that they have different structure. From either one we have recovered solutions that give small residuals and thus can be validated a posteriori (moreover they can be further refined using local root-finding methods).

Example 4. Here we tackle another common benchmark in the literature, the substantially larger Katsura-8 system.

```
Timing[solnsKatsura8 = NSolve[polysKatsura8]; ]
{222.58, Null}
Length[solnsKatsura8]
256
Max[Abs[polysKatsura8/.solnsKatsura8]]
2.59914108496 × 10^-138
```

7. Overdetermined and Ill Conditioned Systems

The previous section shows several standard examples that use numeric Gröbner basis computations. Those cases were exactly determined and, in some sense, well conditioned. They required no special tolerancing. Indeed, while the Caprasse system and the perturbed variant have exact or near multiplicity in roots, this can be handled directly by the numerical arithmetic as described in [24].

We will now show several examples that require tolerancing. There are two reasons this might be necessary. One is when the system is both overdetermined and only known to low approximation. In such cases a nontolerancing approach will result in (1), that is, a determination that the polynomials generate the entire ring. Another situation where tolerancing is important is when coefficients for our system lie approximately on a discriminant variety in the parameter space of all possible coefficients [39]. In such cases we may have spurious "solutions" that are approximations of infinities; where the coefficients and arithmetic exact they would not be present at all. Such examples arise in practical settings that have configurations of physical systems (robotics, molecules, etc.); hence it is of some use to remove the undesired solutions during the computation.

Example 5. We first work with a system that is in fact exactly determined, but nonetheless shows quite interesting behavior if not handled with tolerancing. It is a kinematics problem for a certain type of Stewart platform. The polynomial system comes from [38] where it is taken from earlier work by

Husty and Karger (Verschelde, private communication). As the input is long we will not show it, but simply describe the problem. It has nine polynomials in nine variables. The coefficients are machine double precision complex numbers.

With default settings, NSolve will find 80 solutions. This is twice the number given at [38], and moreover half of the solutions give large residuals and are themselves large. This makes one suspect they are erroneous. But raising the precision of the input and solving anew still give 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 algebraic constraints in order to correctly specify the type of platform in question. In making the coefficients machine doubles, 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 and thereby moved from a singular manifold to the generic case. We emphasize that this is not a situation where numeric difficulties arise from an artificially discontinuity in a Gröbner basis. To the contrary, we seek a Gröbner basis that is different in structure, rather than one coming from the generic case arising for nearby systems.

The tolerancing that repairs this is quite straightforward. We use 10^{-10} and 10^{-3} for relative and absolute tolerances, respectively. The overall result is that 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.

Around the time this paper was first submitted a similar but smaller example appeared on the internet StackExchange forum http://mathematica.stackexchange.com/questions/13947/nsolve-gives-additional-solutions-that-dont-satisfy-the-equations/. As it is fairly concise we show this example below and provide explicit input in the code appendix. The first line below shows how one tells the solver to pass on the tolerance specifications to the Gröbner basis code.

```
SetSystemOptions["NSolveOptions" →
{"Tolerance" → 10^(−10)}];
sol = NSolve[trigparamexprs, Variables
[trigparamexprs]]
```

We check explicitly that all residuals are zero (to close approximation).

```
trigparamexprs/.sol//Chop
{{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}}
```

When this is solved without tolerancing there are eight rather than two solutions. Six give residuals that indicate they are not of high quality. If one instead does the untoleranced computation after first raising precision of the inputs, then residuals become small. Nevertheless they remain notably larger than both solution precision and residuals from the "good" results, by a power-of-two factor roughly comparable to the number of bits of machine floats. This helps to explain how these artifact solutions arise. They come about from a system with coefficients that originate on the discriminant variety. They then become slightly removed due to round-off error in representing them as machine numbers. The distance they lie from the discriminant variety is manifested in the sizes of these solutions and in the discrepancy between solution precision and residual size.

