Bases de Gröbner e ideales diferenciables

Celia Aguelo Jiménez

Trabajo de fin de grado en Matemáticas
Universidad de Zaragoza

Director del trabajo: José Ignacio Cogolludo
27 de noviembre 2018

Abstract

Gröbner bases are a particular form of generating a set of ideals in a ring of polynomials over any field. By employing this way of ideal generators, it is possible to obtain unique remainders when making the division algorithm. This allows one to solve a series of problems such as the set of zeroes of a polynomial equation or a system.

At the beginning of the project, a series of very standard definitions and properties will be explained, starting from the definition of monomial, greatest common divisor, polynomials, affine spaces, affine varieties, and ideals. Also, a first introduction to the division algorithm for polynomials in one variable will be given.

After introducing these terms and properties, a study of the Gröbner bases construction method will be carried out, more specifically, Buchberger's algorithm, which is based on the division algorithm in $k[x_1, ..., x_n]$.

In order to do this, the concept of monomial orderings will be presented and some examples of it will be given. This concept is of great importance to carry out the division algorithm of polynomials in several variables so that once a monomial ordering is selected, the remainder can be uniquely determined.

Once this is clarified, this project focuses on giving the definition of a Gröbner basis, via the ideal of leading terms. The celebrated Hilbert basis theorem can be proved using this ideal of leading terms. Also a series of properties and applications such as the ascending chain condition and the concept of S-polynomial is provided.

To end this section the Buchberger's algorithm will be explained. This algorithm generates a Gröbner basis from an ideal. Using Hilbert's basis theorem one can find a finite set of generators. Starting with these generators, a set of S-polinomials is generated and by means of the division algorithm the non-zero remainders are added to the set of generators. This process is repeated until a Gröbner basis is obtained. Nevertheless, generating a Gröbner basis this way results in redundant terms. For this reason, the concepts of minimal Gröbner basis and reduced Gröbner bases are introduced, through which these redundant terms are excluded.

Due to the good result of Buchberger's algorithm, the ultimate goal of this project is to find a similar method in the situation where differential polynomial rings are being worked upon.

This is not simple, since we have new definitions of differential rings, differential ideals, differential polynomials and differential monomials. The concepts of weight, quasi-homogeneous differential polynomial, and quasi-homogeneous differential component are introduced.

One of the found difficulties is that these polynomial rings have a infinite number of variables and so both differential rings and differential ideals need to satisfy some special properties in order to allow a finite process similar to Buchberger's algorithm. One of these properties is that the ground field is a

field of constants of characteristic zero. In addition, ideals must be radical.

Once the difficulties are known, the process begins by introducing the concept of H-bases of differential ideals that will act as the equivalent of Gröbner bases. In order to construct it, a process similar to the Bucheberger's algorithm will take place, which we will call this the reduction process.

To wrap up, an example of the reduction process will be shown. Starting from a differential system that cannot be solved in a simple way, each one of the steps will be applied to obtain a simpler differential system that can be easily solved.

Each of the equations of the system is a part of the initial differential ideal, and a calculation of the S-polynomial as in the Buchberger's algorithm is performed, only the employed way of calculating it will be different because of the weights. This polynomial is the one the reduction process is applied to, which in reality is similar to the division algorithm, with the purpose of lowering the weight of the new differential polynomial. This way this new polynomial will be added to the basis until we get a zero polynomial, and thus the H-bases will be generated, which will allow to simplify our system.

Índice general

Introducción

La teoría de Bases de Gröbner se desarrolló en los años 60 junto con el algoritmo de Buchberger teniendo un gran impacto en el álgebra computacional. Esto ha hecho posible el cálculo eficiente de ecuaciones polinómicas permitiendo investigar complicados ejemplos. Es por esto que las bases de Gröbner tienen multitud de aplicaciones en campos como la criptografía, la teoría de grafos, la robótica, la resolución de sistemas de ecuaciones polinómicas,...

Entre todas estas aplicaciones nos vamos a centrar en la resolución de sistemas de ecuaciones con polinomios diferenciales.

El objetivo de este trabajo es desarrollar la teoría de las bases de Gröbner para llegar a resolver ecuaciones polinómicas y sistemas de ecuaciones diferenciales polinómicas tanto de manera teórica como realizando un programa en Sage que permita su resolución.

Para ello este trabajo está estructurado en cuatro capítulos:

En el capítulo 1 presentamos definiciones que nos permiten introducir las bases de Gröbner, su desarrollo está basado en el texto [1], (Capítulo 1).

En el capítulo 2 se introduce el concepto de las bases de Gröbner y el algoritmo de Buchberger que permite calcularlas, basándonos también en el texto [1], (Capítulo 2, párrafos 1-7).

En el capítulo 3 se extienden los conceptos aprendidos en los capítulos anteriores a anillos diferenciales, explicando un método equivalente al algoritmo de Buchberger llamado proceso de reducción. Para el cual nos hemos apoyado en los textos [2] y [3].

En el capítulo 4 se realiza un ejemplo que ilustra la teoría anterior. Para su resolución se ha implementado en Sage una serie de programas que facilitan su resolución que se muestran en el anexo (página 29).

Capítulo 1

Primeras definiciones

En este capítulo se introducen una serie de conceptos para polinomios algebraicos que posteriormente extenderemos a polinomios diferenciales.

1.1. Polinomios y espacio afín

Definición 1.1. Un monomio en $x_1,...,x_n$ es un producto de la forma: $x_1^{\alpha_1}\cdot x_2^{\alpha_2}\cdot ...\cdot x_n^{\alpha_n}$, donde los α_i con $i=1,...,n$ son enteros no negativos. A la suma de estos exponentes se le llama *grado total del monomio*.

Un monomio se escribe de forma simplificada como: x^α donde $\alpha=(\alpha_1,\alpha_2,...,\alpha_n)$ así podemos denotar el grado total como $|\alpha|$. Tenemos que tener en cuenta que si $\alpha=(0,0,...,0)$ entones $x^\alpha=1$.

Definición 1.2. Llamamos polinomio en $x_1,...,x_n$ con coeficientes, a_α, en k, a una combinación lineal de monomios y lo podemos escribir como:

$$f=\sum_\alpha a_\alpha x^\alpha.$$

En un polinomio llamamos término de f a $a_\alpha x^\alpha$ si el coeficiente $a_\alpha\neq 0$.

El grado total de f se denota $gr(f)$ y es el máximo grado de todos los monomios con coeficiente no nulo.

El conjunto de los polinomios con coeficientes en k se denota $k[x_1,...,x_n]$.

La suma y el producto de polinomios es también un polinomio de $k[x_1,...,x_n]$.

Diremos que f divide a un polinomio g si $g=fh$ para algún $h\in k[x_1,...,x_n]$.

El conjunto de polinomios $k[x_1,...,x_n]$, es un anillo, además es conmutativo y lo denotamos anillo de polinomios.

Definición 1.3. Llamamos espacio afín n-dimensional con n un entero al conjunto:

$$k^n=\{(a_1,...,a_n):\ a_1,...,a_n\in\ k\}.$$

Por ejemplo $\mathbb{R}$ (la recta afín) y $\mathbb{R}^2$ (el plano afín) son espacios afines.

Para relacionar los polinomios con los espacios afines vemos que los polinomios nos dan una función $f:k^n\to k$ que dada $(a_1,...,a_n)\in k^n$ sustituye cada x_i por a_i. De esta manera tenemos una relación entre el álgebra y la geometría. Esta doble naturaleza nos permite obtener resultados como:

Proposición 1.1. *Sea k un cuerpo infinito y $f\in k[x_1,...,x_n]$. Entonces $f=0$ en $k[x_1,...,x_n]$ si y solo si $f:k^n\to k$ es la función nula.*

Corolario 1.2. *Sea k un cuerpo infinito y $f,g\in k[x_1,...,x_n]$. Entonces $f=g$ en $k[x_1,...,x_n]$ si y solo si $f:k^n\to k$ y $g:k^n\to k$ son la misma función.*

Teorema 1.3. *Todo polinomio $f\in\mathbb{C}[x]$ no constante tiene una raíz que pertenece a $\mathbb{C}$.*

1.2. Variedades afines

Definición 1.4. Se dice variedad afín definida sobre $f_1,...,f_s$, un conjunto de polinomios en $k[x_1,...,x_n]$ con k cuerpo, a $\mathbf{V}(f_1,...,f_s) = \{(a_1,...,a_n) \in k^n : f_i(a_1,...,a_n) = 0 \ \forall 1 \leq i \leq s\}$. Es el conjunto de soluciones de un sistema de ecuaciones $f_1(x_1,...,x_n) = ... = f_s(x_1,...,x_n) = 0$.

Notamos que en el plano $\mathbb{R}^2$ los círculos, elipses, parábolas e hipérbolas son variedades afines, así como los grafos de funciones polinómicas. En el espacio $\mathbb{R}^3$ algunos ejemplos de variedades afines son el paraboloide de revolución $\mathbf{V}(z-x^2-y^2)$, el cono $\mathbf{V}(z^2-x^2-y^2)$ o una superficie dada por $\mathbf{V}(x^2-y^2z^2+z^3)$ que tienen puntos de singularidad. También es una variedad la curva cúbica torcida (twisted cubic) $\mathbf{V}(y-x^2,z-x^3)$ que es la intersección de las superficies $y=x^2$ y $z=x^3$, así tenemos que dos superficies en $\mathbb{R}^3$ dan una curva.

En mayor dimensión podemos considerar un cuerpo k y un sistema de m ecuaciones lineales con n incógnitas $x_1,...,x_n$ con coeficientes en k:

$$\begin{aligned} a_{11}x_1 + ... + a_{1n}x_n &= b_1, \\ &\vdots \\ a_{m1}x_1 + ... + a_{mn}x_n &= b_m. \end{aligned}$$

La solución forma una variedad en k^n llamada variedad lineal. De hecho, las rectas y planos son variedades lineales. En las variedades lineales la codimensión la determina el número de ecuaciones independientes.

Por último notamos que si $\mathbf{V},\mathbf{W} \subset k^n$ son variedades afines, entonces la unión, $\mathbf{V} \cup \mathbf{W}$, e intersección, $\mathbf{V} \cap \mathbf{W}$, también lo son.

Definición 1.5. Dado un cuerpo k. Una función racional en $t_1,...,t_n$ con coeficientes en k es un cociente f/g de dos polinomios $f,g \in k[t_1,...,t_n]$, donde g es un polinomio no nulo. Además, dos funciones racionales f/g y p/h son iguales siempre que $hf = pg$ en $k[t_1,...,t_n]$.

El conjunto de las funciones racionales se denota $k(t_1,...,t_n)$ y es un cuerpo.

Hay varias formas de representar una variedad afín aunque no siempre podemos representarlas de todas estas formas.

Suponemos que tenemos una variedad $V = \mathbf{V}(f_1,...,f_s) \subset k^n$. Entonces una representación paramétrica racional de $\mathbf{V}$ es un conjunto de funciones racionales $r_1,...,r_n \in k(t_1,...,t_m)$ tal que los puntos dados por:

$$\begin{aligned} x_1 &= r_1(t_1,...,t_m), \\ x_2 &= r_2(t_1,...,t_m), \\ &\vdots \\ x_n &= r_n(t_1,...,t_m) \end{aligned}$$

están en la variedad $\mathbf{V}$.

Cuando en una parametrización de una variedad tenemos polinomios en vez de funciones lo llamamos representación paramétrica polinómica de $\mathbf{V}$. Las ecuaciones originales $f_1 = ... = f_s = 0$ que definen la variedad $\mathbf{V}$ se llaman representación implícita de la variedad.

1.3. Ideales

Definición 1.6. Un subconjunto $I \subset k[x_1,...,x_n]$ es un ideal si cumple:

I) $0 \in I$

II) Si $f,g \in I$ entonces $f+g \in I$

III) Si $f \in I$ y $h \in k[x_1,...,x_n]$, entonces $hf \in I$

Definición 1.7. Dados los polinomios $f_1,...,f_s \in k[x_1,...,x_n]$ el conjunto

$$\langle f_1,...,f_s\rangle = \left\{ \sum_{t=1}^{s} h_t f_t : h_1,...,h_s \in k[x_1,...,x_n] \right\}$$

se denomina ideal generado por $f_1,...,f_s$.

