Computing Gröbner Bases in Python with Buchberger's Algorithm

Michael Weiss

September 11, 2014

Abstract

This paper describes the theory of Gröbner bases and the author's implementation of Buchberger's algorithm used for computing them. We outline the need for a well defined notion of basis for multivariate polynomial ideals and some of the problems that this theory can solve. We discuss an implementation in Python of the polynomial arithmetic necessary for computing Gröbner bases using Buchberger's algorithm.

Introduction

Gröbner bases, introduced by Bruno Buchberger in his PhD thesis in 1970, are an important tool in computational algebraic geometry and commutative algebra. Before their discovery, one knew by the Hilbert Basis Theorem of the existence of a finite generating set for polynomial ideals, but not always how to construct one. A generating set for a polynomial ideal is a computationally important object since it allows us to answer some fundamental questions about polynomial ideals. For instance we would like to be able to tell when two ideals are equal. Also given an ideal we would like to be able to determine whether an arbitrary polynomial is a member of that ideal. The latter problem is called the ideal membership problem and will serve as the motivation for the theory presented in this paper.

In a univariate polynomial ring the ideal membership problem is easily resolved. Since univariate polynomial rings are principal ideal domains, one can find a single generator for any polynomial ideal. A remainder of zero after division by this generator is a necessary and sufficient condition for a polynomial's membership of the ideal. We will see that the division algorithm for multivariate polynomials is not always unique and thus to formulate a criteria for membership of an ideal, one needs a particular generating set that guarantees a unique remainder upon division. Gröbner bases are generating sets with this property. Surprisingly, this problem was not resolved until 40 years ago with Buchberger's introduction of Gröbner bases.

To implement Buchberger's algorithm we used the object-oriented Python programming language. We used classes in Python to simulate elements of polynomial rings and coefficient fields and successfully implemented polynomial arithmetic.

Monomial Orderings

In a univariate polynomial ring, there is a natural ordering on the monomials:

$$\cdots > x^{n+1} > x^n > \cdots > x^2 > x > 1$$

In a multivariate polynomial ring, there are multiple conventions for ordering monomials, leading to a number of different possible 'orderings' of monomials. Certainly it is not enough to compare the total degree of multivariate monomials would leave us unclear as to whether

$$(1) \qquad x^3y^2z < x^3yz^2 \qquad \text{or} \qquad (2) \qquad x^3y^2z > x^3yz^2.$$

Monomial orderings are a particular concern in computation since the results of certain algorithms, such as the division algorithm, can vary depending on which monomial ordering is chosen.

Definition. Let k be a field. A **monomial ordering** on $k[x_1, x_2, \ldots, x_n]$ is any relation $>$ on $\mathbb{Z}^n_{\geq 0}$ or equivalently, any relation on the set of monomials x^α, $\alpha \in \mathbb{Z}^n_{\geq 0}$, such that:

1. $>$ is a total ordering on $\mathbb{Z}^n_{\geq 0}$
2. If $x^\alpha > x^\beta$ and $\gamma \in \mathbb{Z}^n_{\geq 0}$, then $x^{\alpha+\gamma} > x^{\beta+\gamma}$
3. $>$ is a well-ordering on $\mathbb{Z}^n_{\geq 0}$. [1]

Note that here $\mathbb{Z}_{\geq 0}$ denotes the set of positive integers greater than or equal to 0 and $\mathbb{Z}^n_{\geq 0}$ n-tuples with entries in that set. It is convenient and sometimes simpler to represent monomials of more than one variable as a tuple, where each entry is the degree of the corresponding variable, that is, $x^\alpha = x_1^{\alpha_1} \cdots x_n^{\alpha_n}$ has representation $\alpha = (\alpha_1, \ldots, \alpha_n)$ in $\mathbb{Z}^n_{\geq 0}$. Also note that here the purpose of the well-ordering condition is to guarantee that the multivariate polynomial division algorithm, which we will introduce later, will eventually terminate [**?**].

In this project we considered three monomial orderings: lexicographic, graded-lexicographic, and graded reverse lexicographic ordering.

Definition. (**Lexicographic Ordering**)
Let $\alpha = (\alpha_1, \ldots, \alpha_n)$ and $\beta = (\beta_1, \ldots, \beta_n) \in \mathbb{Z}^n_{\geq 0}$. We say $\alpha >_{\text{lex}} \beta$ if, in the difference $\alpha - \beta \in \mathbb{Z}^n$, the leftmost nonzero entry is positive. So we say $x^\alpha >_{\text{lex}} x^\beta$, if $\alpha >_{\text{lex}} \beta$.