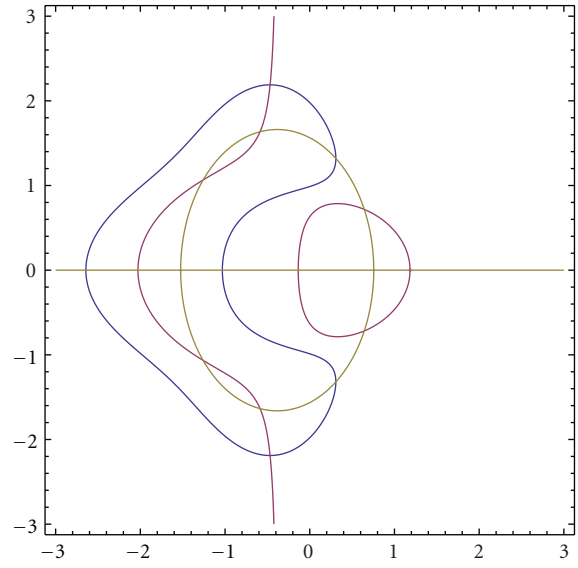

Figure 1

Example 6. Here is an example from [28]. 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 derivatives all (approximately) vanish.

```
poly = 4.0y^4 + 17.0x^2 y^2 + 13.07xy^2 − 19.572938y^2 +
4.0x^4 + 5.228x^3 − 18.29175x^2 − 5.228x + 15.29175;
polys = {poly, ∂x poly, ∂y poly};
SetSystemOptions["NSolveOptions" → {Tolerance
→ 10^(−3)}];
soln = NSolve[polys, {x, y}]
{{x → 1.18344, y → 0.}}
```

From the plot (Figure 1) of the zero set of the polynomial and derivatives, one sees that the derivatives cross near the solution value but the polynomial itself (the closed curve with leftward pointing arcs centered on the x-axis) does not come near to that solution.

```
d = 3;
cp = ContourPlot[Evaluate[polys], {x, −d, d},
{y, −d, d}, Contours → {0},
ContourShading → False, PlotPoints → 90]
```

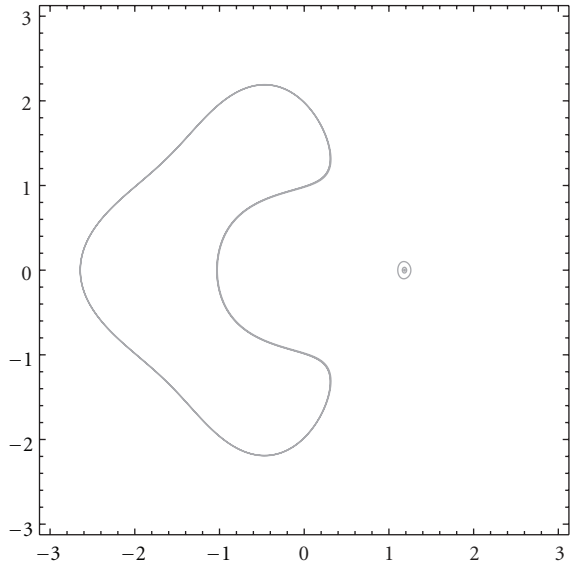

Figure 2

Figure 2 may help to explain why the approximate solution makes sense. We show not just the zero set but also level sets where the polynomial takes on certain small values. Now one sees that the polynomial is indeed small in a region near that approximate solution of (1.18344, 0).

```
d = 3;
cp = ContourPlot[poly, {x, −d, d},
{y, −d, d}, Contours → {0, .02, .2},
ContourShading → False,
PlotPoints → 90]
```

Here are the residuals of the polynomial and first derivatives, evaluated at the approximate zero of that system.

```
polys/.soln
{{−0.00236885, −0.0373837, 0.}}
```

Example 7. We now show an overdetermined camera pose problem from [27]. 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 small imaginary parts.

```
polys = {−4 + x1^2 − 1.49071x1x2 + x2^2, −8 + x1^2 − 0.4x1x3
+ x3^2, −4 + x1^2 − 0.894427x1x4 + x4^2, −4 + x2^2
− 1.49071x2x3 + x3^2, −8 + x2^2 − 0.666667x2x4
+ x4^2, −4 + x3^2 − 0.894427x3x4 + x4^2};
SetSystemOptions["NSolveOptions"
→ {Tolerance → {10^(−3), 0}}];
Chop[soln = Quiet[NSolve[polys, WorkingPrecision
→ 8]], 10^(−3)]
{{x1 → 2.23606, x2 → 2.99999, x3 → 2.23607,
x4 → 0.999999}, {x1 → 2.23606, x2 → 2.99999,
x3 → 2.23607, x4 → 0.999999}, {x1 → −2.23606,
x2 → −2.99999, x3 → −2.23607, x4 → −0.999999},
{x1 → −2.23606, x2 → −2.99999, x3 → −2.23607,
x4 → −0.999999}}
```

We check that the worst residual is not terribly large.