Lema 1.4. *Si $f_1,...,f_s \in k[x_1,...,x_n]$ entonces $\langle f_1,...,f_s\rangle$ es un ideal de $k[x_1,...,x_n]$.*

Dados $f_1,...,f_s \in k[x_1,...,x_n]$, obtenemos el sistema de ecuaciones:

$$\begin{aligned} f_1 &= 0, \\ &\vdots \\ f_s &= 0. \end{aligned}$$

Desde estas ecuaciones podemos obtener otras. Por ejemplo, si multiplicamos la primera ecuación por $h_1 \in k[x_1,...x_n]$ la segunda por $h_2 \in k[x_1,...x_n]$ y así sucesivamente y luego sumamos todas obtenemos:

$$h_1 f_1 + h_2 f_2 + ... + h_s f_s = 0.$$

El lado izquierdo de esta ecuación es un elemento del ideal $\langle f_1,...,f_s\rangle$.

Decimos que I es un ideal finitamente generado si existe $f_1,...,f_s \in k[x_1,...,x_n]$ tal que $I = \langle f_1,...,f_s\rangle$, y decimos que $f_1,...,f_s$ es una base de I.

Proposición 1.5. *Si $f_1,...,f_s$ y $g_1,...,g_t$ son bases del mismo ideal en $k[x_1,...,x_n]$, tal que $\langle f_1,...,f_s\rangle = \langle g_1,...,g_t\rangle$, entonces $\mathbf{V}(f_1,...,f_s) = \mathbf{V}(g_1,...,g_t)$.*

Definición 1.8. Dada $\mathbf{V} \subset k^n$ una variedad afín, definimos el conjunto:

$$\mathbf{I}(\mathbf{V}) = \{f \in k[x_1,...,x_n] : f(a_1,...,a_n) = 0 \ \ \forall (a_1,...,a_n) \in \mathbf{V}\}.$$

Lema 1.6. *Si $\mathbf{V} \subset k^n$ es una variedad afín, entonces $\mathbf{I}(\mathbf{V}) \subset k[x_1,...,x_n]$ es un ideal, llamado ideal de $\mathbf{V}$.*

Lema 1.7. *Si $f_1,...,f_s \in k[x_1,...,x_n]$, entonces $\langle f_1,...,f_s\rangle \subset \mathbf{I}(\mathbf{V}(f_1,...,f_s))$, pero la igualdad no es siempre cierta.*

Para cuerpos arbitrarios la relación entre $\langle f_1,...,f_s\rangle$ y $\mathbf{I}(\mathbf{V}(f_1,...,f_s))$ puede ser sutil, sin embargo sobre un cuerpo algebraicamente cerrado como $\mathbb{C}$ hay una sencilla relación entre estos ideales.

Aunque, en los cuerpos, en general $\mathbf{I}(\mathbf{V}(f_1,...,f_s))$ puede no ser igual a $\langle f_1,...,f_s\rangle$. El ideal de una variedad siempre contiene suficiente información para poder determinar la variedad de forma unívoca.

Proposición 1.8. *Sean $\mathbf{V}$ y $\mathbf{W}$ dos variedades afines en k^n: Entonces:*

I) $\mathbf{V} \subset \mathbf{W}$ *si y solo si* $\mathbf{I}(\mathbf{V}) \supset \mathbf{I}(\mathbf{W})$.

II) $\mathbf{V} = \mathbf{W}$ *si y solo si* $\mathbf{I}(\mathbf{V}) = \mathbf{I}(\mathbf{W})$.

1.4. Polinomios de una variable

Definición 1.9. Dado un polinomio $f \in k[x]$ no nulo, tomamos:

$$f = a_0 x^m + a_1 x^{m-1} + ... + a_m$$

donde $a_t \in k$ y $a_0 \neq 0$, además el grado de f es m. Entonces decimos que el término principal de f es $a_0 x^m$ y lo denotamos $TP(f)$.

Así, dados dos polinomios no nulos f, g podemos decir que:

$$gr(f) \leq gr(g) \Leftrightarrow TP(f) \text{ divide } TP(g)$$

Proposición 1.9. ***El Algoritmo de la División****: Sea k un cuerpo y g un polinomio no nulo en $k[x]$. Entonces, cada $f \in k[x]$ puede escribirse como:*

$$f = qg + r$$

donde $q, r \in k[x]$ y también $r = 0$ o el $gr(r) < gr(g)$. Además, q y r son los únicos polinomios que cumplen estas propiedades.

Demostración. Describamos primero el algoritmo de la división en pseudocódigo:

INPUT: g, f
OUTPUT: q, r
    $q := 0; r := f$
WHILE: $r \neq 0$ **AND** $TP(g)$ divide $TP(r)$ **DO**
    $q := q + TP(r)/TP(g)$
    $r := r - (TP(r)/TP(g))g$

En cada paso del algoritmo vamos obteniendo nuevos valores de q y r hasta obtener sus valores finales.

Veamos que funciona: Primero notamos que $f = qg + r$ se cumple para los valores iniciales de q y r dados y que si lo redefinimos sigue siendo cierto ya que se cumple:

$$f = qg + r = (q + TP(r)/TP(f))g + (r - (TP(r)/TP(g))g)$$

El siguiente paso del algoritmo es comprobar que no se cumple $r \neq 0$ y que $TP(g)$ divide $TP(r)$ si es así produce una nueva q y r y repetimos hasta que deje de ser falso.

La clave es observar que $r - (TP(r)/TP(g))g$ es 0 o de menor grado que r. Pues, suponiendo que:

$$r = a_0x^m + ... + a_m,\ TP(r) = a_0x^m,$$

$$g = b_=X^K + ... + b_k,\ TP(g) = b_0x^k,$$

y suponemos que $m \geq k$, entonces:

$$r - (TP(r)/TP(g))g = (a_0x^m + ...) - \frac{a_0}{b_0}x^{m-k}(b_0x^k + ...)$$

y esto significa que el grado de r va bajando. Y como el grado es finito, concluimos que el algoritmo es finito.

Por último, tenemos que ver que q y r son únicos. Supongamos que $f = qg + r = q'g + r'$ donde r y r' poseen menor grado que g. Si $r \neq r'$ entonces $gr(r' - r) < gr(g)$, pero por otro lado

$$(q - q')g = r' - r$$

veíamos que $q - q' \neq 0$ y por tanto:

$$gr(r' - r) = gr((q - q')g) = gr(q - q') + gr(g) \geq gr(g).$$

Esto es una contradicción y por tanto $r = r'$ y también tenemos que $q = q'$. Así, obtenemos la unicidad. □

Corolario 1.10. *Si k es un cuerpo y $f \in k[x]$ es un polinomio no nulo, entonces el grado de f es como mucho el número de raíces de f en k.*

Corolario 1.11. *Si k es un cuerpo, entonces cada ideal de $k[x]$ puede escribirse como $\langle f \rangle$ para algún $f \in k[x]$. Además, f es único salvo multiplicación por constantes no nulas de k.*

Definición 1.10. Un Máximo Común Divisor de dos polinomios $f, g \in k[x]$ es un polinomio h tal que :

I) h divide a f y g.

II) Si p es otro polinomio que divide a f y g, entonces p divide a h.

Si h cumple esto se escribe $h = \mathbf{MCD}(f,g)$

Proposición 1.12. *Sea $f, g \in k[x]$. Entonces.*

I) $\mathbf{MCD}(f,g)$ *existe y es el único salvo multiplicación por constantes no nulas en k*

II) $\mathbf{MCD}(f,g)$ *es un generador del ideal $\langle f, g \rangle$*

III) *El* $\mathbf{MCD}(f,g)$ *se puede encontrar mediante un algoritmo.*

Demostración. Veamos este algoritmo. Tomamos la siguiente notación, $f, g \in k[x]$ donde $g \neq 0$ y escribimos $f = qg + r$ donde q y r son polinomios como los de la proposición 1.9. Cogemos r como el resto de f y g ($r = resto(f,g)$). Podemos emplear, por tanto el algoritmo de Euclides para encontrar el $\mathbf{MCD}(f,g)$:

INPUT: g, f
OUTPUT: h
$s := g$; $h := f$
WHILE: $s \neq 0$ **DO**:
$res := resto(h,s)$
$h := s$
$s := res$

Por el algoritmo de la división, llamando $f = qg + r$ podemos afirmar que:

$$\mathbf{MCD}(f,g) = \mathbf{MCD}(f - qg, g) = \mathbf{MCD}(r,g).$$

Notamos que el $gr(g) > gr(r)$ o $r = 0$. Si $r \neq 0$ repetimos el proceso para hacerlo todavía mas pequeño, aplicando el algoritmo de la división ahora a $g = q'r + r'$ y así llegamos a que:

$$\mathbf{MCD}(g,r) = \mathbf{MCD}(r,r').$$

Donde $gr(r) > gr(r')$ o $r' = 0$ y si no es así seguimos repitiendo.

Además podemos afirmar que este algoritmo termina porque el grado de s va disminuyendo, entonces en algún momento llegará a 0 y si eso pasa el $\mathbf{MCD}(f,g) = \mathbf{MCD}(h,0) = h$. □

Definición 1.11. El Máximo Común Divisor de los polinomios $f_1, ..., f_s \in k[x]$ es un polinomio h tal que :

I) h divide a $f_1, ..., f_s$.

II) Si p es otro polinomio que divide a $f_1, ..., f_s$ entonces p divide a h.

Entonces, se escribe $h = \mathbf{MCD}(f_1, ..., f_s)$.

Y al igual que hemos extendido la definición de **MCD** para dos polinomios a s polinomios podemos extender la proposición anterior añadiendo: si $s \geq 3$, entonces $\mathbf{MCD}(f_1, ..., f_s) = \mathbf{MCD}(f_1, \mathbf{MCD}(f_2, ..., f_s))$.

Capítulo 2

Bases de Gröbner

A partir de las bases de Gröbner intentamos dar respuesta a cuatro problemas:

1. Conseguir saber si los ideales están generados por un conjunto finito.
2. Resolver el problema de pertenencia a un ideal.
3. Encontrar las soluciones de un sistema de ecuaciones.
4. Buscar si es posible encontrar un sistema de ecuaciones a partir de las soluciones.

Ejemplo 1. Dado un ideal $I \subset k[x]$, es decir, $n = 1$, por el corolario 1.11 sabemos que $I = \langle g \rangle$ para algún $g \in k[x]$, es decir, están generados por un conjunto finito.

También es fácil encontrar los miembros de los ideales usando el algoritmo de la división,1.9: $f \in I = \langle g \rangle$ si y solo si al dividir f entre g se tiene que $r = 0$.

Ejemplo 2. Consideramos un sistema de ecuaciones:

$$\begin{aligned} a_{11}x_1 + ... + a_{1n}x_n + b_1 = 0, \\ \vdots \\ a_{m1}x_1 + ... + a_{mn}x_n + b_m = 0. \end{aligned}$$

Podemos reducir las filas a una matriz:

$$\begin{pmatrix} a_{11} & \cdots & a_{1n} & -b_1 \\ \vdots & & \vdots & \vdots \\ a_{m1} & \cdots & a_{mn} & -b_m \end{pmatrix}$$

Hacemos operaciones con las filas hasta obtener una matriz escalonada y luego podemos encontrar las soluciones sustituyendo los valores por las variables libres.

En algunos casos puede haber una solución o no haberla.

Ejemplo 3. Tomamos un n arbitrario y consideramos $\mathbf{V} \subset K^n$ parametrizada como:

$$\begin{aligned} x_1 = a_{11}t_1 + ... + a_{1m}t_m + b_1, \\ \vdots \\ x_n = a_{n1}t_1 + ... + a_{nm}t_m + b_n. \end{aligned}$$

$\mathbf{V}$ es un subespacio afín lineal de k^n ya que $\mathbf{V}$ es la imagen de $F : k^n \longrightarrow k^n$ definida por:

$$F(t_1, ..., t_m) = (a_{11}t_1 + ... + a_{1m}t_m + b_1, ..., a_{n1}t_1 + ... + a_{nm}t_m + b_n).$$

Buscamos un sistema de ecuaciones lineales cuya solución sea el conjunto de puntos de $\mathbf{V}$.

Escribimos las ecuaciones restando el término x_i a ambos lados y escribimos la matriz correspondiente. Operamos las filas para obtener una matriz escalonada con unos en la diagonal.

Cogiendo las últimas filas obtenemos ecuaciones que no dependen de los términos t_j que son las ecuaciones que definen $\mathbf{V}$.

