Approximate Gröbner Bases and Overdetermined Algebraic Systems

Daniel Lichtblau
danl@wolfram.com
Wolfram Research, Inc.
100 Trade Center Dr.
Champaign IL 61820

ACA 2008
Linz Austria
Session: Symbolic and Numeric Computation

ABSTRACT

We discuss computation of Gröbner bases using approximate arithmetic for coefficients. We show how certain considerations of tolerance, corre−sponding roughly to accuracy and precision from numeric computation, allow us to obtain good approximate solutions to problems that are overdeter−mined. 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.

Numeric Gröbner Bases

- Gröbner bases in original form: only supports exact computation.
  - Reason: Must know when sums of coefficients cancel.
- Advantages to approximate arithmetic:
  - No coefficient swell.
  - Can work with approximate input e.g. from scientific measurement.
- Numeric Gröbner Bases appear around 15 years ago.
  - 1993: Shirayanagi indicates way to use approximate arithmetic and handle "zero recognition" problem.
  - Since 1993: Several approaches have appeared that use approximate arithmetic.
- Typical applications:
  - Solve a system of polynomial equations (`NSolve` in *Mathematica* relies on numeric GBs).
  - Compute approximate greatest common divisor (this can be cast as an overdetermined system of equations).

Issues

- Low input precision.
  - For exactly determined systems one can artificially raise precision (solving a nearby system).
- Overdetermined systems.
  - Raising precision does not suffice (though might still be needed).
- False zero recognition (exact case would give small nonzero value).
  - Rare in practice.
- Systems is overdetermined AND has low precision (i.e. has only "pseudozeros").
  - Camera pose.
  - Approximate gcds.

Of these issues, the one we focus on is the last. Overdetermined systems that are only approximately solvable cause difficulty because we need a way to decide what to regard as zero, and full cancellation no longer (alone) suffices.

Overdetermined Systems

How to avoid empty solution set?

We need tools to decide when coefficients are "small enough" to be regarded as zero.

Idea: Coarsen our classification of what to consider as full cancellation.

Various Computation Methods

- Use bookkeeping to keep track of precision. Regard a sum as zero if all significant digits have cancelled. Reduce precision of sums and products based on first order estimates of error propagation. (Shirayanagi, L.)

- Work in two different fixed precisions. Use a threshold (of relative size, say) for deciding a sum is zero. (Traverso and Zanoni)

- If a system has exact input, use a finite modulus (Gröbner trace) as an oracle to determine when terms vanish. Can be used in conjunction with other methods above. (Traverso and Zanoni)
    - They 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 our precision tolerance, and the first is similar to the accuracy tolerance (to be discussed later).

- Extend Gröbner basis definition and algorithm to distinguish and handle "small" head terms. (Stetter and Kondratyev)
    - 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.

- Use tolerances (Sasaki and Kako, Traverso and Zanoni, L.)
    - Sasaki and Kako used ideas similar to our accuracy tolerancing for the detection of zero polynomials.

Precision and Accuracy, Informally

- Precision: Relative concept.
  - One considers ratio of estimated error to value.
- Accuracy: Absolute magnitude of error.

Arithmetic Considerations for Numeric Gröbner Bases

Recall that the key operations in Gröbner basis computations are forming of S−polynomials and reduction thereof.

- Our main concerns
  - Do not want to retain leading coefficients that, in an exact computation, would have vanished.
  - Do not want polynomials that should have vanished in their entirety.
  - Conclusion: Must employ some tactics for recognizing cancellation.
- Observations
  - Loss of a leading coefficient is analogous to a precision issue: one coefficient is notably smaller than most or all others.
  - An entire polynomial of "small" coefficients is analogous to an accuracy issue: a pair of two polynomials, summed, approximately cancels.

Tolerancing the Arithmetic

We will have a "precision" tolerance and an "accuracy" tolerance.

All manipulations involve addition of pairs of polynomials.