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

8. 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 [28–31] 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. At the time of this writing this method is used to support some *Mathematica* control theory code, in the situation where one needs to cancel approximate zeros with poles.

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 T. Sasaki and F. Sasaki in [33].

Example 8. Here is another example from [28]. With a relative error tolerance of two digits we recover a nontrivial approximate GCD.

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

Example 9. Now we show an example from [29], 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 = $x^9 - (5.833333 + 2.333333i)x^8 + (12.888889 + 11.7222222i)x^7 + (-13.416667 - 24.694444i)x^6 + (5.293210 + 28.703704i)x^5 + (2.389403 - 20.183642i)x^4 + (-3.790123 + 8.750857i)x^3 + (1.880630 - 2.247914i)x^2 + (-.452884 + .299535i)x + (.045217 - .013868i)$;

Chop[N[First[GroebnerBasis[setCoefficient Precision[{poly, D[poly, x]}, 40], x, Coefficient Domain → InexactNumbers, Tolerance → {$1/10^4$, $1/10^2$}]]]]

$(0.013033486039440238 + 0.24739461042947114i) - (0.3199779084192973 + 1.7517086950261698i)x + (1.9659824714996788 + 5.134785008354451i)x^2 - (5.221808113608173 + 7.66519441636273i)x^3 + (6.888250445595586 + 5.721144296886046i)x^4 - (4.33302767357858 + 1.6663679299044327i)x^5 + 1.x^6$

9. Multivariate Approximate GCD

Multivariate polynomial approximate GCDs algorithms are presented in [28–32, 40]. They tend to use matrix methods or polynomial sequences. We instead take an elimination ideal method from [35], using approximate Gröbner bases as the main computational engine to get the (approximate) LCM. We follow with generalized division to extract the GCD. The elimination ideal method gives an overdetermined system of equations and hence fits nicely into the focus of this paper.

We show some examples below. Note that we make no effort to locally improve the result, for example, by Newton's method.

Example 10. This is example exF07 from [30]. 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]], {$1/10^8$, $1/10^4$}]]

$\{0.816051, 2.00000000003 + 6.x + 10.x^2 + 8.x^3 - 2.x^4 + 7.64022323556 \times 10^{-11} y + 7.99999999999xy + 2.54508805434 \times 10^{-13} x^2 y - 8.x^3 y - 7.99999999993y^2 - 8.00000000002xy^2 - 10.x^2 y^2 + 8.00000000001y^3 - 3.99999999999xy^3 - 5.99999999999y^4 + 5.99999999998z - 10.xz - 10.x^2 z + 2.09215361085 \times 10^{-14} x^3 z + 10.yz + 8.xyz + 4.x^2 yz - 3.99999999988y^2 z + 1.99999999999xy^2 z - 8.00000000004y^3 z + 5.99999999999z^2 - 4.xz^2 + 2.x^2 z^2 - 10.yz^2 - 10.xyz^2 - 9.99999999997y^2 z^2 - 2.z^3 + 4.xz^3 - 6.yz^3 - 2.z^4\}$

Example 11. This is an example from [1].

c[x_, u_, n_] := $(x + \sum_j^n u[j]^j + 1)^2$

f2[x_, u_, n_] := $(x^2 - \sum_j^n u[j] - .5)^2$

g2[x_, u_, n_] := $(x^2 + \sum_j^n u[j] + .5)^2$

We create a pair of polynomials with prescribed 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], {$1/10^6$, $1/10^2$}]]

