The F_4 Algorithm

Dylan Peifer

May 4, 2017

1 Introduction

Gröbner bases are sets of polynomials with properties that make them especially useful for computation. Many questions about sets of polynomials, such as the existence of common roots or implicitization of a parametrization, can be computed easily from a Gröbner basis, so one of the key problems in computational algebra is how to efficiently compute a Gröbner basis from an arbitrary set of polynomials.

1.1 Gröbner bases

Let $R = k[x_1, \dots, x_n]$ be a polynomial ring over some field k.

Example 1. *Let $R = \mathbb{Q}[x, y]$. Then elements of R are all polynomials in variables x and y with rational coefficients. For example, $3x^2 + x + 1$ and $xy + \frac{1}{2}y^2x - y$ are both in R.*

One of the key tools used in manipulating single-variable polynomials is the division algorithm, which requires that we can order the terms of any polynomial. Thus to define a multivariate division algorithm we need a way to order terms in multivariate polynomials.

Definition 2. *Let $x_1^{\alpha_1}x_2^{\alpha_2}\dots x_n^{\alpha_n} = x^\alpha$ denote an arbitrary monomial where $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_n)$ is the vector of exponents. A* monomial order *on $R = k[x_1, \dots, x_n]$ is a relation $>$ on the monomials of R such that $>$ is a well-ordering and if $x^\alpha > x^\beta$ then $x^\gamma x^\alpha > x^\gamma x^\beta$ for any x^γ (i.e., $>$ respects multiplication).*

Example 3. *Lexicographic order (lex) is defined by $\alpha > \beta$ if the leftmost nonzero component of $\alpha - \beta$ is positive. For example, supposing $x > y$ gives $xy > y^4$ and $x^2y > x^2$.*

Definition 4. *Given a monomial order, define* $\mathrm{LT}(f)$ *to be the leading term of polynomial f. Define* $\mathrm{LM}(f)$ *to be the leading monomial (i.e.,* $\mathrm{LT}(f)$ *without coefficient).*

With ordering defined we can do multivariate division in approximately the same way as univariate division. In fact, the conditions on a monomial order are chosen precisely so that terms can be ordered, the division algorithm will terminate, and terms will not change order mid-step. In multivariate division we commonly have multiple divisors and can choose any valid divisor for each step. With dividend f and divisors $f_1, \dots, f_k$, this produces an expression of the form $f = f_1q_1 + \dots + f_kq_k + r$, where r is the remainder. Letting

$F = f_1, \ldots, f_k$ we will write $f^F \to r$ in this situation, which means that f reduces to or leaves a remainder of r when divided by F. Note that the q_i and even r are not necessarily unique since there may have been choices in the steps of the division algorithm.

Often when given a set of polynomials $\{f_1, \ldots, f_m\} \subseteq R$ we are really concerned with all possible polynomials we can generate as linear combinations of the f_i.

Definition 5. *The* ideal *I generated by polynomials $f_1, \ldots, f_m \in R$ is the set*

$$\langle f_1, \ldots, f_m \rangle = \{a_1 f_1 + \cdots + a_m f_m \mid a_1, a_2, \ldots, a_m \in R\}$$

Example 6. *Suppose we are searching for solutions of both $x^2y - y^2 = 0$ and $xy + x - 3 = 0$. Then for any polynomials $f(x, y)$ and $g(x, y)$ we know that $f(x, y)(x^2y - y^2) + g(x, y)(xy + x - 3) = 0$ when evaluated on these solutions, so we are really looking for common roots to all polynomials in $I = \langle x^2y - y^2, xy + x - 3 \rangle$.*

The advantage of looking at ideals is that a given ideal can have many different sets of generators, and since we are really only concerned with the ideal we are free to choose among different generating sets. A Gröbner basis of an ideal is simply a set of generators with a useful property.

