

Buchberger's Algorithm*

Graeme Taylor

January 20, 2005

An Algorithm that generates a Gröbner basis for a collection of polynomials.

The various definitions of a Gröbner basis in terms of ideals describe the properties desired, but do not lend themselves to easy testing of a candidate basis. This is because the ideal generated by a basis will usually be of infinite size, and hence exhaustive testing of reductions will never be completed. Fortunately, with an additional concept it is possible to reformulate the notion of a Gröbner basis into a version that not only allows for testing a finite number of cases, but in fact allows for a basis to be grown from the original set of polynomials.

Definition: If f and g are polynomials, the S polynomial of f and g is given by

$$S(f, g) = \frac{lcm(lm(f), lm(g)) \times f}{lm(f)} - \frac{lcm(lm(f), lm(g)) \times g}{lm(g)}$$

Where lcm denotes the least common multiple and lm the leading monomial- so $lcm(lm(f), lm(g))$ is the smallest monomial divisible by both leading monomials.

It may not be immediately obvious, but with respect to a given ordering, the S polynomial will be 'simpler' than either of f and g - the leading monomial of each gets annihilated during the generation of S . Recognising this, we can now define a Gröbner basis in terms of S polynomials.

Given a set of polynomials $F = \{f_1, \dots, f_n\}$, a set $G = \{g_1, \dots, g_m\}$ is a Gröbner basis for F if the following conditions hold:

- The zeros of G are the zeros of F .
- $\forall 1 \leq i, j \leq m, S(g_i, g_j) \rightarrow_{G^*} 0$

This test will (for a finite G) therefore only require finitely many reductions to be carried out. Furthermore, if any of those reductions fail, we can stop immediately as we know we don't have a Gröbner basis.

*First appeared on Everything2, at http://www.everything2.com/index.pl?node_id=1694036

But should we? If a Gröbner basis is our goal, then all need not be lost because of a failed reduction. This is because any polynomial can be trivially reduced to zero- by itself. Furthermore, the S polynomial is an element of the ideal generated by its two arguments, and hence of that generated by the candidate basis as a whole. Thus, if we add it to the basis, we do not violate the first property (of the zeros matching).

This motivates Buchberger's algorithm. Given a finite set of polynomials F , we know a set of polynomials with the same zeros- F ! So we test the second property on all the pairings of elements of F , and if any of them should fail to reduce to zero, we add it to F . This will require more pairs to be tested, but not infinitely many (see later) so in the end we will have a suitable Gröbner basis. Note, however, that the order of testing might influence the shape of our basis- since an added S polynomial might have been able to reduce a previously bad case. Essentially, we do not get a unique result unless we go to the additional effort of auto-reducing the basis- eliminating elements from it to get a minimal form.

Note that if $S(f, g)$ reduces to 0, then so does $S(g, f)$. This immediately simplifies the number of pairings to consider. A (pseudocode) formalisation of Buchberger's algorithm would thus look something like the following.

Buchberger's algorithm

Input: $F = \{f_1, \dots, f_n\}$ a set of polynomials; j an ordering (to make sense of 'leading monomial').

Output: $G = \{g_1, \dots, g_m\}$ a gröbner basis for F .

we assume that F is a perfectly good starting point as a basis, and generate our list of pairings, exploiting the symmetry of S .

```

G:=F; m:=n
P := {(i, j) | 1 ≤ i < j ≤ n}
Now, we wish to test all the pairings.
While P is non-empty
  Remove an (i, j) from P
  Find  $S(g_i, g_j) \xrightarrow{G^*} h$ 
  If h equals zero, do nothing. Else,
    let  $g_{n+1} = h$ ,
    Set  $P = P \setminus \{(i, n+1) | 1 \leq i \leq n\}$ 
    increase n by 1,
return G

```

What's going on here? Well, we wanted all the S polynomials to reduce to zero. So we consider all the pairings, testing one at a time (removing it from our list to make sure we don't test it again). If we get zero, wonderful- but if not, we add what we got to the candidate basis. Now, we know it reduces itself to zero- but we may have inadvertently added something which cannot be reduced as an S polynomial with some other basis element. So to make sure, we add in all the new possible pairings to the list for testing.

Will this terminate?

Whilst the size of P decreases by one each time an (i, j) pair is picked for testing, the size of P increases by much more than one every time we find a bad pairing. Since we only return G once our list of pairings has emptied, it seems possible that the algorithm could fail to terminate. Fortunately, a result from pure mathematics (due to Noether) saves us- every time the basis G grows, so does the ideal $(lm(g))$. However, an increasing sequence of ideals is finite. So eventually, no growth in G is possible.

When will this terminate?

This is a much harder question which is of considerable interest in research. This is a somewhat naive algorithm, since we don't seek the most elegant formulation of G . In particular, if F features a large number of redundant polynomials (e.g. $F = \{x, x^2, x^3, x^4, x^5\}$), then computation time is wasted on analysing pairs of no value.

However, even if we supply a fairly well behaved F , the optimal choice of (i, j) pairs is unclear. (the pseudocode above makes no suggestions as to an approach). As discussed earlier, judicious choice of a pairing might provide us early on with an extra element that reduces many members of the basis that otherwise wouldn't- thus saving each of them generating an additional element. It seems that a good strategy is to pick a pair such that $lcm(lm(g_i), lm(g_j))$ is of minimal total degree (in the context of the ordering), but this isn't guaranteed to be unique.

Two other rules can save some effort in attempting the reduction of an S polynomial, as follows.

Buchberger's 1st Criterion

If $lm(f)$, $lm(g)$ are coprime, i.e., $lcm(lm(f), lm(g)) = lm(f)lm(g)$, then f, g suffice to reduce $S(f, g)$ to zero. Hence, no additional element for addition to the basis will be generated from such a pair, and it is preferable to process these cases first.

Buchberger's 3rd Criterion¹

If there is an h in G which divides $\text{lcm}(\text{lm}(f), \text{lm}(g))$, and the pairings (f, h) and (g, h) have already been processed, then $(f, g) \rightarrow_G^* 0$ so needn't be calculated. This saves us some computation.

1 Miscellaneous notes

The ordering employed can also strongly influence the running time of Buchberger's algorithm. In particular, the generally desirable purely lexicographical ordering is a poor choice here. So much so, in fact, that it is probably preferable to carry out Buchberger's algorithm on a basis arranged in total degree reverse lexicographical order, then use the FGLM algorithm to convert that back to a pure-lex form for subsequent determination of the roots.

Churning out a Gröbner basis 'by hand' in a computer algebra system (that is, tracking the lists and pairings yourself and working interactively to calculate reductions) isn't especially fast—although it might help you understand the process for a simple example, or, like me, you may find it a coursework exercise. Generally, though, you'd throw it at a built-in package.

In *Maple*, you'd need to be working `with(Ore_algebra)` and `with(Groebner)`. Then, supposing your variables are x, y, z you can define `A:=poly_algebra(x,y,z)`, place your list of polynomial in an array F , generate a suitable ordering by `T[org]:=termorder(A,tdeg(x,y,z)):`, then finally use the command `G[org]:=gbasis(F,T[org])`; to get a Gröbner basis G .

I'm not familiar with *Mathematica*, the other common CAS, but the *MathWorld* site claims that `GroebnerBasis[{poly1,poly2,...},{x1,x2,...}]` will do the trick.

¹Apparently the 2nd criterion doesn't offer much help!