Some examples:

1. $xy^2 >_{\text{lex}} y^5z^6$ since $\alpha - \beta = (1, -3, -6)$
2. $x^2y^3z^5 >_{\text{lex}} x^2yz^5$ since $\alpha - \beta = (0, 2, 0)$

[1] Note: a well-ordering on $\mathbb{Z}^n_{\geq 0}$ is an ordering such that every nonempty subset of $\mathbb{Z}^n_{\geq 0}$ has a smallest element under $>$.

Lexicographic ordering is also sometimes referred to as dictionary ordering. The variables in the monomials are "read" from left to right, that is we first compare the leftmost variables in each monomial, if they have the same degree then compare the next variable, and so on. There are many lexicographic orderings depending on the ordering of the variables; typically we choose the ordering: $x_1 > x_2 > \ldots > x_n$. As can be observed in the examples, the ordering does not take into account the total degree of the monomial. Thus for any $n, m > 0$ we will always have $x >_{\text{lex}} y^n z^m$.

Definition. (Graded Lexicographic Ordering)
Let $\alpha, \beta \in \mathbb{Z}^n_{\geq 0}$. We say $\alpha >_{\text{grlex}} \beta$ if

$$|\alpha| = \sum_{i=1}^{n} \alpha_i > |\beta| = \sum_{i=1}^{n} \beta_i,$$

or $|\alpha| - |\beta|$ and $\alpha >_{\text{lex}} \beta$.

Some examples:

1. $xy^3z^2 >_{\text{grlex}} x^4y$ since $|(1,3,2)| = 6 > |(4,1,0)| = 5$
2. $xy^3z^2 >_{\text{grlex}} xyz^4$ since $|(1,3,2)| = |(1,1,4)|$ and $(1,3,2) >_{\text{lex}} (1,1,4)$.

Graded lexicographical ordering favors monomials with higher total degree and then uses the lexicographical ordering when the two monomials have equal degree.

Definition. (Graded Reverse Lexicographic Ordering)
Let $\alpha, \beta \in \mathbb{Z}^n_{\geq 0}$. We say $\alpha >_{\text{grlex}} \beta$ if

$$|\alpha| = \sum_{i=1}^{n} \alpha_i > |\beta| = \sum_{i=1}^{n} \beta_i,$$

or $|\alpha| = |\beta|$ and the rightmost nonzero entry of $\alpha - \beta \in \mathbb{Z}^n$ is negative.

Some examples:

1. $x^4y >_{\text{grevlex}} xy^3z$ since $|(1,3,1)| = |(4,1,0)|$ and $(4,1,0) - (1,3,1) = (3,-2,-1)$
2. $xy^3z^2 >_{\text{grevlex}} xyz^4$ since $|(1,3,2)| = |(1,1,4)|$ and $(1,3,2) - (1,1,4) = (0,2,-2)$.

Graded lexicographic still has the property of checking the total degree of the monomials but is the reverse of the graded lexicographic ordering since it looks at the rightmost variable of each monomial and favors the one with the smaller power.

To see how the different monomial orderings affect the ordering of polynomials, consider the polynomial $f = 5x^2y^3z^2 + 3y^5z - 5x^3 + 2xy^2z^3$.

1. With respect to the lexicograpical we order the terms of f:

$$f = -5x^3 + 5x^2y^3z^2 + 2xy^2z^3 + 3y^5z$$

2. With respect to the grlex order, we order the terms of f:
$$f = 5x^2y^3z^2 + 2xy^2z^3 + 3y^5z - 5x^3$$

3. With respect to the grevlex order, we order the terms of f:
$$f = 5x^2y^3z^2 + 3y^5z + 2x^2y^2z^3 - 5x^3$$

Definition. Let $f = \sum_\alpha c_\alpha x^\alpha$ be a nonzero polynomial in $k[x_1, \ldots, x_n]$ and let $>$ be a monomial order.

1. The **multidegree** of f is
$$\text{multideg}(f) = \max(\alpha \in \mathbb{Z}^n_{\geq 0} : a_\alpha \neq 0)$$
where the maximum is taken with respect to the monomial ordering $>$.