Definition 7. *Given a monomial order, a* Gröbner basis *G of a nonzero ideal I is a subset $\{g_1, g_2, \ldots, g_k\} \subseteq I$ such that for all $f \in I$ we have that $\mathrm{LM}(f)$ is divisible by at least one of $\mathrm{LM}(g_1), \ldots, \mathrm{LM}(g_k)$.*

One way to understand this definition is to note that since any element $f \in I$ has a leading term divisible by some leading term of a g_i we can always perform a step of the division algorithm of f by that g_i to remove the leading term of f, and so the final remainder after dividing by G must be 0. This means a Gröbner basis G of I has the property that $f^G \to 0$ if and only if $f \in I$, which is surprisingly not the case for any generating set of I.

Example 8. *The polynomials $f_1 = x^2y - y^2$ and $f_2 = xy + x - 3$ do not form a Gröbner basis in lex order since $f_1 - xf_2 = -x^2 + 3x - y^2$ has a leading term not divisible by either leading term of f_1 or f_2. A Gröbner basis in lex of the ideal generated by f_1 and f_2 is $g_1 = y^4 + 2y^3 + y^2 - 9y$ and $g_2 = 3x + y^3 + y^2 - 9$. Note that g_1 is a single-variable polynomial that we could find roots of much easier than f_1 or f_2. This is one potential use for Gröbner bases.*

1.2 Buchberger's Algorithm

Buchberger's algorithm, which produces a Gröbner basis from any starting set of generators, is useful computationally and proves that any ideal has a Gröbner basis. The basic idea is contained in the following theorem. Proofs and further details can be found in [2].

Definition 9. *Let $S(f, g) = \frac{\mathrm{lcm}(\mathrm{LM}(f), \mathrm{LM}(g))}{\mathrm{LT}(f)} f - \frac{\mathrm{lcm}(\mathrm{LM}(f), \mathrm{LM}(g))}{\mathrm{LT}(g)} g$ where lcm is the least common multiple. This is the S-*polynomial *of f and g, where S stands for subtraction or syzygy.*

Theorem 10 (Buchberger's Criterion)**.** *Let $G = \{g_1, g_2, \ldots, g_k\} \subseteq I$ for some ideal I. If $S(g_i, g_j)^G \to 0$ for all pairs $g_i, g_j \in G$ then G is a Gröbner basis of I.*

This theorem leads naturally to an algorithm to compute a Gröbner basis of I from any generating set of I. The idea is to check all S-polynomials of our current generating set. If any S-polynomials do not leave remainder 0, we force them to reduce to 0 by adding their remainder to our generating set (of course as soon as we add a new element to the generating set we now also have many more pairs to consider). Termination follows from the ascending chain condition, as each time we add a new generator we are strictly increasing the size of the ideal generated by the leading terms of the generators.

```
Buchberger's Algorithm
Input: a set of generators F = {f_1, ..., f_k} for ideal I
Output: a Groebner basis G for I

G = F
k = size(G)
P = {(i, j) | 1 <= i < j <= k}
while size(P) > 0 do
    (i, j) = select(P)
    P = P \ {(i, j)}
    r = remainder of S(G_i, G_j) divided by G
    if r != 0 then
        G = G union {r}
        k = k + 1
        P = P union {(i, k) | 1 <= i < k}
return G
```

Here `P` is the pairs and `select` is a function that chooses the next S-polynomial to consider. Different implementations of `select` can affect the speed of the algorithm. Improved versions of Buchberger's algorithm, which take pairs in special order or can detect when it is possible to throw out pairs before performing the division algorithm, are still the standard method for computing the Gröbner basis of an ideal in most computer algebra systems.

2 F_4

The F_4 algorithm was introduced in [4] by Jean-Charles Faugère as an efficient algorithm for computing Gröbner bases. F_4 works in much the same way as the traditional Buchberger algorithm. The difference is that F_4 reduces large numbers of pairs at the same time by placing these pairs and associated elements in the rows of a matrix and using sparse linear algebra techniques to quickly reduce the matrix to row echelon form. In this way F_4 derives its increased efficiency over Buchberger by performing steps in parallel and taking advantage of fast algorithms for sparse row reduction.

