DON'T PANIC

The Hitch Hiker's Guide to Gröbner Bases: commutative algebra for amateurs

Krister Forsman, Dept. Electrical Engineering
Linköping University, S-581 83 Linköping, Sweden
email: `krister@isy.liu.se`

1992-02-03

Abstract. This report is about Gröbner bases. It is an attempt to explain a little what Gröbner bases are about without introducing commutative algebra, and to show how to use them for solving equation systems. However, being very compact, the report is far from complete. There are lots of examples and Maple computations in the report. Also, it has the words DON'T PANIC inscribed in large friendly letters on its cover.

1 Polynomial Equations

Gröbner bases (let's call them GB for short) are all about solving systems of equations. Nonlinear equations, that is. Actually, we will only allow polynomial nonlinearities...

Suppose we want to solve the system

$$x^3y^3 - 2 = 0 \tag{1}$$
$$x^3y^2 + y - 1 = 0 \tag{2}$$

It is probably a good idea to try and eliminate one of the variables to get an equation in one variable only. We can take $y{\cdot}(2) - (1)$ to get

$$y^2 - y + 2 = 0 \tag{3}$$

So the y solving the system (1)-(2) have to satisfy (3). This was pretty simple. But what about x? One way to find out about that is to use (3) in (2) in order to get

$$x^3(y-2) + y - 1 = 0 \tag{4}$$

that is

$$y(x^3+1) = 1 + 2x^3 \tag{5}$$

that is

$$y = \frac{2x^3+1}{x^3+1} \tag{6}$$

If we take this, plug it into (3) and clear denominators we get

$$4x^6 + 5x^3 + 2 = 0 \tag{7}$$

- But these computations were totally heuristic! There has to be a way of doing this systematically!

- You bet there is...

2 Gröbner Bases

GB:s provide a method for eliminating variables from a system of multivariate, polynomial equations. (A *multivariate* polynomial is a polynomial in several variables.) What operations are allowed during the elimination process? To answer this question we start by adopting the practical convention that all equations are written in normal form, i.e. with right hand side zero. This means that we only have to bother about the left hand sides - instead of manipulating equations we manipulate polynomials. Now during the elimination new polynomials p are formed from pairs of old ones f_1, f_2 in the following way:

$$p = \alpha f_1 + \beta f_2 \tag{8}$$

where α, β are polynomials. The set of all such p is called the *ideal* generated by f_1, f_2. In section 6 you can read more about ideals. This means that all new polynomials formed have those zeroes that are common to the original ones (and maybe some extra: see section 5).

What the GB-algorithm does is it chooses the α and β in (8) cleverly. For further details, see section 6.

So when the algorithm terminates it has produced a set of polynomials which are a consequence of the original ones. Some members of this set are more interesting than others: see below.

The really nice thing with GB:s is that there are many computer programs available to calculate them; for more information, see section 3. Here I will only tell how to do it in Maple. The only thing you have to know to be able to use the Maple GB-package is the following:

There are some parameters in the GB-calculations that are to be chosen by the user. The most important one is the *ranking* of the variables. This ranking determines which variables to eliminate first. For example if we want to eliminate x in order to get a polynomial in y only, then x should be higher ranked than y, written

$$x \succ y \tag{9}$$

(there is no standard notation here, yet). For the example above the GB is

$$\{y - 3 - 4x^3,\ 4x^6 + 5x^3 + 2\} \tag{10}$$

w.r.t. the ranking $y \succ x$, and

$$\{3 - y + 4x^3,\ y^2 - y + 2\} \tag{11}$$

w.r.t. the ranking $x \succ y$. Thus the last element of the GB in these two cases is a polynomial in one variable only: the algorithm has done the elimination for us.

In the next section I will tell you how to compute a GB in Maple.

Remarks:

1) GB:s were invented by Bruno Buchberger who was a student of the great algebraist Wolfgang Gröbner (1899-1980).

2) If you want to read more about Gröbner bases, check out the articles [4, 5, 11] or chapter 3 in the book [7].

3) If there are no solutions whatsoever to a system of equations then the GB is 1.

3 Programs for Computing GB:s

The symbolic algebra programs Maple [6], Reduce [9], Macsyma [14], and Mathematica [15] all contain packages for GB computations, some more advanced than others...

As I said I'm only going to consider the Maple GB-package here.

