Universidad Politécnica
de Madrid

**Escuela Técnica Superior de
Ingenieros Informáticos**

Grado en Matemáticas e Informática

Trabajo Fin de Grado

Bases de Gröbner y Aplicaciones

Autor: Juan García de la Cruz García
Tutor: Jonathan Sánchez Hernández

Madrid, Julio 2020

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Matemáticas e Informática

Título: Bases de Gröbner y Aplicaciones

Julio 2020

Autor: Juan García de la Cruz García
Tutor: Jonathan Sánchez Hernández
Departamento de Matemática Aplicada a las TIC
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

Una base de Gröbner es un tipo especial de conjunto generador de un ideal de un anillo de polinomios que cumple una serie de propiedades muy interesantes. Este trabajo tiene como objetivo su estudio, para ello se realiza una aproximación matemática a las bases de Gröbner, desde los conceptos más fundamentales hasta cononocer algunas de sus aplicaciones. Además se implementa una biblioteca en C# que recoge toda la teoría vista, junto una aplicación gráfica que muestra el potencial de la biblioteca y a par que el de las propias bases de Gröbner. Todo esto irá acompañado de numerosos ejemplos que pretenden facilitar la comprensión.

Abstract

A Gröbner basis is a special kind of generator set of an ideal in a polynomial ring that satisfies a series of very interesting properties. The current project tackles their study. For this, a mathematical approach is done to the subject, starting with the most basic underlying ideas, and walking the reader comprehensively through the not so easy concepts, up to some of their applications. Numerous original examples will be given throughout the document with the aim of easing and helping with the reading and understanding of the theory presented.
Additionaly a C# library is developed, which implements all the theoretical concepts given, this library is accompanied by a graphical application that showcases the potential of both, the library and Gröbner bases.

Tabla de contenidos

Capítulo 0

Introducción

Una base de Gröbner es un tipo especial de conjunto generador de un ideal de un anillo de polinomios. Es epecial ya que cumple una serie de propiedades que nos permite profundizar en la rica estructura de los ideales, permitiendo deducir información sobre ellos y sus variedades algebraicas asociadas, entre otras muchas cosas. Uno de sus usos más comunes es la resolución de sistemas de ecuaciones polimoniales, pero sus aplicaciones no se limitan solo a ello, como veremos hay muchas y algunas de ellas inesperadas. El único inconveniente es su alto coste computacional, aunque en los últimos años se ha realizado mucho trabajo al respecto, reduciendo de manera notable dicho coste.

El concepto de las bases de Gröbner se introdujo formalmente en la tesis doctoral de Bruno Buchberger, y esta a su vez se originó cuando este tras atender a un seminario impartido en 1964 por Wolfgang Gröbner, le solicitó al mismo el estudio del contenido del seminario como la materia de su tesis.[19] Y es que Gröbner ya conocía la existencia de estas bases y un método para calcularlas, sin embargo todavía quedaban cuestiones por resolver que impedían dar una definición formal y un algoritmo que permitiera calcularlas en cualquier caso, no solo en ejemplos concretos. Estas cuestiones por resolver fueron el trabajo que Gröbner entregó a Buchberger. Y este les dio el nombre de su tutor en su honor.

Por otro lado es curioso que ya en 1931 Nikolai Gjunte había llegado a un concepto similar, pero pasó desapercibido para la mayor parte de la comunidad matemática, en el articulo [18] se traduce su trabajo del ruso al inglés y se enlaza con el trabajo de Buchberger.

Existen diversas implementaciones de las bases de Gröbner y la teoría de que las rodea en muchos paquetes matemáticos, las más conocidas son:

- CoCoA
- Maple
- Macaulay 2
- Mathematica
- SINGULAR
- SageMath
- SymPy (biblioteca de Python)

Con la salvedad de Python, estos paquetes suelen ser usados casi exclusivamente en el ámbito académico, siendo el coste de alguno de ellos casi prohibitivo.

En este trabajo se realizará una aproximación matemática a las bases de Gröbner desde la teoría más básica que las sustenta hasta conocer algunas de sus muchas aplicaciones. Por último se desarrollará una biblioteca en C# que recoga toda la teoría vista, junto con un aplicación gráfica que hace uso de ella para resolver el problema de la *k-coloreabilidad*. Con esto se pretende implementar una biblioteca en un lenguaje no tan estrechamente relacionado con el mundo matemático.