Find average magnitude of coefficients in these polynomials (IPCA, for "input polynomial coefficient average").

If a coefficient in the sum is less than the precision tolerance times IPCA, call it zero.

If all coefficients are smaller than the accuracy tolerance times IPCA then call the entire polynomial zero.

- Upshot:
  - *Precision* mode removes coefficients that are small relative to other coefficients. Typical value: 10^{-8}
  - *Accuracy* mode removes an entire polynomial when all coefficients are small in absolute magnitude. Typical value: 10^{-3}

A SImple Tolerancing Example

Let $p = x^2 + 2.2\,x\,y + 3.7\,y$, $q = 2.\,x^2 + 4.2\,x\,y + 7.5\,y$, and $r = x^2 + 2.23\,x\,y - y^2$.

- Notice that the coefficient averages are approximately 2.3, 4.6, and 1.4 respectively.

Is $2\,p - q$ equal to zero?

- The operation gives $0.2\,x\,y - 0.1\,y$.
- If our accuracy tolerance is, say, $1/10$, then we would regard this as zero becaue all coefficients arfe less than the threshhold times the IPCA of the two polynomials used in the operation (obeserve that of course the IPCAs of $2\,p$ and q must be similar, so it does not really matter which we use).

Now look at the leading coefficient of $p - r$, using lexicographic term order with $x \succ y$.

- The difference is $-0.03\,x\,y + y^2 + 3.7\,y$.
- The IPCA of the result is about 2.4 and the leading coefficient is 0.03.
- If our precision tolerance is, say,10^{-5}, then we would retain it.
- But if it is, say, 10^{-2}, then that coefficient would be regarded as zero (since $.03 < 2.4 \times 10^{-2}$).

Example 1: Cassou−Noguès Numeric System

It is exactly determined, hence provides a sort of "baseline" behavior. Requires high precision for the eigensystem phase of the solver.

```
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};
```

```
Timing[Length[solnsCassou = NSolve[polysCassou == 0, WorkingPrecision → 200]]]
```

```
{0.236015, 10}
```

We check that the residuals are indeed small.

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

```
0. × 10^-145
```

◄ | ►

Example 2: Overdetermined Camera Pose System

An overdetermined camera pose problem from the literature.

We need to raise precision artificially so that the `GroebnerBasis` step can run to completion.

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), 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}}
```

The worst residual is not terribly large.

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

```
0.000064876
```

GCDs via Gröbner bases

For univariate polynomials, can extract a GCD via simple Gröbner basis computation.

- This is in effect a form of polynomial remainder sequence.
- It can be regarded as an overdetermined system (two polynomials, one variable).

Multivariate polynomial LCM of (f, g) can be computed as an ideal intersection: eliminate t from $\{t\, f, (1 - t)\, g\}$.

- Follow with generalized division to extract the GCD.

Example 3: A Univariate Polynomial 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;
```

We use a Gröbner basis computation as a remainder sequence approach to obtain the gcd. We artificially raise precision, but lower tolerance for zero determination. Raising of precision is necessitated as a detail of the implementation, and might not be required by other programs.

```
First[N[GroebnerBasis[setCoefficientPrecision[{p1, p2}, 50],
    x, CoefficientDomain → InexactNumbers , Tolerance→ 1/10^2]]]
```

```
-2.00015 + 3.00024 x^5 + 1. x^6
```

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
```

Multivariate Polynomial GCD and Other Code

Here is code used to raise precision artificially.

Here is code to obtain polynomial gcds. We first find the LCM, using a simple module Gröbner basis elimination method from the textbook literature.

```
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]]]
```

◄ | ►

Example 4: A Multivariate Polynomial GCD

This is example exF07 (referred to as `cleanF7_list)` at
http://www4.ncsu.edu/~kaltofen/software/manystln/

The input, if rationalized, has a nontrivial (exact) GCD (so this is not such a hard problem).

[Honking big input below: open at your own risk.]