2. The **leading coefficient** of f is
$$\text{LC}(f) = a_{\text{multideg}(f)} \in k.$$

3. The **leading monomial** of f is
$$\text{LM}(f) = x^{\text{multideg}(f)}.$$

4. The **leading term** of f is
$$\text{LT}(f) = \text{LC}(f) \cdot \text{LM}(f).$$

Example. Consider the polynomial: $f = 4x^2y^2z^3 + 5xy^2z + 2y^3z^4$ with respect to the lex ordering. Then
$$\begin{aligned} \text{multideg}(f) &= (2, 2, 3), \\ \text{LC}(f) &= 4, \\ \text{LM}(f) &= x^2y^2z^3, \\ \text{LT}(f) &= 4x^2y^2z^3. \end{aligned}$$

Division Algorithm

In the division algorithm for polynomials of one variable, for the input of a divisor and a dividend, we are guaranteed a unique and well defined output of a quotient and remainder. However, as we will see for multivariate polynomials, the "quotients" and remainder depend upon the monomial ordering and on the order of the divisors in the division.

The division algorithm in the multivariable case allows us to divide $f \in k[x_1, \ldots, x_n]$ by $g_1, \ldots, g_s \in k[x_1, \ldots, x_n]$, so that we can express f in the form:
$$f = q_1g_1 + \ldots + q_sg_s + r.$$

The strategy is to repeatedly cancel the leading term of f by subtracting off an appropriate multiple of one of the g_i. We will see that the result of the division algorithm fails to be unique for multivariate polynomials because there may be a choice of divisor at each step.

Theorem 1. ***(Division Algorithm in*** $k[x_1, \ldots, x_n]$***)*** *Fix a monomial order $>$ on $\mathbb{Z}^n_{\geq 0}$, and let $G = (g_1, \ldots, g_s)$ be an ordered s-tuple of polynomials in $k[x_1, \ldots, x_n]$. Then every $f \in k[x_1, \ldots, x_n]$ can be written as*

$$f = q_1 g_1 + \ldots + q_s g_s + r,$$

where a_i, $r \in k[x_1, \ldots, x_n]$, and either $r = 0$ or r is a linear combination, with coefficients in k, of monomials, none of which is divisble by any of $\mathrm{LT}(g_1), \ldots, \mathrm{LT}(g_s)$. We call r a ***remainder*** *of f on division by G. Furthermore, if $q_i g_i \neq 0$, then we have*

$$\mathrm{multideg}(f) \geq \mathrm{multideg}(q_i g_i).$$

Example. Consider the following division using the lex ordering on the monomials. Let

$$f = x^3y^2 + xy + x$$

and our divisors be

$$g_1 = y^2 + 1,$$
$$g_2 = xy + 1.$$

Observe that $\mathrm{LT}(g_1) = y^2$ divides $\mathrm{LT}(f) = x^3y^2$. So we update the quotient, adding x^3. Now update the dividend:

$$(x^3y^2 + xy + x) - x^3(y^2 + 1) = -x^3 + xy + x.$$

Since $\mathrm{LT}(g_1) = y^2$ does not divide $-x^3$, we look at the next polynomial in the list of divisors and see that $\mathrm{LT}(g_2) = xy$ also does not divide it. So proceed to the next term in f, which is xy. As g_1 comes first in the list of divisors, we check whether $\mathrm{LT}(g_1) = y^2$ divides xy; since it does not, we move on to $\mathrm{LT}(g_2) = xy$ and notice that it does. So we update the quotient corresponding to g_2 by adding 1. Again we update the dividend to obtain:

$$(-x^3 + xy + x) - 1 \cdot (xy + 1) = -x^3 + x - 1.$$

Observe that none of the leading terms of the divisors divides any of the terms of the dividend. Thus the division is complete and the dividend becomes the remainder.

Collecting everything together we have:

$$x^3y^2 + xy + x = x^3 \cdot (y^2 + 1) + 1 \cdot (xy + 1) + (-x^3 + x - 1).$$

So $q_1 = x^3$, $q_2 = 1$ and $r = -x^3 + x - 1$. Note that if we had instead performed the division with $f_1 = xy + 1$ and $f_2 = y^2 + 1$, we would have obtained a different answer:

$$x^3y^2 + xy + x = 0 \cdot (y^2 + 1) + (x^2y - x + 1) \cdot (xy + 1) + (2x - 1).$$

The need for a well defined remainder upon division is one of the motivations for the definition of "Gröbner" bases.

Monomial Ideals and Gröbner Bases

In general we do not obtain a uniquely determined remainder from the division algorithm. However, the subsequent definition of a Gröbner basis will have the quality that the division of f by G yields the same remainder r no matter how the elements of G are ordered in the division. Since we will show that every ideal I has a Gröbner basis, we are able to resolve the ideal membership problem with a necessary and sufficient condition for a polynomial f to be a member of an ideal I; namely that division of f by the Gröbner basis of I returns a remainder of 0.