$\{4.96, 1. + 2.x + 1.x^2 + 2.u[1] + 2.xu[1] + 1.u[1]^2 + 2.u[2]^2 + 2.xu[2]^2 + 2.u[1]u[2]^2 + 1.u[2]^4 + 2.u[3]^3 + 2.xu[3]^3 + 2.u[1]u[3]^3 + 2.u[2]^2 u[3]^3 + 1.u[3]^6 + 2.u[4]^4 + 2.xu[4]^4 + 2.u[1]u[4]^4 + 2.u[2]^2 u[4]^4 + 2.u[3]^3 u[4]^4 + 1.u[4]^8 + 2.u[5]^5 + 2.xu[5]^5 + 2.u[1]u[5]^5 + 2.u[2]^2 u[5]^5 + 2.u[3]^3 u[5]^5 + 2.u[4]^4 u[5]^5 + 1.u[5]^{10}\}$

Example 12. Here we show 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[approxgcd = floatPolynomialGCD[fnoise [5], gnoise[5], {10^(-2), 10^(-1)}]]

$\{4.93, 0.997 + 2.x + 1.x^2 + 2.u[1] + 2.xu[1] + 1.u[1]^2 + 2.u[2]^2 + 2.xu[2]^2 + 2.u[1]u[2]^2 + 1.u[2]^4 + 2.u[3]^3 + 2.xu[3]^3 + 2.u[1]u[3]^3 + 2.u[2]^2 u[3]^3 + 1.u[3]^6 + 2.u[4]^4 + 2.xu[4]^4 + 2.u[1]u[4]^4 + 2.u[2]^2 u[4]^4 + 2.u[3]^3 u[4]^4 + 1.u[4]^8 + 2.u[5]^5 + 2.xu[5]^5 + 2.u[1]u[5]^5 + 2.u[2]^2 u[5]^5 + 2.u[3]^3 u[5]^5 + 2.u[4]^4 u[5]^5 + 1.u[5]^{10}\}$

A natural question, which we now consider, is what this result might represent. We will show that it is an exact (up to the numerical precision in use) GCD for a nearby set of inputs. For purposes of assessing proximity we will use the customary 1-norm of a given polynomial, defined as the sum of the absolute values of its coefficients. To gauge the quality of our approximated GCD we will try to find a nearby set of inputs that has this GCD exactly. This is straightforward to do using the generalized division with remainder (also called polynomial reduction) of Buchberger's algorithm [22, 23, 35, 36]. We will reduce each input polynomial by the candidate GCD and check the size of the resulting polynomial relative to the input. We then subtract this from the input. That gives our "nearby" input that is exactly divisible by the GCD. We illustrate this with the above example.

We compute the generalized remainders.

quot1 = Expand[PolynomialReduce[fnoise[5], approxgcd][[2]]];

quot2 = Expand[PolynomialReduce[gnoise[5], approxgcd][[2]]];

Here are the norms of the inputs and also of these remainders.

Map[polynomialNorm, {fnoise[5], gnoise[5]}]

{2033.79, 2070.07}

Map[polynomialNorm, {quot1, quot2}]

{20.0604, 5.27909}

So the perturbation polynomials, quot1 and quot2, are indeed small compared to the input polynomials.

Now we check that the perturbations formed by subtracting these from their respective inputs; each is divisible by the GCD polynomial. This is shown by the fact that the divisions give only constant terms on the order of the computational error one expects from the precision we used.

Expand[PolynomialReduce[fnoise[5] − quot1, approxgcd][[2]]]

$-1.22133074543 \times 10^{-17}$

Expand[PolynomialReduce[gnoise[5] − quot2, approxgcd][[2]]]

$-6.10687436863 \times 10^{-18}$

One can moreover check that the approximate GCD computed from these perturbed inputs, using much higher tolerances, agrees (up to constant multiple) with the GCD found in the noisy input problem.

We remark that one can use optimization methods such as iterative refinement in order to attempt to improve the perturbations by making them smaller in norm. In this setting one might also allow for perturbing the GCD, so long as it remains an exact GCD of the (newly) perturbed inputs. Such ideas appear in [28–32, 40] and references therein. One might use the perturbations as computed above as starting points for optimizations. We also remark that we have made no attempt to constrain the coefficient values of the perturbations. Hence the nearby polynomials could have (relatively small) terms that did not appear in the original inputs.