Nuestro objetivo es extender los métodos de los ejemplos hechos en sistemas de grado 1 a cualquier grado y posteriormente ver si también es posible hacerse para sistemas diferenciales.

2.1. Orden monomial en $\mathbf{k[x_1,\ldots,x_n]}$.

Para el algoritmo de la división y la eliminación Gaussiana es muy importante el orden de los términos del polinomio. Para poder ordenarlos podemos describir el monomio $x^\alpha = x_1^{\alpha_1}\cdots x_n^{\alpha_n}$ como la tupla de sus exponentes $\alpha = (\alpha_1,...,\alpha_n)$, estableciendo una correspondencia entre los monomios en $k[x_1,...,x_n]$ y $\mathbb{Z}^n_{\geq 0}$. Además, cualquier orden $>$ establecido en $\mathbb{Z}^n_{\geq 0}$ dará un orden de los monomios y aunque hay muchas formas de ordenarlos nos va a interesar las que sean compatibles con la estructura algebraica de los anillos de polinomios.

Definición 2.1. Un orden monomial en $k[x_1,\ldots,x_n]$ es cualquier relación $>$ en $\mathbb{Z}^n_{\geq 0}$ o equivalentemente, una relación en el conjunto de monomios $x^\alpha, \alpha \in \mathbb{Z}^n_{\geq 0}$ cumpliendo:

(I) $>$ es total (o lineal) orden en $\mathbb{Z}^n_{\geq 0}$.

(II) Si $\alpha > \beta$ y $\gamma \in \mathbb{Z}^n_{\geq 0}$, entonces $\alpha+\gamma > \beta+\gamma$.

(III) $>$ es un buen orden en $\mathbb{Z}^n_{\geq 0}$. Esto quiere decir que cada subconjunto no vacío de $\mathbb{Z}^n_{\geq 0}$ tiene un elemento más pequeño.

Lema 2.1. *Una relación de orden $>$ en $\mathbb{Z}^n_{\geq 0}$ es un buen orden si y solo si cada secuencia estrictamente decreciente en $\mathbb{Z}^n_{\geq 0}$*

$$\alpha(1) > \alpha(2) > \alpha(3) > \ldots$$

termina eventualmente.

Definición 2.2. Orden lexicográfico. Sea $\alpha = (\alpha_1,\ldots,\alpha_n)$ y $\beta = (\beta_1,\ldots,\beta_n) \in \mathbb{Z}^n_{\geq 0}$. Decimos que $\alpha >_{lex} \beta$ si, en la diferencia vectorial $\alpha - \beta \in \mathbb{Z}^n$, el término más a la izquierda, no nulo, es positivo. Escribiremos $x^\alpha >_{lex} x^\beta$ si $\alpha >_{lex} \beta$.

Notar que el orden lexicográfico, es un orden monominal puesto que cumple la definición 2.1. Hay también que darse cuenta de que existen muchos ordenes lexicográficos porque dependen de como se ordenen las variables $x_1,\ldots,x_n$.

Podemos tener en cuenta el grado total del monomio y así obtenemos otra manera de ordenar los monomios.

Definición 2.3. Orden lexicográfico graduado. Dado $\alpha,\beta \in \mathbb{Z}^n_{\geq 0}$. Decimos $\alpha >_{grlex} \beta$ si:

$$|\alpha| = \sum_{i=1}^n \alpha_i > |\beta| = \sum_{i=1}^n \beta_i, \quad \text{o} \quad |\alpha| = |\beta| \ \text{ y } \ \alpha >_{lex} \beta$$

Escribiremos $x^\alpha >_{grlex} x^\beta$ si $\alpha >_{grlex} \beta$.

Definición 2.4. Orden lexicográfico graduado inverso. Dado $\alpha,\beta \in \mathbb{Z}^n_{\geq 0}$. Decimos $\alpha >_{invgrlex} \beta$ si:

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

y en $\alpha - \beta \in \mathbb{Z}^n$ el término más a la derecha no nulo es negativo.

Escribiremos $x^\alpha >_{invgrlex} x^\beta$ si $\alpha >_{invgrlex} \beta$.

Definición 2.5. Dado $f = \sum_\alpha a_\alpha x^\alpha$ un polinomio no nulo en $k[x_1, \dots x_n]$ y dado $>$ un orden monomial:

(I) El multigrado de f es: $multigr(f) = max(\alpha \in \mathbb{Z}^n_{\geq 0} \ : \ a_\alpha \neq 0)$

(II) El coeficiente principal de f es: $CL(f) = a_{multigr(f)} \in k$

(III) El monomio principal de f es: $ML(f) = x^{multigr(f)}$

(IV) El término principal de f es: $TL(f) = CL(f) \cdot ML(f)$

Ejemplo 4. Dado $f(x,y,z) = 2x + 3y^2z + x^2 + z^3$ se pretende dar el polinomio con sus términos ordenados por cada uno de los órdenes monomiales con $x > y > z$ indicando en cada caso su multigrado, su término principal y su monomio principal.

En los tres casos se escribe cada término del polinomio como un vector de 3 componentes, la primera corresponde al grado de la x, la segunda el de la y y la tercera el de la z.

$$x \rightarrowtail (1,0,0)$$

$$y^2z \rightarrowtail (0,2,1)$$

$$x^2 \rightarrowtail (2,0,0)$$

$$z^3 \rightarrowtail (0,0,3)$$

- **Orden lexicográfico**. En este orden debemos restar los vectores y sera mayor el que en esta resta tenga el término más a la izquierda, no nulo, positivo. Así, f ordenado por el orden lexicográfico es: $f(x,y,z) = x^2 + 2x + 3y^2z + z^3$, el multigrado es $(2,0,0)$, su término principal es x^2 que coincide con el monomio principal.

- **Orden lexicográfico graduado**. El término mayor es el que al realizar la suma de las componentes del vector sea mayor, y si dos son iguales el que sea mayor por orden lexicográfico. Por tanto, f ordenado por orden lexicográfico graduado es $f(x,y,z) = 3y^2z + z^3 + x^2 + 2x$ así, $multigr(f) = (0,2,1)$, $TL(f) = 3y^2z$ y $ML(f) = y^2z$

- **Orden lexicográfico graduado inverso**. El término mayor es el que al realizar la suma de las componentes del vector sea mayor y si dos son iguales el que al hacer la resta tenga el término más a la derecha no nulo negativo. De esta manera, el orden lexicográfico graduado inverso coincide con el orden lexicográfico graduado.

Lema 2.2. *Dado $f, g \in k[x_1, \dots, x_n]$ polinomios no nulos, entonces:*

(I) $multigr(fg) = multigr(f) + multigr(g)$.

(II) Si $f + g \neq 0$, entonces $multigr(f+g) \geq max(multigr(f), multigr(g))$.
Si además $multigr(f) \neq multigr(g)$ entonces la igualdad ocurre.

2.2. El algoritmo de la división en $\mathbf{k[x_1, \dots, x_n]}$.

Vamos a extender el algoritmo de la división 1.9, ya visto, a polinomios en $k[x_1, \dots, x_n]$ en vez de $k[x]$. En general el objetivo es dividir $f \in k[x_1, \dots, x_n]$ entre $f_1, \dots, f_s \in k[x_1, \dots, x_n]$, es decir, ver si podemos expresar f como:

$$f = a_1 f_1 + \dots + a_s f_s + r.$$

Donde $a_1, ..., a_n, r \in k[x_1, \dots, x_n]$, y para caracterizarlo necesitaremos los distintos órdenes lexicográficos.

Teorema 2.3. Algoritmo de la división en $\mathbf{k}[\mathbf{x_1},\ldots,\mathbf{x_n}]$. *Fijado un orden monomial $>$ en $\mathbb{Z}^n_{\geq 0}$, y dado $F=(f_1,\ldots,f_s)$ una tupla ordenada de polinomios en $k[x_1,\ldots,x_n]$. Entonces cada $f\in k[x_1,\ldots,x_n]$ puede escribirse como:*

$$f=a_1f_1+\ldots+a_sf_s+r.$$

Donde $a,r\in k[x_1,\ldots,x_n]$ y $r=0$ o una combinación lineal de monomios con coeficientes en k no divisibles por $TL(f_1),\ldots,TL(f_s)$. Llamaremos r el resto de dividir f entre F, además si $a_if_i\neq 0$ tendremos que $multigr(f)\geq multigr(a_if_i)$.

Demostración. Veamos la existencia de $a_1,\ldots,a_s$ y r dando un algoritmo para su construcción y viendo que funciona correctamente.

INPUT: $f_1,\ldots,f_s,f$
OUTPUT: $a_1,\ldots,a_s,r$
    $a_1:=0;\ldots a_:=0;r:=0$
    $p:=f$
WHILE $p\neq 0$ **DO**:
    $i:=1$
    ocurredivision:=false
    WHILE $i\leq s$ **AND** ocurredivision= false **DO**:
        IF $TL(f_i)$ divide $TL(p)$ **THEN**:
            $a_i:=a_i+TL(p)/TL(f_i)$
            $p:=p-(TL(p)/TL(f_i))f_i$
            ocurredivision:=true
        ELSE:
            $i:=i+1$
    IF ocurredivision=false **THEN**:
        $r:=r+TL(p)$
        $p:=p-TL(p)$

Observamos que en el bucle pueden pasar dos cosas:

- (División) Si $TL(f_i)$ divide a $TL(p)$ el algoritmo actúa como si hubiera una variable.
- (Resto) Si $TL(f_i)$ no divide a $TL(p)$ entonces al algoritmo añade el término principal de p al resto r.

Veamos que el algoritmo funciona, primero comprobamos que

$$f=a_1f_1+\ldots+a_sf_s+p+r \tag{2.1}$$

se verifica en cada paso.

Para los valores iniciales es claro. Suponemos que $TL(f_i)$ divide a $TL(p)$ y la igualdad:

$$a_if_i+p=(a_i+TL(p)/TL(f_i))f_i+(p-(TL(p)/TL(f_i))f_i).$$

Vemos que a_if_i+p no cambia, entonces en este caso 2.1 se sigue cumpliendo.

Por otro lado el siguiente paso tenemos que p y r cambian pero $p+r$ no cambia

$$p+r=(p-TL(p))+(r+TL(p)),$$

entonces igual que antes se sigue cumpliendo 2.1.

El algoritmo se empieza a detener cuando $p=0$ en esta situación 2.1 se transforma en:

$$f=a_1f_1+\ldots+a_sf_s+r.$$

Los términos se añaden a r sólo cuando no son divisibles por ningún $TL(f_i)$, $a_1,\ldots,a_s$ y r cumplen las propiedades deseadas cuando el algoritmo termina.

Por último, necesitamos ver que el algoritmo siempre termina. Para ello veamos que en cada paso p se redefine y su multigrado disminuye o se vuelve 0. Suponemos que durante la división p se redefine como:

$$p' = p - \frac{TL(p)}{TL(f_i)} f_i$$

Así;

$$TL\left(\frac{TL(p)}{TL(f_i)} f_i\right) = \frac{TL(p)}{TL(f_i)} TL(f_i) = TL(p)$$

por tanto p y $\frac{TL(p)}{TL(f_i)} f_i$ tiene el mismo término principal. Así, su diferencia p' tiene que tener multigrado menor y si el algoritmo no termina nunca tendríamos un serie decreciente de multigrados y por las propiedades de lo ordenes monomiales esto no puede ocurrir.

Queda pendiente estudiar la relación entre el $multigr(f)$ y el $multigr(a_i f_i)$.

Cada término de a_i es de la forma $TL(p)/TL(f_i)$ para algún valor de la variable p. El algoritmo empieza por $p = f$ y como hemos visto el multigrado de p disminuye, por eso vemos que $TL(p) \leq TL(f)$ entonces tenemos que $multigr(a_i f_i) \leq multigr(f)$ cuando $a_i f_i \neq 0$. □

Ejemplo 5. Sea $f = x^7y^2 + x^3y^2 - y + 1$ se quiere dividir entre $F = [f_1 = xy^2 - x, f_2 = x - y^3]$ usando el orden lexicográfico graduado con $x > y$.