2.1 Outline

The basic outline of F_4 is very similar to Buchberger's algorithm.

```
F4 Algorithm
Input: a set of generators F = {f_1, ..., f_k} for ideal I
Output: a Groebner basis G for I

G = F
k = size(G)
P = {(i, j) | 1 <= i < j <= k}
while size(P) > 0 do
    P’ = select(P)
    P = P \ P’
    G’ = reduction(P’, G)
    for h in G’ do
        G = G union {h}
        k = k + 1
        P = P union {(i, k) | 1 <= i < k}
return G
```

There are two major differences. First, the function `select` now returns a nonempty subset of `P` instead of a single element (in fact, if `select` always chooses a single element subset then F_4 will reduce to Buchberger’s algorithm). Second, we have introduced a new function `reduction` that produces a set of new basis elements from the input pairs `P’` and current basis `G`. This function `reduction` is the key piece of F_4.

```
reduction
Input: a set of pairs P’ and current basis G
Output: a set G’ of new basis elements

L = symbolicPreprocessing(P’, G)
M = matrix with rows the polynomials in L
M’ = reduced row echelon form of M
L’ = polynomials corresponding to the rows of M’
G’ = {f in L’ | LM(f) != LM(g) for any g in L}
return G’
```

In the third line of `reduction` we have replaced the expensive polynomial long division of Buchberger’s algorithm with a reduction to reduced row echelon form of a matrix whose rows represent the polynomials associated to `P’`. This matrix is constructed by taking all monomials found in polynomials of `L`, ordering them with $>$, and labeling the columns of `M` left to right with the monomials. Then each row consists of the coefficients of these monomials in a given polynomial. The goal is to mimic steps in the division algorithm by `G` with row operations, but this is not possible without adding many additional polynomials. In particular, during the division algorithm we multiply the divisor by a monomial and then subtract this product from the dividend, and while the subtraction is a valid row operation, making the product is not. The role of `symbolicPreprocessing` is precisely to make sure all such products are available as precomputed rows.

```
symbolicPreprocessing
Input: a set of pairs P' and current basis G
Output: a set L of polynomials

Left = {LCM(LM(G_i), LM(G_j))/LT(G_i) * G_i | (i, j) in P'}
Right = {LCM(LM(G_i), LM(G_j))/LT(G_j) * G_j | (i, j) in P'}
L = Left union Right
done = {LM(f) | f in L}
while done != Mon(L) do
    m = largest monomial in (Mon(L) \ done)
    done = done union {m}
    if LM(g) divides m for some g in G then
        f = choose g such that LM(g) divides m
        L = L union {m/LM(f) * f}
return L
```

Here `Mon(L)` represents all monomials found in polynomials of `L`. `L` starts with both halves of the S-polynomial for each pair, which will be separate rows in the eventual matrix. Then polynomials are added to `L` so that in the final output if any term in `Mon(L)` is divisible by some leading term in the current basis then there is a corresponding multiple of a basis element in `L` with that monomial as its leading term, which is precisely what is needed to perform a step of the division algorithm with that term.

Many improvements can be made to this basic outline. As with Buchberger's algorithm, different implementations of `select` can affect efficiency, and steps can be taken to avoid reducing pairs that are guaranteed to reduce to 0. Proofs, refinements, and further details along these lines can be found in [4].

2.2 Matrix Reduction

In Buchberger's algorithm the majority of computation is spent doing polynomial long division. In F_4 we have replaced this with spending the majority of the time row reducing matrices in the `reduction` subroutine. Many of these matrices are enormous - examples in [1] include matrices with 10^6 rows and columns. However, these matrices are also typically very sparse, as the total number of terms placed in `L` by `symbolicPreprocessing` is usually much larger than the total number of terms in a polynomial from the current basis. This sparsity helps us row reduce the matrix efficiently. Additionally, steps in row reduction and other matrix operations can be efficiently parallelized, which gives a significant improvement over the sequential reduction in Buchberger's algorithm.