10. Summary and Future Directions

We have demonstrated how relative and absolute error 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 reasonable tolerances for a specific problem are. 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 consecutive 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 [24] to handle overdetermined systems at modest precision. Those methods for polynomial GCD are often significantly faster than what we indicate in this paper. They use approximate arithmetic but require (nearly) exact input. To date we have not succeeded in making them work with tolerancing, so they do not apply to fuzzy systems where a GCD or factorization is only correct up to some modest tolerance. It would thus be quite useful to get these methods to work well with tolerancing.

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 relative error 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 [5, 9–11, 13].

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.

11. 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_?(!Number
Q[#]&), 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]]]
polynomialNorm[poly_Plus]:= Plus@@Map
[polynomialNorm, poly]
polynomialNorm[poly_Times]:= Abs[poly/.
Thread[Variables[poly] → 1]]
polynomialNorm[poly_Power] := 1
polynomialNorm[poly_?NumericQ] := Abs[poly]
```

Here are inputs for several of the examples.

```
polysCassou = {6b^4c^3 + 21b^4c^2d + 15b^4cd^2 + 9b^4d^3 −
8b^2c^2e − 28b^2cde − 144b^2c + 36b^2d^2e − 648b^2d −
120, 9b^4c^4 + 30b^4c^3d + 39b^4c^2d^2 + 18b^4cd^3 − 24b^2c^3e −
16b^2c^2de − 432b^2c^2 + 16b^2cd^2e − 720b^2cd + 24b^2d^3e −
432b^2d^2 + 16c^2e^2 − 32cde^2 + 576ce − 240c + 16d^2e^2 −
576de + 5184, −(15b^2c^3e) + 15b^2c^2de − 81b^2c^2 +
216b^2cd − 162b^2d^2 + 40c^2e^2 − 80cde^2 + 1008ce + 40d^2e^2 −
1008de + 5184, −(4b^2c^2) + 4b^2cd − 3b^2d^2 + 22ce − 22de +
261};
polysCaprasse = {−2x + 2txy − z + y^2z, 2 + 4x^2 − 10ty +
4tx^2y − 10y^2 + 2ty^3 + 4xz − x^3z + 4xy^2z − x + t^2x −
2z + 2tyz, 2 − 10t^2 − 10ty + 2t^3y + 4xz + 4t^2xz + 4z^2 +
4tyz^2 − xz^3};
polysCaprasseModified = {−2x + 2txy
−100001z/100000 + y^2z, 2000001/1000000 + 4x^2 −
10ty + 4tx^2y − 10y^2 + 2ty^3 + 4xz − x^3z + 4xy^2z, −x +
t^2x − 2z + 2tyz, 2 − 10t^2 − 10ty + 2t^3y + 4xz + 4t^2xz +
4z^2 + 4tyz^2 − xz^3};
polysKatsura8 = {−x1 + x1^2 + 2x2^2 + 2x3^2 + 2x4^2 +
2x5^2 + 2x6^2 + 2x7^2 + 2x8^2 + 2x9^2,
−x2 + 2x1x2 + 2x2x3 + 2x3x4 + 2x4x5 + 2x5x6 + 2x6x7 +
2x7x8 + 2x8x9,
x2^2 − x3 + 2x1x3 + 2x2x4 + 2x3x5 + 2x4x6 + 2x5x7 +
2x6x8 + 2x7x9,
2x2x3 − x4 + 2x1x4 + 2x2x5 + 2x3x6 + 2x4x7 + 2x5x8 +
2x6x9,
x3^2 + 2x2x4 − x5 + 2x1x5 + 2x2x6 + 2x3 − x7 + 2x4x8 +
2x5x9, 2x3x4 + 2x2x5 − x6 + 2x1x6 + 2x2x7 + 2x3x8 +
2x4x9,
x4^2 + 2x3x5 + 2x2x6 − x7 + 2x1x7 + 2x2x8 +
2x3x9, 2x4x5 + 2x3x6 + 2x2x7 − x8 + 2x1x8 + 2x2x9,
−1 + x1 + 2x2 + 2x3 + 2x4 + 2x5 + 2x6 + 2x7 + 2x8 + 2x9};
```

Here are equations from http://mathematica.stackexchange.com/questions/13947/nsolve-gives-additional-solutions-that-dont-satisfy-the-equations/.