To load it you type `with(grobner):` once you started Maple. The function for computing GB:s is called `gbasis` and it has the syntax

```
gbasis(pollist,varlist,to)
```

where

- `pollist` is a list containing the polynomial lhs:s of the original problem
- `varlist` is a list containing the variables of the polynomials. This list determines the ranking of the variables: the highest ranking variable first and the lowest one last.
- `to` is a term ordering. I haven't told you about that yet, but that's only because you don't have to worry about it: only put it to `plex` (advanced user's may have a look in section 6 to learn more about this).

The example in sections 1 and 2 looks as follows in Maple:

```
> f1:=x^3*y^3-2;
                                    3  3
                             f1 := x  y  - 2

> f2:=x^3*y^2+y-1;
                                  3  2
                           f2 := x  y  + y - 1

> with(grobner):
> G1:=gbasis([f1,f2],[y,x],plex);

                                    3         3      6
                  G1 := [y - 3 - 4 x , 2 + 5 x  + 4 x ]

> G2:=gbasis([f1,f2],[x,y],plex);

                                      3   2
                   G2 := [- y + 3 + 4 x , y  - y + 2]
```

4 Comparison with Numerical Methods

But if we want to solve equations, why make things complicated? Why don't we just use good ol' numerical methods? Well, there are two categories of reasons:

- there are some things you can't do with numerics

- there are some not-so-nice properties of numerical methods

Among the things you can't do with numerics is

- you can't have parameter solutions

I.e. your system has to have finitely many solutions, which is pretty obviuos: compare with linear algebra where you can't invert a singular matrix.

Some of the non-nice properties of numerical methods for equation solving are:

- which solution is found depends on the initial guess in a really messy way (ever heard about chaos and fractals?) for Gauss-Newton.
- it is difficult to know for sure that all the solutions have been found

Of course there are disadvantages with GB:s too, the most severe one being the computational complexity. It seems to be a nice idea to combine numerics and symbolics to gain the advantages and evade the disadvantages of both.

5 Caveats

Even if GB:s are great there are some things you should bear in mind, not to mess things up.

Firstly, if you're an engineer you're probably looking for solutions in $\mathbb{R}^n$. GB:s don't care about this (you have to realize that this is a really, really tough problem). This means that after you did the elimination to get a polynomial p, say, you have to check if a zero of p corresponds to a real zero of all polynomials in the GB. For example, suppose you have obtained the GB

$$\{y^2 - x,\ x^2 - 1\} \tag{12}$$

There are two real zeroes of the last polynomial: $x = 1$, $x = -1$, but if you plug $x = -1$ into the first polynomial you get $y^2 + 1$ which doesn't have a real zero.

The second caveat is similar to the first one, only worse: it might happen that a zero of p doesn't even correspond to a complex zero of the other polynomials. The simplest example I know of is

$$\{xz - 1,\ y - x\} \tag{13}$$

Here $x = y = 0$ is a zero of the last polynomial, while this makes the first polynomial identically 1.

The moral is: don't just look at the last element of the GB, even if this is what you were looking for originally.

6 Some Advanced Stuff

OK, now that we know how to use GB:s we may dive into some formal, theoretical discussions. I will allow myself to jump freely between different aspects of GB:s and Buchberger's algorithm. I'll even make some formal definitions!

6.1 Some Words about Commutative Algebra

Definition 6.1 The set (or *ring*, if you will) of polynomials in the variables $x_1, \ldots, x_n$ with coefficients from k is denoted $k[x_1, \ldots, x_n]$. Here k can be e.g. $\mathbb{Q}, \mathbb{R}, \mathbb{C}$. □

Definition 6.2 Let $p_1, \ldots, p_m \in k[x_1, \ldots, x_n]$. The *ideal generated by* the p_i is the set of all polynomials f that can be written $f = \alpha_1 p_1 + \ldots + \alpha_m p_m$ for some $\alpha_i \in k[x_1, \ldots, x_n]$. It is denoted $(p_1, \ldots, p_m)$. □

The theory of ideals and their cousins is called *commutative algebra* or, if the main interest is in the zero-sets of the polynomials, *algebraic geometry.* There are no beginner's books on commutative algebra, but one of the most basic ones is [2]. Some books about algebraic geometry are [12, 10]. None of these contain anything about GB:s.

A GB is a special generating set of an ideal. It has the following property:

Theorem 6.1 *Let x, y be two sets of variables. If $\mathbf{G}$ is a Gröbner base for the ideal $I \subset k[x, y]$ w.r.t. the ranking $x \prec y$ then*

$$I \cap k[x] = (\mathbf{G} \cap k[x])$$

Proof. Proven in [5]. □

When x, y are sets, then by $x \prec y$ I mean that all elements of x are lower ranked than all elements of y.