2.2.1 LU Factorization

Problems in computational linear algebra are often expressed as matrix factorizations. The LU factorization is the matrix factorization description of Gaussian elimination, and factors a matrix A into $A = LU$, where L and U are lower and upper triangular matrices respectively. Since triangular systems are easy to invert with forward or back substitution the LU factorization is often used to efficiently solve $Ax = b$ for given b and unknown x.

To apply LU factorization to computing a row reduction of a $k \times m$ matrix M with $k > m$ we first use fast rank methods to determine the first k linearly independent columns and permute them to a front square submatrix A. The remaining columns form submatrix B, and then the row reduction will produce pivots in each column of A and transform B into $A^{-1}B$, which can be computed quickly with an LU factorization of A. Finally, columns are permuted back to starting positions (in fact, the steps can all be done in place).

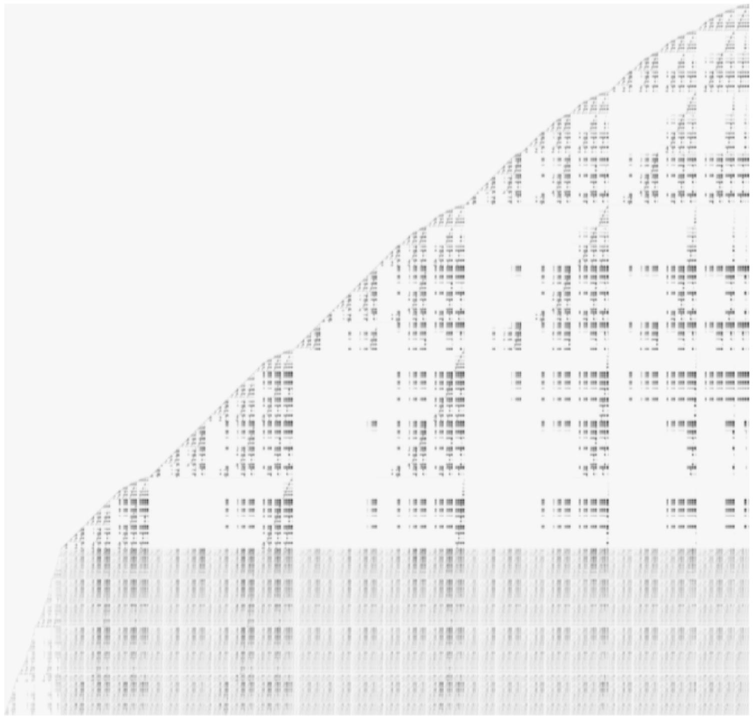
$$M \to \left[\begin{array}{c|c} A & B \end{array}\right] \to \left[\begin{array}{c|c} I & A^{-1}B \end{array}\right] \to \mathrm{rref}(M)$$

More advanced versions of LU factorization use pivoting to ensure that a factorization is found and computations are numerically stable [6].

2.2.2 Matrices from F_4

The standard algorithm for computing LU factorizations of sparse matrices is UMFPACK [3]. While the matrices in F_4 are sparse, they also have significantly more structure. In particular:

- Many pivots are immediately visible, meaning many columns have a row with its first nonzero entry in that column. This is because for every term in `L` (and thus for every column in the eventual matrix) `symbolicPreprocessing` attempted to produce a polynomial with that lead term (and thus a row with first nonzero entry in that column).
- Many rows are multiples of the same polynomial, as each row is constructed as a multiple of one of the current basis elements.
- For Gröbner basis computations we are typically working over finite fields.



An example matrix in F_4, taken from Figure 1 of [1]. Black entries are nonzero, and the almost triangular structure immediately reveals many pivots.