```
trigparamexprs = {c[1]^2+s[1]^2−1, c[2]^2+s[2]^2−
1, c[3]^ 2 + s[3]^2 − 1,
y1[1](−(0.564692c[1]) + c[2] − 0.825302s[1]) − 1,
y1[2](0.503363c[2] + c[3] − 0.864075s[2]) − 1,
y1[3](c[1] − 0.478806c[3] + 0.877921s[3]) − 1,
y1[1](−(0.825302c[1]) + 0.564692s[1] + s[2])
−y2[1](−(0.877921c[3]) + s[1] − 0.478806s[3]),
y2[1](c[1] − 0.478806c[3] + 0.877921s[3]) − 1,
y2[2](−(0.564692c[1]) + c[2] − 0.825302s[1]) − 1,
y1[2](−(0.864075c[2]) − 0.503363s[2] + s[3])
−y2[2](−(0.825302c[1]) + 0.564692s[1] + s[2]),
y2[3](0.503363c[2] + c[3] − 0.864075s[2]) − 1,
y1[3](0.877921c[3] − s[1] + 0.478806s[3])
−y2[3](−(0.864075c[2]) − 0.503363s[2] + s[3])};
vars = Variables[trigparamexprs];
```

We can run the solver as follows.

```
SetSystemOptions["NSolveOptions"
→ {"Tolerance" → 10^(−10)}];
sol = NSolve[trigparamexprs, vars];
```

If instead we run at default settings we get some "spurious" solutions, as explained earlier in the paper. We show here that both solution size and residual size can be quite large.

```
SetSystemOptions["NSolveOptions"
→ {"Tolerance" → 0}];
sol = NSolve[trigparamexprs, vars];
Map[Max[Abs[#]]&, vars/.sol]
Map[Max[Abs[#]]&, trigparamexprs/.sol]
{2.79314 × 10^16, 2.79314 × 10^16, 1.5761 × 10^16, 1.5761 ×
10^16, 1.5096 × 10^16, 1.5096 × 10^16, 17.8457, 17.8457}
{1., 1., 1., 1., 1., 1., 9.99200722163 × 10^−16,
9.99200722163 × 10^−16}
```

We can get a smaller residual by solving a nearby exact system, as below. One may notice that the residuals of the unwanted solutions (all but the last two) are around machine precision larger in scale than the residuals of the two good solutions at the end.