El $TL(f_1) = xy^2$ divide al $TL(f) = x^7y^2$ entonces podemos realizar el primer paso del algoritmo obteniendo como cociente x^6 y resto $r_1 = x^7 - x^3y^2 - y + 1$. Ahora el término principal de r_1 es x^7 que no es dividido ni por $TL(f_1)$ ni por $TL(f_2) = -y^3$ por tanto x^7 es parte del resto de la división y nos queda $-x^3y^2 - y + 1$ para continuar el algoritmo, repitiendo estos pasos hasta obtener un resto donde ninguno de sus términos pueda ser dividido por el termino principal de f_1 ni de f_2, se obtiene, en este ejemplo, que el resto final es $r = x^7 + x^3 - y + 1$ y podemos escribir $f = (x^6 + x^2)f_1 + 0f_2 + r$.

Podemos apreciar algunas propiedades del algoritmo de la división, como que el resto no está determinado de manera única, pues si se escoge otro orden monomial puede cambiar, por ejemplo si ahora dividimos f entre F pero con orden lexicográfico con $x > y$ obtenemos que el resto es $2y^3 - y + 1$, por eso es importante fijar un orden monomial. Incluso con el mismo orden monomial el resto depende de la forma de ordenar la tupla de divisores. Por ejemplo si dividimos f entre $F_1 = [f_2, f_1]$ con orden lexicográfico obtenemos que el resto es $y^{23} + y^{11} - y + 1$ que es distinto al anterior.

Otra propiedad interesante es que si después de dividir f entre $F = (f_1,\ldots,f_s)$ se obtiene $r = 0$ entonces quiere decir que $f \in \langle f_1,\ldots,f_s\rangle$, de hecho, $r = 0$ es condición suficiente para ser miembro de un ideal, pero no es condición necesaria.

Por ejemplo, si dividimos $f = x + z$ entre $F = [xy - z, xy + x]$ obtenemos que el resto es $x + z$ y aunque al aplicar el algoritmo de la división el resto no da 0, f es combinación de los elementos de F pues $f = f_2 - f_1$, lo que implica que $f \in F$.

2.3. Ideales monomiales y lema de Dickson

Definición 2.6. Un ideal $I \subset k[x_1,\ldots,x_n]$ es un ideal monomial si hay un subconjunto $A \subset \mathbb{Z}^n_{\geq 0}$ tal que I contiene a todos los polinomios de la forma $\sum_{\alpha\in A} h_\alpha x^\alpha$ donde $h_\alpha \in k[x_1,...,x_n]$, en este caso

$$I = \langle x^\alpha : \alpha \in A\rangle.$$

Lema 2.4. *Sea $I = \langle x^\alpha : \alpha \in A\rangle$ un ideal monomial. Entonces x^β pertenece a I si y solo si x^β es divisible por x^α para algún $\alpha \in A$.*

Lema 2.5. *Sea I un ideal monomial y sea $f \in k[x_1,\dots,x_n]$, entonces son equivalentes:*

(I) $f \in I$.

(II) Cada término de f está en I.

(III) f es una k-combinación lineal de monomios de I.

Corolario 2.6. *Dos ideales monomiales son el mismo si y solo si contienen los mismos monomios.*

Teorema 2.7. ***Lema de Dickson.*** *Un ideal monomial $I = \langle x^\alpha : \alpha \in A\rangle \subset k[x_1,\dots,x_n]$ se puede escribir de la forma $I = \langle x^{\alpha(1)},\dots,x^{\alpha(s)}\rangle$ donde $\alpha(1),\dots,\alpha(s) \in A$. En particular, I tiene una base finita.*

Demostración. Vamos a verlo por inducción sobre n.

- Si $n = 1$: Entonces I está generado por los monomios x_1^α donde $\alpha \in A \subset \mathbb{Z}_{\geq 0}$. Sea β el elemento más pequeño de $A \subset \mathbb{Z}_{\geq 0}$. Entonces $\beta \leq \alpha$ para todo $\alpha \in A$, así x_1^β divide a todos los otros generadores x_1^α. Así se sigue que $I = \langle x_1^\beta\rangle$.

- Tomando $n > 1$ y que el teorema es cierto para $n-1$. Escribiendo las variables como $x_1,\dots,x_{n-1},y$, así, los monomios en $k[x_1,\dots,x_n,y]$ se pueden escribir como $x^\alpha y^m$ donde $\alpha = (\alpha_1,\dots,\alpha_{n-1}) \in \mathbb{Z}_{\geq 0}^{n-1}$ y $m \in \mathbf{Z}_{\geq 0}$.

  Suponemos que $I \subset k[x_1,\dots,x_{n-1},y]$ es un ideal monomial. Para encontrar los generadores de I, sea J un ideal en $k[x_1,\dots,x_{n-1}]$ generado por los monomios x^α para los cuales $x^\alpha y^m \in I$ para algún $m \geq 0$. Por la hipótesis de inducción tenemos que muchos de los monomios x^α generan J, es decir, tenemos que $J = \langle x^{\alpha(1)},\dots,x^{\alpha(s)}\rangle$.

  Para cada $1 \leq i \leq s$ la definición de J nos dice que $x^{\alpha(i)}y^{m_i} \in I$ para algún $m_i \geq 0$. Sea m el mayor de los m_i, entonces para cada $k \in [0,...,m-1]$ consideramos el ideal $J_k \subset k[x_1,\dots,x_{n_1}]$ generado por los monomios x^β tal que $x^\beta y^k \in I$. Podemos pensar en J_k como una parte de I generada por monomios que contienen a y y exactamente k potencias. Volviendo a usar la hipótesis de inducción vemos que J_k está generado por un conjunto finito de monomios, es decir $J_k = \langle x^{\alpha_\lambda(1)},\dots,x^{\alpha_\lambda(s_\lambda)}\rangle$.

  Vemos que I está generado por alguno de los siguientes monomios:

$$\begin{gathered} J : x^{\alpha(1)}y^m,\dots,x^{\alpha(s)}y^m, \\ J_0 : x^{\alpha_0(1)},\dots,x^{\alpha_0(s_0)}, \\ J_1 : x^{\alpha_1(1)}y,\dots,x^{\alpha_1(s_1)}y, \\ \vdots \\ J_{m-1} : x^{\alpha_{m-1}(1)}y^{m-1},\dots,x^{\alpha_{m-1}(s_{m-1})}y^{m-1}. \end{gathered}$$

  Notamos que cada monomio en I es divisible por uno de la lista, pues sea $x^\alpha y^p \in I$, si $p \geq m$, es divisible por algún $x^{\alpha(i)}y^m$ de la constitución de J. En caso contrario ($p \leq m-1$), es divisible por algún $x^{\alpha_p(j)}y^p$ de la construcción de J_p. Así por el lema 2.5 tenemos que los monomios anteriores generan el mismo ideal que I y por el corolario 2.6 apreciamos que tienen que ser los mismos.

Para terminar la demostración debemos observar si el conjunto finito de los generadores se puede escoger del conjunto de los generadores del ideal. Si escribimos las variables como $x_1,\dots,x_n$ nuestro ideal monomial es $I = \langle x^\alpha : \alpha \in A\rangle \subset k[x_1,\dots,x_n]$. Necesitamos ver que I está generado por un número finito de x^α donde $\alpha \in A$. Anteriormente hemos visto que $I = \langle x^{\beta(1)},\dots,x^{\beta(s)}\rangle$ para algún monomio $x^{\beta(t)} \in I$ y mientras pasa esto, por el lema 2.4, sabemos que $x^{\beta(t)}$ es divisible por $x^{\alpha(t)}$ para algún $\alpha(i) \in A$. Entonces es fácil ver que $I = \langle x^{\alpha(1)},\dots,x^{\alpha(s)}\rangle$. □

Corolario 2.8. *Sea $>$ una relación en $\mathbb{Z}_{\geq 0}^n$ tal que :*

I) $>$ es un orden total en $\mathbb{Z}_{\geq 0}^n$.

II) Si $\alpha > \beta$ y $\gamma \in \mathbb{Z}_{\geq 0}^n$, entonces $\alpha + \gamma > \beta + \gamma$.

Entonces $>$ es un buen orden si y solo si $\alpha \geq 0$ para todo $\alpha \in \mathbb{Z}_{\geq 0}^n$.

2.4. El teorema de las bases de Hilbert y bases de Gröbner.

Definición 2.7. Dado $I \subset k[x_1,\ldots,x_n]$ un ideal distinto de $\{0\}$.

(I) Llamamos $TL(I)$ al conjunto de los términos principal de los elementos de I. Así:

$$TL(I) = \{cx^\alpha : existe\ f \in I\ con\ TL(f) = cx^\alpha\}.$$

(II) Denotamos $\langle TL(I)\rangle$ al ideal generado por los elementos de $TL(I)$.

Si damos un conjunto finito generador de I, $I = \langle f_1,\ldots,f_s\rangle$, entonces $\langle TL(f_1),\ldots,TL(f_s)\rangle$ y $\langle TL(I)\rangle$ pueden ser ideales distintos. Es cierto que $TL(f_t) \in TL(I) \subset \langle TL(I)\rangle$ por definición lo que implica, $\langle TL(f_1),\ldots,TL(f_s)\rangle \subset \langle TL(I)\rangle$. Sin embargo, $\langle TL(I)\rangle$ puede ser estrictamente mayor.

Proposición 2.9. *Dado un ideal $I \subset k[x_1,\ldots,x_n]$:*

(I) $\langle TL(I)\rangle$ es un ideal monomial.

(II) Hay $g_1,\ldots,g_s \in I$ tales que $\langle TL(I)\rangle = \langle TL(g_1),\ldots,TL(g_t)\rangle$.

Teorema 2.10. ***Teorema de Bases de Hilbert***. *Cada ideal $I \subset k[x_1,\ldots,x_n]$ tiene un conjunto finito generador. Este es $I = \langle g_1,\ldots,g_s\rangle$ para algún $g_1,\ldots,g_s \in I$.*

Definición 2.8. Fijado un orden monomial. Un conjunto finito $G = \{g_1,\ldots,g_s\}$ de un ideal I se llama base de Gröbner si:

$$\langle TL(g_1),\ldots,TL(g_t)\rangle = \langle TL(I)\rangle.$$

Corolario 2.11. *Fijado un orden monomial, cada ideal $I \subset k[x_1,\ldots,x_n]$ distinto de $\{0\}$ tiene una base de Gröbner. Además, cualquier base de Gröbner de I es base de I.*

Algunas consecuencias del Teorema de las bases de Hilbert son la condición de las cadena ascendente de ideales (ACC) y las variedades afines son determinadas por ideales.

Teorema 2.12. ***Condición de las cadenas ascendentes***. *Dado $I_1 \subset I_2 \subset I_3 \subset \ldots$ un cadena ascendente de ideales en $k[x_1\ldots,x_n]$. Entonces existe un $N \geq 1$ tal que:*

$$I_N = I_{N+1} = I_{N+2} = \ldots..$$

Definición 2.9. Dado $I \subset k[x_1,\ldots,x_n]$ un ideal. Denotamos por $\mathbf{V}(I)$ al conjunto:

$$\mathbf{V}(I) = \{(a_1,\ldots,a_n) \in k^n : f(a_1,\ldots,a_n) = 0\ \ \forall\, f \in I\}$$

El conjunto $\mathbf{V}(I)$ puede estar definido por un conjunto finito de ecuaciones polinómicas aunque el ideal no nulo I contenga infinitos polinomios diferentes.

Proposición 2.13. *$\mathbf{V}(I)$ es una variedad afín. En particular, si $I = \langle f_1,\ldots,f_s\rangle$ entonces $\mathbf{V}(I) = \mathbf{V}(f_1,\ldots,f_s)$.*

2.5. Propiedades de las Bases de Gröbner.

Hemos visto en el ejemplo 5 que el resto no tiene porque ser único si se cambia el orden de los divisores. Sin embargo, si los divisores forman una base de Gröbner esto no ocurre.

Proposición 2.14. *Sea $G = g_1,\ldots,g_n$ una base de Gröbner para un ideal $I \subset k[x_1,\ldots x_n]$ y sea $f \in k[x_1,\ldots,x_n]$. Entonces hay un único $r \in k[x_1,...,x_n]$ con las siguientes propiedades.*

I) Ningún término de r es divisible por $LT(g_1),...,LT(g_n)$.

II) *Hay un $g \in I$ tal que $f = g + r$.*

En particular, r es el resto de la división de f entre G sin importar el orden de los elementos de G cuando haces el algoritmo de la división.

Corolario 2.15. *Sea $G = g_1, \ldots, g_n$ una base de Gröbner para un ideal $I \subset k[x_1, \ldots x_n]$ y sea $f \in k[x_1, \ldots, x_n]$. Entonces $f \in I$ si y solo si el resto de la división de f entre G es cero.*

Definición 2.10. Llamamos $\overline{f}^F$ al resto en la división de f entre la tupla ordenada $F = (f_1, ..., f_s)$. Si F es una base de Gröbner para $\langle f_1, ..., f_n \rangle$ entonces podemos considerar F como un conjunto.