2.2.3 GBLA

Faugère and Lachartre introduced an algorithm specifically for reducing matrices coming from Gröbner basis computations in [5]. A basic version of this algorithm first permutes all immediately visible pivots into the diagonal of an upper left submatrix. Reduction of these rows then yields an identity and $A^{-1}B$, which can be computed with back substitution since A was already upper triangular. Then the pivots are used to zero out the lower left block, and finally the lower right block is reduced with standard reduction algorithms before reversing the original permutation and reconstructing the final reduced row echelon form.

$$M \to \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right] \to \left[\begin{array}{c|c} I & A^{-1}B \\ \hline C & D \end{array}\right] \to \left[\begin{array}{c|c} I & A^{-1}B \\ \hline 0 & D - CA^{-1}B \end{array}\right] \to \left[\begin{array}{c|c} I & A^{-1}B \\ \hline 0 & \mathrm{rref}(D - CA^{-1}B) \end{array}\right]$$

The block D is typically much smaller than A, so the time for its reduction is typically much smaller than the overall time for the algorithm. Many parts of this algorithm can also be parallelized, such as the action on distinct columns in B and D.

Boyar et al. develop the algorithms and data structures for reduction further in [1], where the Gröbner Basis Linear Algebra package (GBLA) is introduced. GBLA is now included as a linear algebra library for FGb, Faugère's own implementation of F_4 and its successor F_5.

2.3 Results

The following is a list of timing results on examples found in [7]. Each example is an ideal in a polynomial ring over a finite field ($\mathbb{F}_{101}$ or $\mathbb{F}_{32003}$) using grevlex order. Macaulay2 and Magma, two common computer algebra systems, both have built-in implementations of Buchberger's algorithm and F_4, and both were used to compare the two algorithms. Testing was performed on an i7-4770 at 3.40GHz with 16GB of RAM using Macaulay2 1.8.2 and Magma V2.21-5. All timings are in seconds.

	Macaulay2		Magma	
example	Buchberger	F_4	Buchberger	F_4
hcyclic8	320	4	111.6	1.12
jason210	16	6	5.65	2.79
katsura10	68	1	21.46	0.13
katsura11	955	4	272.01	0.64
mayr42	71	66	165.14	28.61
yang1	28	503	92.27	13.22

Magma's F_4 is fastest for all examples, and in particular is always faster than Buchberger, sometimes by several orders of magnitude. Macaulay2's Buchberger does beat Magma's Buchberger for the final two examples, but cannot beat Magma's version of F_4, and surprisingly Macaulay2's F_4 is slower on yang1 than Macaulay2's Buchberger. Magma and FGb are generally considered the fastest programs for Gröbner basis computation, but both are unfortunately closed source. Thus it is not clear what modifications or special implementation details have been made in Magma's algorithms. It is clear, however, that F_4 is a substantial improvement over Buchberger, and it is becoming the standard method in many computer algebra systems.

References

[1] Brice Boyer, Christian Eder, Jean-Charles Faugère, Sylvain Lachartre, and Fayssal Martani. GBLA: Gröbner basis linear algebra package. In *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation*, ISSAC '16, pages 135–142, New York, NY, USA, 2016. ACM.

[2] David A. Cox, John Little, and Donal O'Shea. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra.* Springer, fourth edition, 2015.

[3] Timothy A. Davis. A column pre-ordering strategy for the unsymmetric-pattern multifrontal method. *ACM Trans. Math. Softw.*, 30(2):165–195, 2004.

[4] Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases (F4). *Journal of Pure and Applied Algebra*, 139(1-3):61–88, 1999.

[5] Jean-Charles Faugère and Sylvain Lachartre. Parallel Gaussian Elimination for Gröbner bases computations in finite fields. In *Proceedings of the 4th International Workshop on Parallel and Symbolic Computation*, PASCO '10, pages 89–97, New York, NY, USA, 2010. ACM.

[6] Gene H. Golub and Charles F. Van Loan. *Matrix Computations.* The Johns Hopkins University Press, fourth edition, 2013.

[7] Bjarke Hammersholt Roune and Michael Stillman. Practical Gröbner basis computation. `www.broune.com/papers/issac2012.html`. Accessed: 2017-04-17.