```
Timing[fgcd = floatPolynomialGCD[exF07polys[[1]], exF07polys[[2]], {10^(-8), 10^(-4)}]]
```

$$\{0.816051,\ 2. + 6.\,x + 10.\,x^2 + 8.\,x^3 - 2.\,x^4 + 7.64022 \times 10^{-11}\,y + 8.\,x\,y + 2.54509 \times 10^{-13}\,x^2\,y - 8.\,x^3\,y - 8.\,y^2 - 8.\,x\,y^2 - 10.\,x^2\,y^2 + 8.\,y^3 - 4.\,x\,y^3 - 6.\,y^4 + 6.\,z - 10.\,x\,z - 10.\,x^2\,z + 2.09215 \times 10^{-14}\,x^3\,z + 10.\,y\,z + 8.\,x\,y\,z + 4.\,x^2\,y\,z - 4.\,y^2\,z + 2.\,x\,y^2\,z - 8.\,y^3\,z + 6.\,z^2 - 4.\,x\,z^2 + 2.\,x^2\,z^2 - 10.\,y\,z^2 - 10.\,x\,y\,z^2 - 10.\,y^2\,z^2 - 2.\,z^3 + 4.\,x\,z^3 - 6.\,y\,z^3 - 2.\,z^4\}$$

◄ | ►

Example 5: Another Multivariate Polynomial GCD

Here is an example from a recent article by M. Sanuki.

```
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], {10^(-6), 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}
```

Example 6: A Numeric System Caught Misbehaving

This example is a Stewart platform system from Jan Verschelde's web site.

http://www.math.uic.edu/~jan/Demo/movestew.html

It came to my attention from our QA department. We were getting twice as many solutions as expected (80 instead of 40), and half of them both were huge and gave huge residuals (10^{18} or so), hence seemed somehow wrong.

[Another large system.]

```
Length[Variables[polys]]
```

```
9
```

To check, I artificially raised precision to 600 digits, and solved at high precision. There were still 80 solutions, of which 40 were still "large", but now they gave modest residuals. The time needed to solve at this precision was around 200 seconds.

Misbehaving Numeric System...

Clearly something was afoot. After some reading I decided the issue was that the input coefficients, in order to give the "right" solutions, need to satisfy certain relations and that they only did so to machine precision. Thus we got what would otherwise be solutions at infinity, appearing around 1/(machine epsilon) in size. Use of modest tolerances removed these rogue solutions (or rather, sent them packing to infinity, where they belong). This also ran around six times faster, as there is now no need for high precision.

```
Timing[res = NSolve[polys, Tolerance → {10^(-10), 10^(-3)}];]
Length[res]
Max[Abs[polys /. res]]
Max[Abs[Variables[polys] /. res]]
```

```
{32.738, Null}
```

```
40
```

$$1.58882 \times 10^{-14}$$

```
8.53752
```

SUMMARY

- Precision and accuracy ideas from numerical computation can be adapted to the setting of numerical Gröbner bases.
  - Troublesome cases: Coefficient sizes might be orders of magnitude apart. A precision tolerance might then remove coefficients that are actually needed.
  - Accuracy tolerances are less troublesome but often still require trial−and−error setting.
- Most examples covered seem to work efficiently and give reasonable results.
- Tentative conclusions

REFERENCES

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

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.

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

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.

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.

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.

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

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.

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.

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.

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.

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

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.

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/

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.

J. Keiper (1992). Numerical computation with *Mathematica* (tutorial notes). Electronic version:
http://mathsource.com/Content22/Enhancements/Numerical/0204−028

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

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

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.

D. Lichtblau (2008). Approximate Gröbner bases and overdetermined algebraic systems. Manuscript.

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.

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.

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.

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

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.

K. Shirayanagi (1996). Floating point Gröbner bases. Mathematics and Computers in Simulation 42:509−528.

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

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.

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.

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

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.