Definición 2.11. Sea $f, g \in k[x_1, ..., x_n]$ polinomios no nulos.

I) Si $multigr(f) = \alpha$ y $multigr(g) = \beta$ entonces $\gamma = (\gamma_1, ..., \gamma_n)$, donde $\gamma_i = max(\alpha_i, \beta_i)$ para cada i. Llamamos x^γ al mínimo común múltiplo de $ML(f)$ y $ML(g)$ y escribimos $x^\gamma = MCM(ML(f), ML(g))$.

II) El S-polinomio de f y g es la combinación:

$$S(f,g) = \frac{x^\gamma}{TL(f)} f - \frac{x^\gamma}{TL(g)} g. \tag{2.2}$$

Lema 2.16. *Suponemos que tenemos la suma $\sum_{i=1}^s c_i f_i$ donde $c_i \in k$ y $multigr(f_i) = \delta \in \mathbb{Z}^n_{\geq 0}$ para todo i. Si $multigr(\sum_{i=1}^s c_i f_i) < \delta$ entonces $\sum_{i=1}^s c_i f_i$ es una combinación lineal con coeficientes en k, del S-polinomio $S(f_j, f_k)$ para $1 \geq j, k, \geq s$. Además, cada $S(f_j, f_k)$ tiene multigrado $< \delta$.*

Teorema 2.17. *Sea I un ideal de polinomios. Entonces la base $G = \{g_1, ..., g_n\}$ de I es una base de Gröbner de I si y solo si para todo par $i \neq j$, el resto de la división de $S(g_i, g_j)$ entre G es cero.*

Este teorema es llamado el criterio de Buchberger.

2.6. Algoritmo de Buchberger.

Este algoritmo, parte desde el criterio de Buchberger y nos va a permitir generar bases de Gröbner.

Teorema 2.18. *Dado $I = \langle f_1, \ldots, f_s \rangle \neq 0$ un ideal de polinomios, entonces una base de Gröbner para I puede ser generada en un número finito de pasos con el siguiente algoritmo:*

INPUT: $F = (f_1, \ldots, f_s)$
OUTPUT: Una base de Gröbner $G = C(g_1, \ldots, g_t)$ para I con $F \subset G$
  $G := F$ **REPEAT**
  $G' := G$
FOR cada par $p, q, p \neq q$ en G' **DO**:
    $S := \overline{S(p,q)}^{G'}$
    IF $S \neq 0$ **THEN** $G := G \cup \{S\}$
UNTIL $G = G'$

Demostración. Si, $G = \{g_1, ..., g_s\}$ entonces como tomamos como notación $\langle G \rangle = \langle g_1, ..., g_s \rangle$ y $\langle LT(G) \rangle = \langle LT(g_1), ..., LT(g_t) \rangle$. Veamos que se tiene $G \subset I$ en cada paso del algoritmo.

Sabemos que inicialmente es cierto, en el siguiente paso añadimos a G el resto $S = \overline{S(p,q)}^{G'}$ para $p, q \in G$. Si $G \subset I$ entonces p, q y por tanto $S(p,q)$ están en I y como estamos dividiendo por $G' \subset I$ tenemos que $G \cup \{S\} \subset I$. También vemos que G contiene la base de I dada por F, lo que quiere decir que G es ahora base de I.

El algoritmo termina cuando $G = G'$ que quiere decir que $\overline{S(p,q)}^{G'} = 0$ para todo $p, q \in G$, y por el teorema 2.17 sabemos que G es base de Gröbner de $\langle G \rangle = I$.

Nos queda ver que el algoritmo termina. Necesitamos considerar qué pasa después de cada paso del bucle. El conjunto G es G' con el resto no nulo obtenido de la división del S-polinomio entre los elementos de G'. Suponemos que r es el resto no nulo que se añade a G y por tanto el $TL(r)$ no es divisible por ningún termino principal de los elementos de G', y así, $TL(r) \notin \langle TL(G')\rangle$, pero todavía $TL(r) \notin \langle TL(G)\rangle$ entonces $\langle TL(G')\rangle \subset \langle TL(G)\rangle$ ya que $G' \subset G$. Además si $G' \neq G$, es claro el contenido es estricto.

Por lo que acabamos de ver el ideal $\langle TL(G')\rangle$ y sus sucesivas iteraciones del bucle son una cadena ascendente de ideales en $k[x_1,...,x_n]$. Así, por el teorema 2.12 de la condición de las cadenas ascendentes, sabemos que hay un número finito de iteraciones y la cadena se estabiliza, por eso en algún momento $\langle TL(G)\rangle = \langle TL(G')\rangle$ y esto implicara que $G' = G$, por tanto tenemos que el algoritmo termina en un número finito de pasos.

□

Así tenemos el criterio de Buchberger (2.17) y el algoritmo de Buchberger (2.18) que nos permiten encontrar y dar bases de Gröbner. Pero usando estos dos teoremas a menudo obtenemos bases más grandes de lo necesario y podríamos eliminar algunos generadores redundantes.

Lema 2.19. *Sea G una base de Gröbner para un ideal polinómico I. Sea $p \in G$ un polinomio tal que $TL(p) \in \langle TL(G-\{p\})\rangle$. Entonces $G-\{p\}$ es también una base para I.*

Definición 2.12. Una base de Gröbner minimal para el ideal de polinomios I es una base de Gröbner G de I tal que :

I) $CL(p) = 1$ para todo $p \in G$.

II) Para todo $p \in G$, $TL(p) \notin \langle TL(G-\{p\})\rangle$.

Definición 2.13. Una base de Gröbner reducida para un ideal de polinomios I es una base de Gröbner G de I tal que:

I) $CL(p) = 1$ para todo $p \in G$.

II) Para todo $p \in G$, ningún monomio de p está en $\langle TL(G-\{p\})\rangle$.

Proposición 2.20. *Sea $I \neq \{0\}$ un ideal de polinomios. Entonces, para un orden monomial dado, I tiene una única base de Gröbner reducida.*

Otra consecuencia es que tenemos un algoritmo para ver cuando dos conjuntos de polinomios generan el mismo ideal.

Ejemplo 6. Sean $f_1 = x^2y - z$ y $f_2 = xy - 1$ dos polinomios sobre $\mathbb{Q}[x,y,z]$, $F = [f_1, f_2]$. Estos polinomios generan un ideal I. Fijemos el orden monomial lexicográfico con $x > y > z$. Queremos ver si F es una Base de Gröbner, si no lo es calcularla y obtener su base de Gröbner minimal.

Lo primero que hay que hacer es calcular el S-polinomio utilizando la fórmula 2.2, con ella obtenemos que es $S = -z + x$. El siguiente paso es realizar el algoritmo de la división, dividiendo S entre F, tras hacerlo obtenemos que el resto es $r = -z + x$ que es distinto de 0 por tanto por 2.17 F no es una base de Gröbner.

Para construir una base de Gröbner debemos aplicar el algoritmo de Buchberger, que consiste en añadir el resto de la división de cada uno de los posibles S-polinomios a la base inicial hasta que todos los restos sean cero. Es decir, ahora $F' = [f_1, f_2, r]$, calculamos todos los posibles S-polinomios de F'. Sus restos son siempre cero, lo que prueba que $F' = [x^2y - z, xy - 1, -z + x]$ es una base de Gröbner y que la base de Gröbner minimal, es decir quitando los elementos que sean combinación de los demás, es $F_m = [xy - 1, z - x]$.

Capítulo 3

Anillos de polinomios diferenciales

Una vez vistas las bases de Gröbner para ideales de polinomios algebraicos intentamos encontrar algo similar en anillos de polinomios diferenciales. Para ello, será preciso introducir una serie de conceptos y observaciones.

Definición 3.1. Un anillo R se dice que es un anillo diferencial si existe un operador de R a R, $d : R \to R$ tal que $\forall \alpha, \beta \in R$:

- d es lineal, es decir, $d(\alpha + \beta) = d(\alpha) + d(\beta)$
- $d(\alpha\beta) = d(\alpha)\beta + \alpha d(\beta)$.

Definición 3.2. Un subconjunto I de un anillo diferencial R es un ideal diferencial si es un ideal algebraico de R y además es cerrado para la diferencial, es decir, $d(I) \subset I$.

Si S es un subconjunto de R e I es el menor ideal diferencial de R conteniendo a S. Se dice que S es un sistema generador de I, si $S = \{f_1, ..., f_n\}$ entonces se denota $I = [f_1, ..f_n]$.

Pero tenemos que tener en cuenta que si R es un anillo algebraico se puede considerar el ideal algebraico I generado por S pero este será distinto al ideal diferencial.

Ejemplo 7. Tomamos k un cuerpo y $R = k[x_0, x_1, x_2, ...]$ definamos $d : R \to R$ como $d(x_i) = x_{i+1}$. Entonces el anillo R es un anillo diferencial, que podemos identificar con $k[x, dx, d^{(2)}x, ..., d^{(n)}x]$. Lo denotaremos por $k\{x\}$ y representa el anillo de polinomios diferenciales de x sobre k. Si añadimos más variables tenemos $k\{x_1, ..., x_n\}$ el anillo diferencial de polinomios en $x_1, ..., x_n$ sobre k.

Definición 3.3. Sea M el conjunto de todos los monomios en el anillo $R = k\{x_1, ..., x_n\}$, consideremos la aplicación, llamada peso de los monomios de $k\{x_1, ..., x_n\}$, $\omega : M \to R$ tal que:

- $\omega(x_i) = m_i$ con $m_i > 0$ para $i = 1, ..., n$
- $\omega(d^{(k)}x_i) = k + m_i$ para cualquier entero $k > 0$ y $i = 1, ..., n$
- Para cualquier monomio $m \in M$, $\omega(m) = \sum_i \omega(f_i)$ donde el rango de los f_i sobre los factores de m contiene una única variable o una derivada.

El peso de un polinomio diferencial es el máximo peso de sus monomios (entendiendo monomio como producto de variables y diferenciales de las variables).

Definición 3.4. Un polinomio diferencial se dice cuasi-homogéneo si sus monomios tienen todos el mismo peso respecto a las mismas funciones peso.

Cualquier polinomio se puede escribir de manera única como suma de polinomios cuasi-homogéneos, que se llaman componentes cuasi-homogéneas. A la componente de mayor peso se le llama componente cuasi-homogénea principal.

Definición 3.5. Un ideal diferencial I se dice cuasi-homogéneo si todo polinomio diferencial f de I tiene sus componentes cuasi-homogéneas en I, es decir, si $f \in I$ tal que $f = \sum f_d \Rightarrow f_d \in I$.

Con estas definiciones tenemos las siguientes propiedades:

- $\omega(fg) = \omega(f) + \omega(g)\ \forall f,g \in R = k\{x_1,...,x_n\}$
- $\omega(d(f)) = \omega(f) + 1\ \forall f,g \in R = k\{x_1,...,x_n\}$, y por tanto $\omega(d^{(k)}(f)) = \omega(f) + k\ \forall f,g \in R$
- El conjunto S_j de producto de potencias diferenciales de peso j es finito para cualquier j.

Proposición 3.1. *Un ideal diferencial I en un anillo de polinomios diferenciales con coeficientes constantes es cuasi-homogéneo si y solo si tiene un sistema generador cuasi-homogéneo.*

Tenemos que tener en cuenta que si R es un anillo diferencial necesitamos que $\mathbb{Q}$ esté contenido en R y que los ideales sean radicales para que $k\{x\}$ sea un anillo diferencial.

3.1. H-Bases de ideales

Con estas definiciones podemos definir las H-bases de ideales diferenciales que nos permitirán hacer una construcción similar a lo obtenido con las bases de Gröbner con lo que podremos simplificar sistemas de ecuaciones diferenciales y así poder resolverlos más fácilmente.

Definición 3.6. Sea I un ideal diferencial en $k\{x_1,...,x_n\}$, $H(I)$ es un ideal algebraico generado por el conjunto de todas las componentes cuasi-homogéneas principales de los polinomios diferenciales en I.