Definition. A **monomial ideal** is an ideal generated by a set of monomials.

That is, I is a monomial ideal, if there is a subset $A \subset \mathbb{Z}^n_{\geq 0}$ such that I consists of all polynomials which are finite sums of the form $\sum_{\alpha \in A} h_\alpha x^\alpha$, where $h_\alpha \in k[x_1, \ldots, x_n]$. We write $I = \langle x^\alpha \mid \alpha \in A \rangle$. For example $I = \langle x^5y^2z, x^2yz^2, xy^3z^2 \rangle \subset k[x, y, z]$ is a monomial ideal.

For all monomial ideals we have the fact that if x^β lies in I, then x^β is divisible by x^α for some $\alpha \in A$. Furthermore for every polynomial f in a monomial ideal I, we can say that every term of f lies in I and that f is a k-linear combination of the monomials in I.

Definition. Let $I \subset k[x_1, \ldots, x_n]$ be a nonzero ideal.

1. Let $\mathrm{LT}(I)$ be the set of leading terms of elements of I.

$$\mathrm{LT}(I) = \{cx^\alpha \mid \text{ there exists } f \in I \text{ with } \mathrm{LT}(f) = cx^\alpha\}$$

2. We denote by $\langle \mathrm{LT}(I) \rangle$ the ideal generated by the elements of $\mathrm{LT}(I)$.

So for example $\mathrm{LT}(I)$ is a monomial ideal. As we will soon see the ideals $\langle \mathrm{LT}(I) \rangle$ and $\langle \mathrm{LT}(g_1), \ldots, \mathrm{LT}(g_s) \rangle$ are not always the same. Though we always have $\langle \mathrm{LT}(g_1), \ldots, \mathrm{LT}(g_s) \rangle \subset \langle \mathrm{LT}(I) \rangle$ there are cases where the opposite inclusion does not hold. For instance consider the following example:

Example. Let $I = \langle f_1, f_2 \rangle$ where $f_1 = x^3 - 2xy$ and $f_2 = x^3y - 2y^2 + x$, and use lex ordering on the monomials in $k[x, y]$. Then:

$$f_3 := y \cdot (x^3 - 2xy) - (x^3y - 2y^2 + x) = -2xy^2 + 2y^2 - x,$$

so $f_3 \in I$, and $\mathrm{LT}(f_3) = -2xy^2 \in \langle \mathrm{LT}(I) \rangle$, but not in $\langle \mathrm{LT}(f_1), \mathrm{LT}(f_2) \rangle$ since it is not divisible by the leading terms of f_1 or f_2.

Since our goal is to obtain ideals that have the property that $\langle \mathrm{LT}(I) \rangle = \langle \mathrm{LT}(g_1), \ldots, \mathrm{LT}(g_s) \rangle$, we want to eliminate cases like the above by making sure that our basis generates all of $\langle \mathrm{LT}(I) \rangle$. This motivates the following definition:

Definition. Let a monomial ordering on $k[x_1, \ldots, x_n]$ be fixed. A finite subset $G = \{g_1, \ldots, g_t\}$ is a **Gröbner basis** if

$$\langle \mathrm{LT}(g_1), \ldots, \mathrm{LT}(g_s) \rangle = \langle \mathrm{LT}(I) \rangle$$

As a corollary to the Hilbert Basis Theorem applied to $\langle \mathrm{LT}(I) \rangle$ we have:

Corollary 1. *Let I be a non zero polynomial ideal, than I has a Gröbner basis.*

While this corollary gets us started by proving the existence of a Gröbner basis, it's proof is not constructive and offers us little insight as to how to actually obtain one. We would like to obtain a generating set such that all the leading terms of the polynomials in the set generate the leading terms of the ideal I. This fails when there is a cancellation of leading terms of the kind in the previous example.

To better determine when this cancellation occurs we construct a special polynomial that produces new leading terms.

Definition. Let $f, g \in k[x_1, \ldots, x_n]$ be nonzero polynomials.

1. If $\text{multidegree}(f) = \alpha$ and $\text{multidegree}(g) = \beta$, then let $\gamma = (\gamma_1, \ldots, \gamma_n)$, where $\gamma_i = \max(\alpha_i, \beta_i)$ for each i. We call x^γ the **least common multiple** of $\text{LT}(f)$ and $\text{LT}(g)$, written $x^\gamma = \text{LCM}(\text{LM}(f), \text{LM}(g))$.