It turns out that, with some extra requirements, GB:s are unique, so we may write $GB(I)$ for the Gröbner base of I.

There is one more thing I would like to explain about commutative algebra: it is the concept of algebraic dependence.

Definition 6.3 $p_1, \ldots, p_m \in k[x_1, \ldots, x_n]$ are *algebraically dependent* over k if there is a nonzero polynomial $f \in k[X_1, \ldots, X_m]$ such that $f(p_1, \ldots, p_m) \equiv 0$. □

There is a nice way to retrieve the dependency relation f using GB:s, which is described in e.g. [13]. The idea is simply to form the ideal

$$I = (z_1 - p_1(x), z_2 - p_2(x), \ldots, z_m - p_m(x)) \tag{14}$$

and then compute $GB(I)$ w.r.t. some ranking that eliminates the x_i. The z_i are called *tag variables.*

6.2 Another Useful Maple Function

A *term ordering* is a total ordering of all monomials in $k[x_1, \ldots, x_n]$, satisfying some axioms [11]. There are term orderings other than `plex`. These don't do elimination but are useful anyway, because some of them have a much lower complexity. The Maple function `finduni` first computes a GB w.r.t. a term order different from `plex` and then uses some linear algebra to find a univariate polynomial (in a given variable) belonging to the ideal. The reasons that this is often not enough are those mentioned in section 5. The idea is explained in [3]. An example of how to use `finduni`:

```
> f1:=x^3*y^3-2:  f2:=x^3*y^2+y-1:
> finduni(y,[f1,f2],{x,y});
                                    2
                                   y  - y + 2

> finduni(x,[f1,f2],{x,y});
                                        3      6
                                 2 + 5 x  + 4 x
```

6.3 Buchberger's Algorithm

The operations performed by Buchberger's algorithm are of two kinds: *remainders* and *S-polynomials*. The remainder and S-polynomial both depend on the term-ordering used. An example: we are using `plex` with ranking $y \succ x$. This means that, for instance,

$$y^3 > y^2x > y^2 > yx^7 > yx > x \tag{15}$$

Define the *initial term* of a polynomial as the highest ranked term of that polynomial, so that the inital term of (2) is y^2x^3. When taking the remainder of f_1 w.r.t. f_2 we check if the inital term of f_2 divides *any* term of f_1. If it does we subtract a suitable multiple of f_2. What we did to get (3) was simply to take the remainder of (1) w.r.t. (2). The Maple function `normalf` computes the remainder of a polynomial f w.r.t. a list F of polynomials. In order for this to be meaningful, F should be a GB, unless it consists of one element only. The syntax of `normalf` is: `normalf(f,F,vars,to)`. For example:

```
> f1:=x^3*y^3-2:  f2:=x^3*y^2+y-1:
> normalf(f1,[f2],[y,x],plex);
                                     2
                                  - y  + y - 2
```

The S-polynomial of two polynomials is a kind of pseudo remainder. Let us denote the initial term of the polynomial f by $\operatorname{in} f$. We have

Definition 6.4 Let $f_1, f_2 \in k[x_1, \ldots, x_n]$. The *S-polynomial* of f_1 and f_2 is defined

$$S(f_1, f_2) = h_1 f_1 - h_2 f_2 \tag{16}$$

where

$$h_1 = \frac{\operatorname{lcm}(\operatorname{in} f_1, \operatorname{in} f_2)}{\operatorname{in} f_1}, \qquad h_2 = \frac{\operatorname{lcm}(\operatorname{in} f_1, \operatorname{in} f_2)}{\operatorname{in} f_2} \tag{17}$$

□

It is proved in [4] that by just taking remainders and S-polynomials a Gröbner base is obtained in a finite number of steps. The Maple function `spoly` computes S-polynomials. It has the syntax `spoly(p1,p2,vars,to)`.

6.4 Relations with Other Algorithms

- How do GB:s relate to other algorithms for special cases of equation systems?

Well, in the special case where all polynomials are linear, Buchberger's algorithm is equivalent to Gaussian elimination and in the case of two univariate polynomials it is identical with Euclid's algorithm.

6.5 An Application to Control Theory

Of course there are lots and lots of applications of Gröbner bases to all kinds of science and technology. Applications to control theory can be found in [8].