Estructura del documento

El documento se estructura en 6 capítulos. El primero presenta los conceptos base necesarios, es decir, *anillos*, *ideales*, ideales en *anillos de polinomios* y la *división en una sola variable* en anillos de polinomios. Esta materia es por lo general parte del programa de la mayoría de lo grados de matemáticas. Al final del capítulo se relacionarán estos conceptos con la geometría, intentando así suscitar la necesidad de lo que serán las bases de Gröbner. En el segundo capítulo se entrará ya en un terreno un poco más avanzado, introduciendo la *división multivariable* en anillos de polimonios, mostrando previamente nociones sobre orden de polinomios. Esto será necesario e imprescindible para poder abarcar el capítulo 3, que será el que introducirá formalmente las bases de Gröbner. En este tercer capítulo se definirán y se darán algunas de sus propiedades más importantes. También se enseñará como hallarlas y se estudiarán algunos casos particulares que por sus propiedades resultarán esenciales. En el cuarto capítulo se expondrán algunas de sus aplicaciones, empezando por las elementales y pasando a alguna más compleja, como el *teorema de los ceros de Hilbert*, siendo este de gran interés matemático. Se mostrarán además alguna aplicación curiosa y/o inesperada. En el quinto capítulo se dará la implentación que se ha llevado a cabo de la biblioteca, explicándola detalladamente y pormenorizadamente. En el mismo capítulo se mostrará la aplicación gráfica también desarrollada. Todos estos capítulos incluyen sendos ejemplos que ayudan a entender los conceptos presentados, siendo todos, los cerca de 50 ejemplos, originales de este documento. Por último, en el sexto capítulo, se recogerán algunas conclusiones y reflexiones del autor sobre el trabajo realizado.

Capítulo 1

Anillos e Ideales

En este capítulo se pretende introducir de forma breve y concisa algunos conceptos fundamentales del álgebra abstracta que serán los cimientos subyacentes de lo que se presentará en los siguientes capítulos.

1.1. Anillos

Definición 1.1.1 (Anillo)**.** Un *anillo* es un triplete $(R, +, \cdot)$ donde R es un conjunto y $+, \cdot$ son dos operaciones internas, que denominaremos suma y producto respectivamente, que satisfacen las siguientes propiedades:

1. $(R, +)$ es un grupo abeliano con 0 como elemento neutro.
2. $(a \cdot b) \cdot c = a \cdot (b \cdot c)$, para todo $a, b, c \in R$.
3. $(a + b) \cdot c = (a \cdot c) + (b \cdot c) \qquad y \qquad a \cdot (b + c) = (a \cdot b) + (a \cdot c)$ para todo $a, b, c \in R$.

Definición 1.1.2 (Anillo conmutativo)**.** Sea R un anillo, R es *anillo conmutativo* si:

$$a \cdot b = b \cdot a \quad \text{para todo } a, b \in R.$$

Definición 1.1.3 (Anillo con identidad)**.** Sea R un anillo, R es *anillo con identidad* si existe $1_R \in R$ tal que $1 \neq 0$ y $1 \cdot a = a$ para todo $a \in R$.

Definición 1.1.4 (Anillo de división)**.** Sea R un anillo con identidad, R es *anillo de división* si para todo $a \in R$ con $a \neq 0_R$ existe a^{-1} tal que $a^{-1} \cdot a = a \cdot a^{-1} = 1_R$.

Definición 1.1.5 (Cuerpo)**.** Un anillo conmutativo con identidad y de división se denonima *cuerpo*.

Ejemplo 1.1.1. Con las operaciones de multiplicación y suma ordinarias $(\mathbb{R}, +, \cdot)$ es cuerpo. En cambio es fácil ver que $(\mathbb{Z}, +, \cdot)$ es anillo pero no es cuerpo. Por ejemplo escogiendo $3 \in \mathbb{Z}$ no existe $x \in \mathbb{Z}$ tal que $3 \cdot x = 1$.

Definición 1.1.6 (Subanillo)**.** Sea $(R, +, \cdot)$ un anillo y S un subconjunto no vacio de R. Diremos que S es un *subanillo* si $(S, +, \cdot)$ es anillo.

Proposición 1.1.1 (Caracterización de subanillo). *Sea R un anillo y S un subconjunto no vacio de R. S es un **subanillo** si y solo si satisface las siguientes propiedades:*

1. $a \cdot b \in S$, *para todo* $a, b \in S$.
2. $a - b \in S$, *para todo* $a, b \in S$.

Ejemplo 1.1.2. El conjunto $2\mathbb{Z} = \{n \in \mathbb{Z} \mid n = 2 \cdot a, \text{ para todo } a \in \mathbb{Z}\}$ es subanillo de $\mathbb{Z}$. Tenemos que para todo a,b $\in 2\mathbb{Z}$, $a = 2 \cdot c$ y $b = 2 \cdot d$ con $c, d \in \mathbb{Z}$, entonces:

1. $a \cdot b = (2 \cdot c) \cdot b = 2 \cdot (c \cdot b) \in 2\mathbb{Z}$ es fácil ver que también se cumple para $b \cdot a$.
2. $a - b = (2 \cdot c) + (2 \cdot -d) = 2 \cdot (c - d) \in 2\mathbb{Z}$.

1.2. Ideales

En esta sección vamos a definir un tipo especial de subanillo que será elemental a lo largo del documento.

Definición 1.2.1 (Ideal)**.** Sea R un anillo y S un subanillo conmutativo de R. I es un *ideal* si para todo $a \in I$ y $r \in R$ se cumple:

$$a \cdot r \in I.$$

Proposición 1.2.1. *[2] Sea R un anillo y sea* $A = \{a_1, ..., a_m\} \subseteq R$. *Se llama* ***ideal generado*** *por el conjunto A al mínimo ideal que contiene a A, y se representa por:*

$$I = \langle a_1, ..., a_m \rangle.$$

Definición 1.2.2 (Ideal principal)**.** Un ideal de R generado por un único elemento se llama *ideal principal*.