Así, un polinomio diferencial f está en $H(I)$ si todas sus componentes cuasi-homogéneas están en $H(I)$. Necesitaremos estar en el cuerpo de constantes para que $H(I)$ sea un ideal diferencial, pues de no estar las derivadas y extracción de componentes cuasi-homogéneas principales no se puede calcular en general. Además podemos definir $H(I)_i = H(I) \cap A_i$ donde A_i que es un k-espacio vectorial de polinomios cuasi-homogéneos de peso i.

Notamos que dada una base S para un ideal diferencial I consideramos el ideal $H(S)$ generado por las componentes cuasi-homogéneas principales de los polinomios del conjunto

$$\overline{S} = \{g : g = d^k s \ con\ s \in S\ k \in \mathbb{N} \cup \{o\}\},$$

entonces generalmente $H(S) \subset H(I)$.

Definición 3.7. Una base S de un ideal diferencial I se llama H-base si $H(S) = H(I)$.

Al estar en el cuerpo de las contantes $H(S)$ también es un ideal diferencial.

Veamos ahora un procedimiento para conseguir una H-base, empezando por un conjunto finito de generadores. Este proceso consiste en determinar para cualquier peso k el conjunto $H(S)_k$ y comprobar cualquier relación lineal entre los elementos de cada conjunto. Estas relaciones proporcionan nuevos generadores que tendrán menor peso y se podrán añadir a la base S y así podremos completar hasta obtener una H-base.

Este proceso es similar al visto para obtener una base de Gröbner y aunque tiene una serie de inconvenientes, como la restricción de los ideales mencionada anteriormente o que el cuerpo ha de ser el de las contantes, nos proporciona información útil sobre el ideal.

Sea S el conjunto de polinomios diferenciales tales que para cualquier entero fijo k hay un numero finito de elementos en S cuyo pero es k. Es posible introducir una variable por un procedimiento de reescritura para polinomios diferenciales usando elementos de S.

Sea f un polinomio diferencial cuasi-homogéneo. Suponemos $f_1,...f_p$ son elementos de S con peso $\omega(f)$: sus componentes cuasi-homogéneas principales generan un subespacio V del espacio vectorial W

de los polinomios cuasi-homogéneos de peso $\omega(f)$. Como podemos calcular el cuerpo de los coeficientes es posible encontrar $h_1(f)$ y $h_2(f)$ tales que se tiene $h(f) = h_1(f) + h_2(f)$ con $h_1(f) \in V$ y $h_2(f)$ en el subespacio complementario ortogonal a V. Es decir, las componentes cuasi-homogéneas principales de f (como f es cuasi-homogéneo $h(f) = f$) se pueden ponen como suma de las componentes principales de f que están en V y las que están en su complementario ortogonal.

Además se puede calcular los elementos $a_i \in k$ tales que:

$$h_1(f) = a_1 h(f_1) + ... + a_p h(f_p).$$

Con estos elementos que hemos encontrado podemos calcular la reducción de f módulo S que será:

$$\tilde{f} = f - (a_1 f_1 + ... + a_p f_p).$$

Esta relación de reducción la denotamos $f \to^S \tilde{f}$.

El proceso de reducción puede generalizarse para cualquier polinomio diferencial g sin necesidad de ser cuasi-homogéneo, pues sabemos que pueden ponerse como sumas de componentes cuasi-homogéneas, por tanto bastaría aplicarlo a cada una de sus componentes.

Proposición 3.2. *Sea I un ideal diferencial S una H-base de I y f un polinomio diferencial. Entonces f está en I si y sólo si alguna cadena maximal de reducción con respecto a S termina en 0.*

Capítulo 4

Ejemplo

En este capítulo aplicaremos el método explicado en el capítulo anterior a un caso particular para poder mostrar el procedimiento general que se encuentra implementado en el Anexo (ver pág. 29).

Consistirá en transformar un sistema de ecuaciones diferenciales polinómicas con una difícil solución en un sistema sencillo del cual la solución es conocida o fácil de obtener.

Para ello, partimos del siguiente sistema:

$$\begin{cases} y(t) + \frac{dx(t)}{dt} - \left(\frac{dx(t)}{dt} - \frac{dy^2(t)}{d^2t}\right)^{20} x(t)^{12} y(t)^{10} = 0 \\ \frac{dy(t)}{dt} - x(t) = 0. \end{cases}$$

Con estas dos ecuaciones podemos denotar $f_1 = y + \dot{x} - (\dot{x} - \ddot{y})^{20} x^{12} y^{10}$ y $f_2 = \dot{y} - x$ y así tomamos la base inicial $F = [f_1, f_2]$ donde f_1 y f_2 están en $\mathbb{C}\{x, y\}$ y fijamos el orden lexicográfico con $x > y$.

La manera de aplicar el método de reducción es análoga al algoritmo de Buchberger (2.18), así pues, comenzamos construyendo un *S*-polinomio a partir de estos dos polinomios diferenciales.

Como en el ejemplo 7 debemos entender $\dot{x}, \ddot{x}, ..., \dot{y}, \ddot{y}, ...$ como nuevas variables. Definiremos el peso de x y de y como 1 y así, podemos definir pesos al resto de variables de modo que $\omega(x_{i+1}) = \omega(x_i) + 1$ y análogamente para las y_i. Esta diferencia de peso nos afectará a la hora de encontrar el término principal pues dependiendo del orden fijado el peso afectara a su elección. Por ejemplo respecto al orden lexicográfico graduado si x tiene peso 1 e y tiene peso 3 el polinomio $f = x^2 + y$ tiene como término principal y mientras que si ambas tuvieran peso 1 seria x^2.

En nuestro caso hemos fijado el orden lexicográfico ya que así los pesos no afectan a la hora de ordenar los monomios. Así, en este ejemplo el *S*-polinomio es:

$$f_s = y + \dot{x} - (\dot{x} - \ddot{y})^{20} x^{12} y^{10} - x^{11} \dot{x}^{20} y^{10} \dot{y} + x^{12} \dot{x}^{20} y^{10}.$$

El siguiente paso es reducir el *S*-polinomio obtenido y añadir el resto a la base inicial, como se hacía con el resto en el algoritmo de Buchberger.

Para iniciar el proceso de reducción necesitamos saber el peso del *S*-polinomio y si es cuasi-homogéneo o no. De no serlo, como es el caso, ya que no todos los monomios tienen el mismo peso, nos quedamos con la componente principal.

El peso del *S*-polinomio es 82 con este peso calcularemos la base *B*, obtenida de realizar combinaciones de f_1 y f_2, hasta obtener todos los posibles polinomios diferenciales de peso 82.

El siguiente paso es poner la componente cuasi-homogénea principal del S-polinomio como combinación de las componentes cuasi-homogéneas principales de los elementos de la base B.

Así, por ejemplo la componente cuasi-homogénea principal del S-polinomio, f_s es $h(f_s) = -x^{12}y^{10}\dot{y}^{20}$ que es 1 por la componente principal de f_1.

Una vez hecho esto, el proceso de reducción se continúa restando al polinomio inicial la componente cuasi-homogénea principal y sumando los coeficientes obtenidos en el paso anterior multiplicando por su correspondiente elemento de la base B. En nuestro ejemplo, en el primer paso del método, obtenemos que el S-polinomio queda reducido a:

$$x^{12}\dot{x}^{20}y^{10} - x^{11}\dot{x}^{20}y^{10}\dot{y}$$

Continuamos reduciendo este segundo polinomio hasta que el polinomio reducido sea 0, igual que en el paso anterior o que el peso del polinomio obtenido sea menor que el de los elementos de la base inicial (F en nuestro ejemplo).

En este ejemplo, en concreto, lo que sucede es que da 0, es decir, que la H-base está compuesta únicamente por f_1, f_2 y seguimos sin tener un sistema más sencillo. De hecho es el mismo, por tanto lo que debemos hacer es reducir un elemento de la base inicial con respecto al otro.

Como el grado de f_1 es mucho mayor que el de f_2 vamos a intentar reducir f_1 respecto a f_2 obteniendo los elementos de la base B desde f_2 y siguiendo los mismos pasos que anteriormente, obtenemos que f_1 se puede expresar como $\dot{x} + y$ y por tanto, nuestro base inicial quedará como $F' = [\dot{x} + y, \dot{y} - x]$. Viéndolo como sistema obtenemos que nuestro sistema inicial es equivalente a:

$$\begin{cases} \frac{dx(t)}{dt} + y(t) = 0 \\ -x(t) + \frac{dy(t)}{dt} = 0 \end{cases}$$

Se trata de un sistema muy sencillo cuya solución es:

$$x(t) = \cos(t)x(0) - \sin(t)y(0)$$
$$y(t) = \sin(t)x(0) + \cos(t)y(0)$$

Para la obtención de los resultados en Sage se ha tenido que realizar un cambio de notación, de modo que a x se le ha llamado x_0 a $\dot{x}$ se le llama x_1 y a $\ddot{x}$, x_2 y de manera análoga con la y, de esta forma es necesario definir lo que es derivar.

También hay que observar que el anillo estará formado por $x_0, x_1, x_2, y_0, y_1, y_2$, pasa a ser finito y por tanto no se puede derivar de nuevo x_2 e y_2, es decir, tomamos la licencia de impedir la existencia de derivadas terceras ya que en nuestras ecuaciones iniciales no aparecen. Como consecuencia de esto el algoritmo realizado no sirve en general sino en unos casos determinados. Pero la modificación para otro tipo de casos no siendo compleja escapa de los objetivos de este trabajo.

Bibliografía

[1] DAVID COX, JOHN LITTLE, DONAL O'SHEA, *Ideals, Varieties and Algorithms*, Colección Springer, Tercera Edición, 2007.

[2] GIOVANI GALLO, BHUBANESWAR MISHRA, FRANÇOIS OLLIVER, *Some Constructions in Rings of Differential Polynomials*, Applied algebra, algebraic algorithms and error-correcting codes (New Orleans, LA, 1991), 171-182, Lecture Notes in Comput. Sci., **539**, Springer, Berlin, 1991.

[3] FRANÇOIS BOULIER, *Differential elimination and biological modelling*, Gröbner bases in symbolic analysis, 109-137, Radon Ser. Comput. Appl. Math., 2, Walter de Gruyter, Berlin, 2007

Índice alfabético

Anexo

Se muestran aquí los algoritmos empleados para la resolución de los ejemplos. Comenzamos con los programas empleados en el caso de anillos de polinomios no diferenciales

- S_polinomio.

Input (P, f, g). Donde P es el anillo, y f y g dos polinomios.
Output (s). s es un polinomio, concretamente el S-polinomio obtenido de f y g.

```
def S_polinomio(P,f,g):
    a=P(f).lm()
    b=P(g).lm()
    c=lcm(a,b)
    s=P((c/a)*f-(c/b)*g)
    return s
```

- alg. Algoritmo de la división.

Input (P, G, f). Donde P es el anillo, y G es una lista de polinomios y f un polinomio.
Output (I, r). Una lista de dos elementos, donde I es una lista de polinomios, los cuales corresponden a los cocientes al aplicar el algoritmo de la división, r un polinomio que corresponde al resto.

```
def alg(P,G,f):
   n=len(G)
   p=f
   I=n*[0]
   r=0
   while p!=0:
     i=0
     do=False
     while i<n and do==False:
        a=P(G[i]).lt()
        b=P(p).lt()
        if b%a==0:
           coci=P(b/a)
           I[i]=I[i]+coci
           p=p-coci*G[i]
           do=True
        else:
           i=i+1
     if do==False:
       r=r+b
       p=p-b
```

```
        return (I,r)
```

- baseG. Algoritmo de Buchberger.

Input (P,I). P es un anillo, I una lista de polinomios.
Output (G). G es una lista de polinomios, que forman una base de Gröbner.

```
def baseG(P,I):
   n=len(I)
   G=I
   for i in [0..n-2]:
      for j in [i+1..n-1]:
         S=S_polinomio(P2,G[i],G[j])
         M=alg(P,G,S)
         if M[1]!=0:
            G=G+[M[1]]
            return baseG(P,G)
   return G
```

- baseGmin. Cálculo de base minimal.

Input $(P,I,k=0)$. P es un anillo, I una lista de polinomios,k es un numero que si no se indica es 0.
Output $(G2)$. $G2$ es una lista de polinomios, que forman una base de Gröbner minimal.

```
def baseGmin(P,I,k=0):
   G=baseG(P,I)
   n=len(G)
   for i in [k..n-1]:
      G0=[G[j] for j in [0..n-1] if j<>i]
      G1=[G[j] for j in [k..n-1] if j<>i]
      M=alg(P,G1,G[i])
      if M[1]==0:
        return baseGmin(P,G0,i)
      G2=[G[s]/G[s].lc() for s in range(n)]
      return G2
```

Las siguientes funciones corresponden al caso de polinomios diferenciales

- derivada.