```
exactexprs = Rationalize[trigparamexprs, 0];
sol50 = NSolve[exactexprs == 0, vars,
WorkingPrecision → 50];
```

Of course if we plug the solutions to the exact problem into the original one, the residuals from the bad solutions are again quite large. This goes to illustrate the extreme ill conditioning that gave rise to the unwanted solutions.

Map[Max[Log[Abs[#]+1]]&, trigparamexprs/.sol50]

{0.936292, 0.936292, 0.628562, 0.628562, 0.98432,

0.98432, $8.881784197 \times 10^{-16}$,

$8.881784197 \times 10^{-16}$}

This following code is for the camera pose problem.

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[$x[\#1]^2 + x[\#2]^2 - \#3x[\#1]x[\#2] - \#4$&, Transpose[coords]]

Acknowledgments

I thank the reviewers of this and earlier drafts for providing several helpful comments, suggestions, and references. It is my hope that the attempt to address their remarks has improved this paper.

References

[1] K. Shirayanagi, "An algorithm to compute floating point Groebner bases," in *Mathematical Computation With Maple V, Ideas and Applications*, T. Lee, Ed., pp. 95–106, Birkhauser, Boston, Mass, USA, 1993.

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

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

[4] D. Lichtblau, "Gröbner bases in Mathematica 3.0," *The Mathematica Journal*, vol. 6, no. 4, pp. 81–88, 1996.

[5] T. Sasaki and F. Kako, "Computing floating-point Gröbner bases stably," in *Proceedings of the 2007 International Workshop on Symbolic-numeric Computation (SNC '07)*, pp. 180–189, ACM Press, July 2007.

[6] T. Sasaki and F. Kako, "Floating-point Gröbner basis computation with ill-conditionedness estimation," in *Computer Mathematics*, D. Kapur, Ed., vol. 5081 of *Lecture Notes In Artificial Intelligence*, pp. 278–292, Springer-Verlag, Berlin, Germany, 2008.

[7] T. Sasaki and F. Kako, "Term cancellations in computing floatingpoint Gröbner bases," in *Proceedings of the 12th international conference on Computer algebra in scientific computing (CASC '10)*, V. P. Gerdt, W. Koepf, E. W. Mayr, and E. V. Vorozhtsov, Eds., pp. 220–231, Springer-Verlag, Berlin, Germany, 2010.

[8] K. Shirayanagi, "Floating point Gröbner bases," *Mathematics and Computers in Simulation*, vol. 42, no. 4-6, pp. 509–528, 1996.

[9] H. J. Stetter, "Stabilization of polynomial systems solving with Groebner bases," in *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation (ISSAC '97)*, W. Küchlin, Ed., pp. 117–124, ACM Press, July 1997.

[10] A. Zanoni, *Numerical Gröbner bases [Ph.D. thesis]*, Università di Firenze, Florence, Italy, C. Traverso, advisor, 2003.

[11] C. Traverso and A. Zanoni, "Numerical stability and stabilization of Groebner basis computation," in *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation (ISSAC '02)*, T. Mora, Ed., pp. 262–269, ACM Press, July 2002.

[12] J.-C. Faugere and Y. Liang, "Pivoting in extended rings for computing approximate Gröbner bases," *Mathematics in Computer Science*, vol. 5, no. 2, pp. 179–194, 2011.

[13] A. Kondratyev, *Numerical computation of Gröbner bases [Ph.D. thesis]*, Johannes Kepler Universität, Linz, Austria, 2003.

[14] A. Kondratyev, H. Stetter, and F. Winkler, "Numerical computation of Gröbner bases," in *Proceedings of the 7th Workshop on Computer Algebra in Scientific Computing (CASC '04)*, V. G. Ghanza, E. W. Mayr, and E. V. Vorozhtov, Eds., pp. 295–306, Technical University Munich, St. Petersburg, Russia, 2004.

[15] J.-C. Faugere and Y. Liang, "Artificial discontinuities of single-parametric Gröbner bases," *Journal of Symbolic Computation*, vol. 46, no. 4, pp. 459–466, 2011.

[16] B. Mourrain and P. Trebuchet, "Generalized normal forms and polynomial system solving," in *Proceedings of the 2005 International Symposium on Symbolic and Algebraic Computation (ISSAC '05)*, pp. 253–260, July 2005.

[17] W. Auzinger and H. Stetter, "An elimination algorithm for the computation of all zeros of a system of multivariate polynomial equations," R. P. Agarwal, Y. M. Chow, and S. J. Wilson, Eds., International Series of Numerical Mathematics 86, pp. 11–31, Birkhäuser Verlag, Berlin, Germany, 1988.

[18] R. Corless, "Editor's corner: Gröbner bases and matrix eigenproblems," *Communications in Computer Algebra*, vol. 30, no. 4, pp. 26–32, 1996.