Let me give one application to nonlinear systems. The problem is to find the input-output relation for a nonlinear system given in state space form. We assume that all nonlinearities are polynomial (as a matter of fact I can do it for rational functions, too). We then have

$$\dot{x}_1 = f_1(x,u),\ \ldots,\ \dot{x}_n = f_n(x,u), \quad y = h(x,u) \tag{18}$$

What we do now is we differentiate y w.r.t. time and replace all $\dot{x}_i$ with f_i (formally, we take *Lie derivatives*). In this manner we get two polynomials h_0, h_1 in $x_1, \ldots, x_n, u, \dot{u}$, subscript on h denoting time derivatives. We continue to do this until we reach y_n. Now, there is a rather elementary theorem in commutative algebra that states that $h_0, \ldots, h_n$ have to be algebraically dependent [8]. So we proceed as described in section 6.1 to find the dependency relation, i.e. the I/O-relation. The y_i work as tag variables!

A simple example:

$$\dot{x}_1 = u - x_1x_2, \quad \dot{x}_2 = x_1^2 - 2x_2, \qquad y = x_1 \tag{19}$$

We get that

$$I = \Big(y_0 - x_1,\ y_1 - u_0 + x_1x_2,\ y_2 - u_1 + (u_0 - x_1x_2)x_2 + x_1(x_1^2 - 2x_2)\Big) \tag{20}$$

The Gröbner base $GB(I)$ w.r.t. the ranking

$$x_2 \succ x_1 \succ y_2 \succ y_1 \succ y_0 \succ u_1 \succ u_0 \tag{21}$$

is easily computed. It contains one polynomial in the u_i and y_i, namely

$$p = y_0y_2 + 2y_0y_1 - 2y_0u_0 - y_0u_1 + y_0^4 - y_1^2 + y_1u_0 \tag{22}$$

We conclude that the I/O relation for the system (19) is

$$(\ddot{y} + 2\dot{y} - 2u - \dot{u} + y^3)\, y - \dot{y}^2 + u\dot{y} = 0 \tag{23}$$

Of course you can do this for discrete time systems too.

References

[1] D. Adams. *The Hitch Hiker's Guide to the Galaxy.* Pan Books, 1979.

[2] M.F. Atiyah and I.G. MacDonald. *Introduction to Commutative Algebra.* Addison-Wesley, 1969.

[3] W. Boege, R. Gebauer, and H. Kredel. Some examples for solving systems of algebraic equations by calculating Gröbner bases. *J. Symbolic Computation*, 1:83–98, 1986.

[4] B. Buchberger. Ein algorithmisches Kriterium für die Lösbarkeit eines algebraischen Gleichungssystems. *Aequationes Mathematicae*, 4:374–383, 1970.

[5] B. Buchberger. Gröbner bases: An algorithmic method in polynomial ideal theory. In N.K. Bose, editor, *Multidimensional Systems Theory*, pages 184–232. Dordrecht Reidel, 1985.

[6] B. Char, K.O. Geddes, G.H. Gonnet, M.B. Monagan, and S.M. Watt. *Maple Reference Manual.* Symbolic Computation Group, Univ. of Waterloo, fifth edition, March 1988.

[7] J.H. Davenport, Y. Siret, and E. Tournier. *Computer Algebra. Systems and Algorithms for Algebraic Computation.* Academic Press, 1988.

[8] K. Forsman. *Constructive Commutative Algebra in Nonlinear Control Theory.* PhD thesis, Dept. Electrical Engineering, Linköping University, S-581 83 Linköping, Sweden, 1991.

[9] A.C. Hearn. *Reduce User's Manual Version 3.4.* RAND Corp., Santa Monica, California, USA, July 1991.

[10] E. Kunz. *Introduction to Commutative Algebra and Algebraic Geometry.* Birkhäuser, 1985.

[11] F. Pauer and M. Pfeifhofer. The theory of Gröbner bases. *L'Enseignement Mathématique*, 34:215–232, 1988.

[12] M. Reid. *Undergraduate Algebraic Geometry*, volume 12 of *London Mathematical Society Student Texts.* Cambridge University Press, 1988.

[13] D. Shannon and M. Sweedler. Using Gröbner bases to determine algebra membership, split surjective algebra homomorphisms and determine birational equivalence. In L. Robbiano, editor, *Computational Aspects of Commutative Algebra*, pages 133–139. Academic Press, 1989. From J. Symb. Comp. Vol. 6, nr. 2-3.

[14] Symbolics Inc. *Macsyma Reference Manual*, November 1988. Version 13.

[15] S. Wolfram. *Mathematica. A System for Doing Mathematics by Computer.* Addison-Wesley, second edition, 1991.