Input (p). p es un polinomio.
Output (d). d es un polinomio, resultante de calcular la derivada de p.

```
def derivada(p):
   mon=p.monomials()
   coe=[p.monomial_coefficient(_) for _ in mon]
   n=len(mon)
   d=0
   for i in range(n):
      if mon[i]%x2==0 or mon[i]%y2==0:
         return 'hay que cambiar el anillo'
      m=[x0,x1,y0,y1]
```

```
            dm=[x1,x2,y1,y2]
            for j in range(4):
               k=1
               while mon[i]%m[j]^k==0:
                   k=k+1
               d=d+coe[i]*(k-1)*dm[j]*mon[i]/m[j]

        return d
```

- grmon. Grado de un monomio.

Input (m). m es un monomio.
Output (gr, f, e). Es una lista formada por 3 elementos, gr es un número, el grado del monomio,f es una lista de números, los pesos de cada una de las variables,e es una lista de números, los exponentes de las variables del monomio.

```
def grmon(m):
   g=0
   if m%x3==0 or m%y3==0:
      return 'hay que cambiar el anillo'
   v=[x0,x1,x2,y0,y1,y2]
   e=6*[0]
   for j in range(len(v)):
      k=1
      while m%v[j]^k==0:
           k=k+1
      e[j]=e[j]+(k-1)
      f=[e[0],2*e[1],3*e[2],e[3],2*e[4],3*e[5]]
   return e[0]+e[3]+2*(e[1]+e[4])+3*(e[2]+e[5]),f,e
```

- resta.

Input (a, b). a y b son dos listas.
Output (r) r es una lista, obtenida de restar cada los valores de las listas a y b. Si las listas son de distinto tamaño devuelve que no puede restar.

```
def resta(a,b):
    n=len(a)
    m=len(b)
    if n!=m:
       return 'no se pueden restar'
    else:
       r=[a[i]-b[i] for i in range(n)]
       return r
```

- tplex. Término y monomio principal por el orden lexicográfico.

Input (p). Donde p es un polinomio.
Output (ter, mon). Una lista formada por dos elementos uno ter, correspondiente al término principal y otro mon, correspondiente al monomio principal, ambos por el orden lexicográfico.

```
 def tplex(p):
     mon=(p).monomials()
```

```
        coe=[p.monomial_coefficient(_) for _ in mon]
        n=len(mon)
        g=[grmon(i)[1] for i in mon]
        h=[grmon(i)[2] for i in mon]
        t=g[0]
        e=h[0]
        c=coe[0]
        for j in [1..n-1]:
            r=resta(t,g[j])
            k=0
            while r[k]==0:
                k=k+1
            if r[k]<0:
               t=g[j]
               e=h[j]
               c=coe[j]
            else:
               t=t
               e=e
               c=c
        ter=c*x0^e[0]*x1^e[1]*x2^e[2]*y0^e[3]*y1^e[4]*y2^e[5]
        mo=x0^e[0]*x1^e[1]*x2^e[2]*y0^e[3]*y1^e[4]*y2^e[5]
        return ter,mo
```

- tplex. Término y monomio principal por el orden lexicográfico graduado.

Input (p). Donde p es un polinomio.
Output (ter, mon). Una lista formada por dos elementos uno *ter*, correspondiente al término principal y otro *mon*, correspondiente al monomio principal, ambos por el orden lexicográfico graduado.

```
def tplexg(p):
    mon=(p).monomials()
    coe=[p.monomial_coefficient(_) for _ in mon]
    n=len(mon)
    o=[grmon(i)[0] for i in mon]
    g=[grmon(i)[1] for i in mon]
    h=[grmon(i)[2] for i in mon]
    O=o[0]
    t=g[0]
    e=h[0]
    c=coe[0]
    for j in [1,n-1]:
        if O<o[j]:
           O=o[j]
           t=g[j]
           e=h[j]
           c=coe[j]
        if O>o[j]:
           O=O
           t=t
           e=e
           c=c
```

```
    c=0
    s=0
    while s<n:
        if 0==o[s]:
           c=c+1
           s=s+1
    if c<2:
        ter=c*x0^e[0]*x1^e[1]*x2^e[2]*y0^e[3]*y1^e[4]*y2^e[5]
        mo=x0^e[0]*x1^e[1]*x2^e[2]*y0^e[3]*y1^e[4]*y2^e[5]
        return ter,mo
    else:
        pos=[i for i in [0..n-1] if o[i]==0]
        q=0
        for k in pos:
           q=q+mon[k]
        return tplex(q)
```

- tplexi. Término y monomio principal por el orden lexicográfico inverso.

Input (p). Donde p es un polinomio.
Output (ter, mon). Una lista formada por dos elementos uno *ter*, correspondiente al término principal y otro *mon*, correspondiente al monomio principal, ambos por el orden lexicográfico inverso.

```
def tplexi(p):
    mon=(p).monomials()
    coe=[p.monomial_coefficient(_) for _ in mon]
    n=len(mon)
    g=[grmon(i)[1] for i in mon]
    h=[grmon(i)[2] for i in mon]
    t=g[0]
    e=h[0]
    c=coe[0]
    for j in [1..n-1]:
        r=resta(t,g[j])
        k=len(r)-1
        while r[k]==0:
             k=k-1
        if r[k]>0:
             t=g[j]
             e=h[j]
             c=coe[j]
        else:
             t=t
             e=e
             c=c
    ter=c*x0^e[0]*x1^e[1]*x2^e[2]*y0^e[3]*y1^e[4]*y2^e[5]
    mo=x0^e[0]*x1^e[1]*x2^e[2]*y0^e[3]*y1^e[4]*y2^e[5]
    return ter,mo
```

- tplex. Término y monomio principal por el orden lexicográfico inverso graduado.

Input (p). Donde p es un polinomio.
Output (ter, mon). Una lista formada por dos elementos uno *ter*, correspondiente al término principal y otro *mon*, correspondiente al monomio principal, ambos por el orden lexicográfico inverso graduado.

```
def tplexgi(p):
    mon=(p).monomials()
    coe=[p.monomial_coefficient(_) for _ in mon]
    n=len(mon)
    o=[grmon(i)[0] for i in mon]
    g=[grmon(i)[1] for i in mon]
    h=[grmon(i)[2] for i in mon]
    O=o[0]
    t=g[0]
    e=h[0]
    c=coe[0]
    for j in [1,n-1]:
        if O<o[j]:
           O=o[j]
           t=g[j]
           e=h[j]
           c=coe[j]
        if O>o[j]:
           O=O
           t=t
           e=e
           c=c
    c=0
    s=0
    while s<n:
        if O==o[s]:
           c=c+1
        s=s+1
    if c<2:
        ter=c*x0^e[0]*x1^e[1]*x2^e[2]*y0^e[3]*y1^e[4]*y2^e[5]
        mo=x0^e[0]*x1^e[1]*x2^e[2]*y0^e[3]*y1^e[4]*y2^e[5]
        return ter,mo
    else:
        pos=[i for i in [0..n-1] if o[i]==0]
        q=0
        for k in pos:
            q=q+mon[k]
        return tplexi(q)
```

- dS_polinomio.

Input $(orden, A, f, g)$. *orden* es un número del 1 al 4 que indica el tipo de orden monomial con el que se va a calcular el S-polinomio. (1-lexicográfico, 2-lexicográfico graduado, 3-lexicográfico inverso, 4-lexicográfico graduado inverso). A es el anillo, f y g son dos polinomios.

Output (s), donde s es un polinomio correspondiente al S-polinomio.

```
def dS_polinomio(orden,A,f,g):
    if orden==1:
       a=tplex(f)[1]
       b=tplex(g)[1]
    if orden==2:
       a=tplexg(f)[1]
```

```
        b=tplexg(g)[1]
    if orden==3:
        a=tplexi(f)[1]
        b=tplexi(g)[1]
    if orden==4:
        a=tplexgi(f)[1]
        b=tplexgi(g)[1]
    c=lcm(a,b)
    s=A((c/a)*f-(c/b)*g)
    return s
```

- eli.

Input $(lista)$. *lista* es una lista
Output (l). l es una lista igual que *lista* pero sin elementos repetidos.

```
def eli(lista):
    l=[]
    for i in lista:
        if i not in l:
           l.append(i)
    return sorted(l)
```

- comqh. Componentes quasi-homogénas

Input (p). p es un polinomio.
Output (s) s es una lista formada por las componentes quasi-homogéneas de p es decir una lista polinomios que tienen todos los términos con el mismo peso.

```
def comqh(p):
    mon=(p).monomials()
    coe=[p.monomial_coefficient(_) for _ in mon]
    n=len(mon)
    o=[grmon(i)[0] for i in mon]
    l=eli(o)
    m=len(l)
    s=m*[0]
    i=0
    while i<n:
          j=0
          while j<m:
              if o[i]==l[j]:
                 s[j]=s[j]+coe[i]*mon[i]
              j=j+1
          i=i+1
     return s
```

- comqhp. Componente quasi-homogénea principal.

Input (p). p es un polinomio.
Output (s). s es un polinomio con todos sus términos con el mismo peso siendo este el peso mayor de todas las componentes quasi-homogéneas, es decir s es la componente quasi-homogénea principal.

```
def comqhp(p):
    mon=(p).monomials()
    coe=[p.monomial_coefficient(_) for _ in mon]
    n=len(mon)
    o=[grmon(i)[0] for i in mon]
    l=eli(o)
    m=len(l)
    g=l[m-1]
    s=0
    i=0
    while i<n:
        if o[i]==g:
           s=s+coe[i]*mon[i]
        i=i+1
   return s
```

- wp. Peso de un polinomio

Input (p). p es un polinomio.
Output g. g es un número que corresponde al peso de p.

```
def wp(p):
    mon=(p).monomials()
    n=len(mon)
    o=[grmon(i)[0] for i in mon]
    l=eli(o)
    m=len(l)
    g=l[m-1]
    return g
```

- eliw

Input $(I1,w)$. $I1$ es una lista de listas, donde cada sublista tiene 2 elementos es decir $I1 = [[a_i,b_1]...[a_n,b_n]]$ donde a_i es la componente quasi-homogénea de b_i. w es un número.

Output (Lw,L). Devuelve 2 listas de listas, la primeraLw está formada por los elementos de $I1$ cuya primera componente (a_i) tiene peso w. La segunda L está formada por los elementos de $I1$ cuya primera componente tiene peso menos que w.

```
def eliw(I1,w):
    n=len(I1)
    Lw=[]
    L=[]
    for j in range(n):
        if wp(I1[j][0])==w:
           Lw=Lw+[I1[j]]
        if wp(I1[j][0])<w:
           L=L+[I1[j]]
    return Lw,L
```

- eliminar.

Input (L). L es una lista de listas formadas donde cada sublista tiene 2 elementos es decir $L = [[a_i,b_1]...[a_n,b_n]]$ donde a_i es la componente quasi-homogénea de b_i.

Output (L). L es igual que la L del input, la diferencia es que se han eliminado las sublistas que tenían el a_i repetido.

```
def eliminar(L):
    n=len(L)
    aux=L
    for i in range(n-1):
        for j in [i+1..n-1]:
            if aux[i][0]==aux[j][0]:
               aux=[aux[k] for k in [0..j-1]+[j+1..n-1] ]
               return eliminar(aux)
   return L
```

- baseSmul.

Input (I,w). I es una lista de listas donde cada sublista tiene 2 elementos es decir $I=[[a_i,b_1]...[a_n,b_n]]$ donde a_i es la componente quasi-homogénea de b_i. w es un número.

Output (Lw).Lw es una lista de listas donde cada sublista tiene 2 elementos es decir $Lw=[[a_i,b_1]...[a_n,b_n]]$ donde a_i es la componente quasi-homogénea de b_i y todas las a_i tienen peso w.

```
def baseSmul(I,w):
    Lw=eliw(I,w)[0]
    L=eliw(I,w)[1]
    V=[x0,x1,x2,y0,y1,y2]
    n=len(L)
    while n!=0:
        L0=[[L[0][0]*i,L[0][1]*i] for i in V]
        L1=eliw(L0,w)[0]
        L2=eliw(L0,w)[1]
        Lw=Lw+L1
        L=[i for i in L if i<>L[0]]+L2
        n=len(L)
    return eliminar(Lw)
```

- baseSw.Base S de polinomios de peso w.