2. The **S-polynomial** of f and g is the combination:

$$S(f, g) = \frac{x^\gamma}{\text{LT}(f)} \cdot f - \frac{x^\gamma}{\text{LT}(g)} \cdot g.$$

Example. Let $f = x^4yz + x^2y^3z + xz$ and $g = 2x^2y^2z + xy^2 + xz^3$ in $\mathbb{Q}[x, y, z]$ with the lexicographic ordering on the monomials. Then $\gamma = (4, 2, 1)$ and we have:

$$\begin{aligned} S(f, g) &= \frac{x^4y^2z}{x^4yz} \cdot f - \frac{x^4y^2z}{2x^2y^2z} \cdot g \\ &= y \cdot f - \frac{1}{2}x^2 \cdot g \\ &= -\frac{1}{2}x^3y^2 - \frac{1}{2}x^3z^3 + x^2y^4z + xyz \end{aligned}$$

Notice the cancellation of the leading terms occured by the construction of the S-polynomial. Once a basis contains all the possible S-polynomials of the polynomials in the ideal generating set, there are no extra polynomials in $\langle \text{LT}(I) \rangle$ that are not in $\langle \text{LT}(I) \rangle = \langle \text{LT}(g_1), \ldots, \text{LT}(g_s) \rangle$. This leads to the very important:

Theorem 2. *(**Buchberger's Criterion**). Let I be a polynomial ideal. Then a basis $G = \{g_1, \ldots, g_s\}$ for I is a Gröbner basis for I if and only if for all pairs $i \neq j$, the remainder on division of $S(g_i, g_j)$ by G is zero.*

Gröbner Bases and Buchberger's Algorithm

In the following discussion we will give the formulation of Buchberger's algorithm and an example of its use to calculate a Gröbner basis.

For greater clarity in our discussion, we introduce the following notation:

Definition. We write $\overline{f}^G$ for the remainder on division of f by the list of polynomials $G = \{g_1, \ldots, g_s\}$.

For example if $G = (x^3y^2 - y^2z, xy^2 - yz)$ using lex order on the monomials, then:

$$\overline{x^5y^3}^G = yz^3$$

since we have by the division algorithm

$$x^5y^3 = (x^2y) \cdot (x^3y^2 - y^2z) + (xyz + z^2) \cdot (xy^2 - yz) + yz^3.$$

Theorem 3. *(**Buchberger's Algorithm**) Let $I = \langle f_1, \ldots, f_s \rangle \neq (0)$ be a polynomial ideal. Then a Gröbner basis for I can be constructed in a finite number of steps.*

Buchberger's algorithm proceeds like this: Let $F = (f_1, \ldots f_s)$ be a list of the polynomials defining I. For each pair of polynomials f_i, f_j in F calculate their S-polynomial, S, and divide it by the polynomials $f_1, \ldots f_s$ in F obtaining $\overline{S}^F$. If $\overline{S}^F \neq 0$, add $\overline{S}^F$ to F and start again with $F = F \cup \{\overline{S}^F\}$. Repeat the process until all S-polynomials of polynomials in F have remainder 0 after division by F.

Example.
Consider the ring $k[x, y, z]$ with lex order. Let $I = \langle -2xy + x, x^3y - 2x^2 + y \rangle$. Let $F = (-2xy + x, x^3y - 2x^2 + y)$. Since $S(f_1, f_2) = \frac{1}{2}x^3 - 2x^3 + y$ and $\overline{S(f_1, f_2)}^F = \frac{1}{2}x^3 - 2x^2 + y \neq 0$, we add $\overline{S(f_1, f_2)}^F$ to F by adding it as a new generator $f_3 = \frac{1}{2}x^3 - 2x^2 + y$. And set $F = (f_1, f_2, f_3)$. Computing S polynomials we obtain:

$$\begin{aligned}
S(f_1, f_2) &= \frac{1}{2}x^3 - 2x^2 + y \\
\overline{S(f_1, f_2)}^F &= 0 \\
S(f_1, f_3) &= \frac{1}{2}x^3 - 4x^2y + 2y^2 \\
\overline{S(f_1, f_3)}^F &= 2y^2 - y
\end{aligned}$$

Thus we must add $f_4 = 2y^2 - y$ to our generating set. Letting $F = (f_1, f_2, f_3, f_4)$, compute:

$\overline{S(f_1, f_2)}^F = \overline{S(f_1, f_3)}^F = 0$

$S(f_2, f_3) = -4x^2y + 2x^2 + 2y^2 - y = 2x \cdot (-2xy + x) + 1 \cdot (2y^2 - y)$

$\overline{S(f_2, f_3)}^F = 0$

$S(f_2, f_4) = -\frac{1}{2}x^3y + 2x^2y - y^2 = \left(\frac{1}{4}x^2 - x\right) \cdot (-2xy + x) - \frac{1}{2} \cdot \left(\frac{1}{2}x^3 - 2x^2 + y\right) - \frac{1}{2} \cdot (2y^2 - y)$

$\overline{S(f_2, f_4)}^F = 0$

$S(f_3, f_4) = -\frac{1}{2}x^2y^2 - 2y^3 = \left(\frac{1}{4}x^2 - 2xy - x\right) \cdot (-2xy + x) - \frac{1}{2} \cdot \left(\frac{1}{2}x^3 - 2x^2 + y\right) + \left(-y - \frac{1}{2}\right) \cdot (2y^2 - y)$

$\overline{S(f_3, f_4)}^F = 0$

Since $S(f_1, f_4) = 0$, we have $\overline{S(f_i, f_j)}^F = 0$ for all $1 \leq i \leq j \leq 4$. By Buchberger's Criterion, it follows that $F = (f_1, f_2, f_3, f_4) = (-2xy + x, x^3y - 2x^2 + y, \frac{1}{2}x^3 - 2x^2 + y, 2y^2 - y)$ is a Gröbner basis for I.

Implementation

As this was a mostly exploratory project, much of the research time was spent writing a computer algebra system in the Python language to implement the arithmetic over a multivariate polynomial ring. Though any of the well-known computer algebra systems like Magma, Mathematica, or Sage already have the ability to output a Gröbner basis from a polynomial ideal – indeed, at a much faster speed than could be accomplished by a project of this scope – this route was taken for the sake of a deeper understanding of the algorithms.

Python is a high-level object oriented programming language. In this project we used the language's class objects to implement a polynomial ring and elements of rings. In all the implementation was comprised of a hierarchy of four classes: coefficient fields, polynomial rings, monomials, and polynomials. The structure of this implementation followed the algebraic structure for polynomials and the class hierarchy was constructed according to this structure.

We used two coefficient fields: the rational field and the prime number field. Since most of the arithmetic over these fields is already implemented as the arithmetic over the integers, the main consideration for this portion was to maintain a standard representation of elements. For this purpose we used the classical algorithms for computing the greatest common divisor, least common multiple, and the extended euclidean algorithm. The class was written so that the output of the arithmetic could consistently interface with the other routines.

It contained routines for representing the variables of a polynomial ring and a coercion function for conversion of objects of different classes to a 'polynomial' class object. We wrote three subclasses of the polynomial ring class, each one to implement one of the different monomial orderings. The subsequent monomial class contained basic operations for monomial multiplication and monomial division.

The polynomial class was the largest and the most computationally interesting of the classes. In this implementation, polynomials were represented as sparse lists of monomials, sorted with respect to a monomial order $>$ of monomials together with a list of coefficients. Polynomial multiplication was implemented using a sparse polynomial multiplication algorithm. The algorithm can be described as follows: let $f = a_1X_1 + \ldots a_nX_n$ and $g = b_1Y_1 + \ldots b_mY_m$, with $n > m$ and monomials X_i and Y_j not necessarily in order. Calculate $f \cdot b_1Y_1, \ldots, f \cdot b_nY_n$ and then use a "divide and conquer" approach to order these terms, that is recursively summing the first $\frac{n}{2}$ and the last $\frac{n}{2}$ terms and then summing. This multiplication algorithm runs in $O(mn \log m)$ time [**?**].

We were successful in implementing Buchberger's algorithm. While the algorithm implemented did not have an optimal runtime, it was accurate. The results of the algorithm were crosschecked with the results in Sage for polynomial rings with prime field coefficient fields of low characteristic.

Conclusion

We have discussed the theory of Gröbner bases and their use in solving certain computational problems. Gröbner bases provide for a uniquely determined remainder in the division of polynomial algorithms and thus give a solution to the ideal membership problem. Using Python one can implement polynomial arithmetic and buchberger's algorithm. We used some of the classical and best known algorithms for polynomial arithmetic in our implementation. This implementation

could be extended to perform numerous other operations in a polynomial ring and provides a fast and accessible foundation for further inquiries in algorithms over polynomial rings.