[19] D. Cox, "Introduction to Gröbner bases," in *Proceedings of Symposia in Applied Mathematics*, D. Cox and B. Sturmfels, Eds., vol. 53, pp. 1–24, ACM Press, 1998.

[20] P. Gianni and T. Mora, "Algebraic solutions of systems of polynomial equations using Gröbner baseseditors," in *Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes (AAECC 5)*, L. Hugeut and A. Poli, Eds., vol. LCNS 356, pp. 247–257, Springer, Berlin, Germany, 1988.

[21] D. Lichtblau, "Solving finite algebraic systems using numeric Gröbner bases and eigenvalues," in *Proceedings of the World Conference on Systemics, Cybernetics, and Informatics (SCI '00)*, vol. 10, pp. 555–560, 2000.

[22] D. Cox, J. Little, and D. O'Shea, *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Computer Algebra. Undergraduate Texts in Mathematics*, Springer-Verlag, Berlin, Germany, 1992.

[23] B. Buchberger, "Gröbner bases: an algorithmic method in polynomial ideal theory," in *Multidimensional Systems Theory*, N. K. Bose, Ed., chapter 6, D. Reidel Publishing Company, Dordrecht, The Netherlands, 1985.

[24] D. Lichtblau, "Polynomial GCD and factorization via approximate Gröbner bases," in *Proceedings of the 12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC '10)*, pp. 29–36, 2010.

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

[26] M. Sofroniou and G. Spaletta, "Precise numerical computation," *Journal of Logic and Algebraic Programming*, vol. 64, no. 1, pp. 113–134, 2005.

[27] G. Reid, J. Tang, and L. Zhi, "A complete symbolic-numeric linear method for camera pose determination," in *Proceedings of the 2003 International Symposium on Symbolic and Algebraic Computation (ISSAC '03)*, R. Sendra, Ed., pp. 215–223, ACM Press, August 2003.

[28] R. M. Corless, P. M. Gianni, B. M. Trager, and S. M. Watt, "Singular value decomposition for polynomial systems," in *Proceedings of the 1995 International Symposium on Symbolic and Algebraic Computation (ISSAC '95)*, H. M. Levelt, Ed., pp. 195–207, ACM Press, July 1995.

[29] V. Hribernig and H. J. Stetter, "Detection and validation of clusters of polynomial zeros," *Journal of Symbolic Computation*, vol. 24, no. 6, pp. 667–682, 1997.

[30] E. Kaltofen, Z. Yang, and L. Zhi, "Approximate greatest common divisors of several polynomials with linearly constrained coefficients and singular polynomials," in *Proceedings of the 2006 International Symposium on Symbolic and Algebraic Computation (ISSAC '06)*, J.-G. Dumas, Ed., pp. 169–176, ACM Press, July 2006.

[31] N. Karmarkar and Y. N. Lakshman, "Approximate polynomial greatest common divisors and nearest singular polynomials," in *Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation (ISSAC '96)*, Y. N. Lakshman, Ed., pp. 35–39, ACM Press, July 1996.

[32] M. Sanuki, "Computing approximate GCD of multivariate polynomials," in *Proceedings of the 2005 International Workshop on Symbolic-numeric Computation (SNC '05)*, Trends in Mathematics, pp. 55–68, Birkhauser, 2007.

[33] T. Sasaki and F. Sasaki, "Polynomial remainder sequence and approximate GCD," *Communications in Computer Algebra*, vol. 31, no. 3, pp. 4–10, 1997.

[34] Y. Huang, W. Wu, H. J. Sletter, and L. Zhi, "Pseudofactors of multivariate polynomials," in *Proceedings of the 2000 International Symposium on Symbolic and Algebraic Computation (ISSAC '00)*, C. Traverso, Ed., pp. 161–168, ACM Press, August 2000.

[35] W. Adams and P. Loustaunau, *An Introduction To Gröbner Bases. Graduate Studies in Mathematics*, vol. 3, American Mathematical Society, Providence, RI, USA, 1994.

[36] T. Becker, W. Weispfenning, and H. Kredel, *Gröbner Bases: A Computational Approach To Computer Algebra. Graduate Texts in Mathematics*, vol. 141, Springer-Verlag, Berlin, Germany, 1993.

[37] Wolfram Research, Inc. Champaign, Illinois. Mathematica 8, http://www.wolfram.com/, 2010.

[38] J. Verschelde, Web site of polynomial systems, http://homepages.math.uic.edu/~jan/Demo/cassou.html, http://www.math.uic.edu/~jan/Demo/movestew.html, 2000.

[39] D. Lazard and F. Rouillier, "Solving parametric polynomial systems," *Journal of Symbolic Computation*, vol. 42, no. 6, pp. 636–667, 2007.

[40] Z. Zeng and B. Dayton, "The approximate GCD of inexact polynomials," in *Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation (ISSAC '04)*, J. Gutierrez, Ed., pp. 320–327, ACM Press, 2004.