Input (I,gr,R). I es una lista de polinomios,gr es un número R es el anillo de polinomios.

Output (L). Lw es una lista de listas donde cada sublista tiene 2 elementos es decir $Lw=[[a_i,b_1]...[a_n,b_n]]$ donde a_i es la componente quasi-homogénea de b_i y todas las a_i tienen peso w

```
def baseSw(I,gr,R):
   I1=[[comqhp(i),i] for i in I]
   L=baseSmul(I1,gr)
   L1=[[comqhp(i),i] for i in I if alg(R,[x2,y2],comqhp(i))[0]!=[0,0]]
   L2=[[comqhp(i),i] for i in alg(R,[x2,y2],comqhp(i))[0]==[0,0]]
   L3=baseSmul(L1,gr)
   L4=[[R(derivada(L2[j][0])),R(derivada(L2[j][1]))] for j in range(len(L2))]
   L5=eliw(L4,gr)[0]
   L6=eliw(L4,gr)[1]
   n=len(L6)
   while n!=0:
      l1=[i for i in L6 if alg(R,[x2,y2],i[0])[0]!=[0,0]]
      l2=[i for i in L6 if alg(R,[x2,y2],i[0])[0]==[0,0]]
      L3=L3+baseSmul(l1,gr)
      L4=[[R(derivada(l2[j][0])),R(derivada(l2[j][1]))] for j in range(len(l2))]
      L5=L5+eliw(L4+L3,gr)[0]
```

```
    L6=eliw(L4+L3,gr)[1]
    n=len(L6)
  return eliminar(L+L3+L5)
```

- GL

Input (L). L es una lista de listas donde cada sublista tiene 2 elementos, es decir, $Lw = [[a_i, b_1]...[a_n, b_n]]$ donde a_i es la componente quasi-homogénea de b_i y todas las a_i tienen peso w

Output (L). L es una lista de listas donde cada sublista tiene 2 elementos es decir $L = [[a_i, b_1]...[a_n, b_n]]$ donde a_i es la componente quasi-homogénea de b_i y todas las a_i tienen peso w principal, la diferencia es que sus coeficientes forman un matriz triangular superior.

```
def GL(L):
   L1=sorted(L,reverse=True)
   n=len(L1)
   i=0
   while i<n-1:
       j=i+1
       while j<n:
           c=alg(R,[L1[i][0]],L1[j][0])
           if c[0]!=[0]:
              L2=[L1[k] for k in [0..n-1] if k<>j]
              L3=[[L1[i][0]-c[0][0]*L1[j][0],L1[i][1]-c[0][0]*L1[j][1]]]
              L1=L2+L3
              j=j+1
           else:
              j=j+1
           i=i+1
           L1=sorted(L1,reverse=True)
       L1=sorted(L1,reverse=True)
   return sorted(L1,reverse=True)
```

- reducqh. Método de reducción para componente quasi-homogéneas.

Input (p, I, R). p es un polinomio quasi-homogéneo, I es una lista de polinomios, R es un anillo.

Output (q). q es un polinomio obtenido del proceso de reducción de p respecto a la base S obtenida de I.

```
def reducqh(p,I,R):
    w=wp(p)
    S=baseSw(I,w,R)
    S0=GL(S)
    S1=[i[0] for i in S0 if i[0]<>0]
    c=alg(R,S1,p)[0]
    L=[[c[i],S0[i][1]] for i in range(len(S0)) if c[i]!=0]
    q=p
    for j in range(len(L)):
        q=q-L[j][0]*L[j][1]
    return q
```

- reducp. Método de reducción.

Input (p, I, R). p es un polinomio, I es una lista de polinomios, R es un anillo.

Output $(q1)$. $q1$ es un polinomio obtenido del proceso de reducción de p respecto a la base S obtenida de I.

```
def reducp(p,I,R):
    p1=comqhp(p)
    q=reducqh(p1,I,R)
    q1=p-p1+q
    H=[wp(i) for i in I]
    H1=sorted(H)
    h=H1[0]
    if q1==0:
       return q1
    if wp(q1)>=h:
       return reducp(q1,I,R)
    else:
       return q1
```

- reduc. Proceso de reducción.

Input $(I,R,orden)$. I es un ideal, R es un anillo, *orden* es un número del 1 al 4 que indica el tipo de orden monomial (1-lexicográfico, 2-lexicográfico graduado, 3-lexicográfico inverso, 4-lexicográfico graduado inverso).

Output (G). G es un lista de polinomios, que constituye una H-base.

```
def reduc(I,R,orden):
    n=len(I)
    G=I
    for i in [0..n-2]:
        for j in [i+1..n-1]:
            S=dS_polinomio(orden,R,G[i],G[j])
            M=reducp(S,I,R)
            if M!=0:
               G=G+[M]
               return reduc(S,G)
    return G
```

Tras realizar todos estos algoritmos y probar algún ejemplo como:

$$\begin{cases} g1 = x0x1 + y1^2 \\ g2 = 3x0 + y0x1 + y1^3 \end{cases}$$

$G = [g1, g2]$ es el ideal, con el que obtenemos resultados como que el S-polinomio es:

$$-x1y1^3 - x1^2y0 - 2x0x1 + y1^2.$$

Y al reducirlo nos da 0 y al construir la H-base obtenemos que nos da el mismo ideal G.

Pero al hacerlo con polinomios de mayor peso el calculo de la base S tarda mucho pues ha de realizar muchas operaciones por eso algunas de las funciones han sido mejoradas para que en lugar de calcular toda la base S solo calcule los elementos que son necesarios para realizar la reducción. Estas funciones se muestra a continuación.

- baseSmul1

Input (p,I,R). p es un polinomio,I es una lista de polinomios,R es un anillo.

Output (Lw). Lw es una lista de lista, con cada sublista formada por dos elementos $[a_i, b_1]$, el primero, las componentes quasi-homogéneas de peso el de p formada a partir de las componentes quasi-homogéneas de I y el segundo los polinomios de peso el de p de los cuales a_i son las componentes quasi-homogéneas principales.

```
def baseSmul1(p,I,R):
    p1=comqhp(p)
    w=wp(p)
    I1=[[comqhp(i),i] for i in I]
    n=len(I)
    I2=[i for i in I1  if alg(R,[R(i[0])],R(p1))[1]==0]
    Lw=eliw(I2,w)[0]
    L=eliw(I2,w)[1]
    L=[i for i in L if alg(R,[i[0]],p1)[1]==0]
    n=len(L)
    m=len(Lw)
    if m!=0:
       return eliminar(Lw)
    else:
       while n!=0:
           a=[alg(R,[R(L[i][0])],R(p1))[0][0] for i in range(n)]
           L0=[[L[0][0]*a[i],L[0][1]*a[i]] for i in range(n)]
           L1=eliw(L0,w)[0]
           L2=eliw(L0,w)[1]
           Lw=eliminar(Lw+L1)
           m=len(Lw)
           if m!=0:
              return eliminar(Lw)
           else:
               L=[i for i in L if i<>L[0]]+L2
               L=[L[i] for i in [0..len(L)-1] if alg(R,[L[i][0]],p1)[1]==0]
               n=len(L)
       return eliminar(Lw)
```

- baseSw1.

Input (p,I,R). p es un polinomio,I es una lista de polinomios,R es un anillo.

Output (Lw). Lw es una lista de lista, con cada sublista formada por dos elementos $[a_i,b_1]$, el primero, las componentes quasi-homogéneas de peso el de p formada a partir de las componentes quasi-homogéneas de I y el segundo los polinomios de peso el de p de los cuales a_i son las componentes quasi-homogéneas principales.

```
def baseSw1(p,I,R):
   gr=wp(p)
   L=baseSmul1(p,I,R)
   m=len(L)
   if m!=0:
     return eliminar(L)
   else:
     p1=comqhp(p)
     I1=[[comqhp(i),i] for i in I]
     L1=[i for i in I if alg(R,[x2,y2],comqhp(i))[0]!=[0,0]]
     L2=[[comqhp(i),i] for i in I if alg(R,[x2,y2],comqhp(i))[0]==[0,0]]
     e=len(L2)
     L3=baseSmul1(p,L1,R)
     L4=[[R(derivada(L2[j][0])),R(derivada(L2[j][1]))] for j in range(e)]
     L4=[i for i in L4 if alg(R,[i[0]],p1)[1]==0]
```

```
L5=eliw(L4,gr)[0]
L6=eliw(L4,gr)[1]
n=len(L6)
while n!=0:
   l1=[i[1] for i in L6 if alg(R,[x2,y2],i[0])[0]!=[0,0]]
   l2=[i for i in L6 if alg(R,[x2,y2],i[0])[0]==[0,0]]
   L3=L3+baseSmul1(p,l1,R)
   e2=len(l2)
   L4=[[R(derivada(l2[j][0])),R(derivada(l2[j][1]))] for j in range(e2)]
   L4=[i for i in L4 if alg(R,[i[0]],p1)[1]==0]
   L5=L5+eliw(L4+L3,gr)[0]
   L6=eliw(L4+L3,gr)[1]
   n=len(L6)
return eliminar(L+L3+L5)
```

- reducqh1. Método de reducción para componente quasi-homogéneas.

Input (p,I,R). p es un polinomio quasi-homogéneo, I es una lista de polinomios, R es un anillo.
Output (q). q es un polinomio obtenido del proceso de reducción de p respecto a la base S obtenida de I.

```
def reducqh1(p,I,R):
    w=wp(p)
    S=baseSw1(p,I,R)
    S0=GL(S)
    S1=[i[0] for i in S0 if i[0]<>0]
    q=p
    if len(S1)!=0:
       c=alg(R,S1,p)[0]
       L=[[c[i],S0[i][1]] for i in range(len(S0)) if c[i]!=0]
       for j in range(len(L)):
           q=q-L[j][0]*L[j][1]
    return q
```

- reducp1. Método de reducción.

Input (p,I,R). p es un polinomio, I es una lista de polinomios, R es un anillo.
Output $(q1)$. $q1$ es un polinomio obtenido del proceso de reducción de p respecto a la base S obtenida de I.

```
def reducp1(p,I,R):
    p1=comqhp(p)
    q=reducqh1(p1,I,R)
    q1=p-p1+q
    if q1==p:
       return q1
    H=[wp(i) for i in I]
    H1=sorted(H)
    h=H1[0]
    if q1==0:
       return q1
    if wp(q1)>=h:
       return reducp1(q1,I,R)
    return q1
```

- basedG. H-base.

Input $(P,I,orden)$. P es un anillo, I es una lista de ideales, *orden* es un número del 1 al 4 que indica el tipo de orden monomial (1-lexicográfico, 2-lexicográfico graduado, 3-lexicográfico inverso, 4-lexicográfico graduado inverso).

Output (G). G es una lista de polinomios.

```
def basedG(P,I,orden):
    n=len(I)
    G=I
    for i in [0..n-2]:
        for j in [i+1..n-1]:
            S=dS_polinomio(orden,P,G[i],G[j])
            M=reducp1(S,G,P)
            if M!=0:
               G=G+[M]
               return basedG(P,G,orden)
     return G
```

- basedGmin. H-base minimal.

Input $(P,I,orden,k=0)$. P es un anillo, I es una lista de polinomios, *orden* es un número del 1 al 4 que indica el tipo de orden monomial (1-lexicográfico, 2-lexicográfico graduado, 3-lexicográfico inverso, 4-lexicográfico graduado inverso) y k es un número que si no se da vale 0.

Output $(G2)$. $G2$ es una lista de polinomios.

```
def basedGmin(P,I,orden,k=0):
    G=basedG(P,I,orden)
    n=len(G)
    if n==len(I):
       H=[[wp(i),i] for i in I]
       H1=sorted(H,reverse=True)
       h=H1[0][1]
       I1=[i for i in I if i<>h]
       r=reducp1(h,I1,P)
       G=[r]+I1
       G2=[G[s]/G[s].lc() for s in range(n)]
       return G2
    for i in [k..n-1]:
       G0=[G[j] for j in [0..n-1] if j<>i]
       G1=[G[j] for j in [k..n-1] if j<>i]
       M=alg(P,G1,G[i])
       if M[1]==0:
          return basedGmin(P,G0,orden,i)
    G2=[G[s]/G[s].lc() for s in range(n)]
    return G2
```