Proposición 1.2.2. *[2] Sea R un anillo conmutativo con identidad. Entonces el ideal generado por* $A = \{a_1, ..., a_m\} \subseteq R$ *es:*

$$I = \langle a_1, ..., a_m \rangle = \{r_1 a_1 + ... + r_m a_m : r_1 ... r_m \in R\}$$

Ejemplo 1.2.1. El subanillo $2\mathbb{Z} = \langle 2 \rangle$ visto en el ejemplo 1.1.2 es claramente un ideal generado por $2 \in \mathbb{Z}$. De hecho para cualquier $n \in \mathbb{Z}$ se tiene que $n\mathbb{Z}$ es un ideal, que además es principal.

1.3. Anillos de Polinomios

Los polinomios generalmente se relacionan con las funciones, pero estos cumplen una serie de propiedades que hace interesante su estudio más allá de este campo. Además, como veremos, definen una clase de anillo que utilizaremos de forma constante a lo largo de este trabajo.

Definición 1.3.1 (Polinomios sobre un cuerpo)**.** Sea k un cuerpo. Se llama *polinomio de n variables sobre k* a la suma finita de términos de la forma $a x_1^{\beta_1} \cdots x_n^{\beta_n}$ con $a \in k$ y $\beta_i \in \mathbb{N}, i = 1, ...n$.

Ejemplo 1.3.1. Por ejemplo, $f = 1 + x_1 + x_2^2 + x_3^3$ y $g = x_1^2 + 10x_2 + x_2 x_3^3$ son polinomios de 3 variables.

Nota. Denotaremos por $k[x_1, ..., x_n]$ al conjunto de todos los polinomios de n variables con coeficientes en el cuerpo k.

Definición 1.3.2 (Anillo de polinomios)**.** Sea k un cuerpo. Entonces $k[x_1, ..., x_n]$, con las operaciones usuales de suma y producto de polinomios, es un anillo conmutativo con identidad al que llamaremos *anillo de polinomios*.

Nota. Cuando se trabaje con una, dos o tres variables se utilizarán x, y o z en vez de x_i según sea necesario.

Ejemplo 1.3.2. Los polinomios $f = x + y + z + 2$ y $g = x - 3y + z^4$ pertenecen al anillo de los polinomios $\mathbb{R}[x, y, z]$.

Ejemplo 1.3.3. $I = \langle x + 1, y \rangle$ es un ideal de $\mathbb{R}[x, y]$. Es fácil apreciar que los polinomios $f = 2x^2 + x^3 + x$ y $g = xy + y + y^3$ pertenecen a I puesto que $f = 2x^2 + x^3 + x = (x + 1)(x + x^2) + y \cdot 0$ y $g = xy + y + y^3 = (x + 1)y + y \cdot y^2$.

A continuación presentaremos dos teoremas de gran relevancia, gracias a los cuáles se ha conseguido ahondar y expandir la visión que se tiene del álgebra abstracta.

Teorema 1.3.1 (Teorema de la base de Hilbert). *Todo ideal $I \subset k[x_1, ..., x_n]$ tiene un conjunto generador finito.*

Teorema 1.3.2 (Condición de la cadena ascendente). *Sea $I_1 \subset I_2 \subset I_3 \subset \cdots$ una cadena ascendente de ideales en $k[x_1, ..., x_n]$. Entonces existe un $N \geq 1$ tal que $I_N = I_{N+1} = I_{N+2} = \cdots$.*

La demostración de ambos queda fuera del alcance de este trabajo.

1.3.1. División en una variable

La división entre polinomios de $k[x]$ suele ser objeto de estudio en los cursos de secundaria. Aquí lo revisitaremos , pero introduciendo una terminología que nos permitirá prufundizar en la estructura de los ideales de $k[x]$.

Definición 1.3.3 (Término principal)**.** Sea $f = a_0x^m + a_1x^{m-1} + \cdots + a_m, \in k[x]$ no nulo con $a_i \in k$ y $a \neq 0$, llamaremos a a_0x^m *términino principal de f*, denotado $lt(f)$.

Definición 1.3.4 (Grado)**.** Sea $f \in k[x]$ llamaremos *grado de f* al exponente del término principal. Denotaremos al grado de f por $gr(f)$.

Definición 1.3.5. Diremos que un polinomio es *mónico* si el coeficiente de su término principal es 1.

Proposición 1.3.1. *Sean $f, g \in k[x]$ no nulos, entonces*

$$gr(f) \leq gr(g) \leftrightarrow lt(f) \text{ divide a } lt(g).$$

Ejemplo 1.3.4. El polinomio $f = 4x^3 + 7x^2 - 3x + 1$ tiene como término principal a $4x^3$, y su grado es 3. Por otro lado, dado que el coeficiente de $lt(f) = 4x^3$ es 4, se tiene que f no es mónico.

Teorema 1.3.3. *[3] Sea g un polinomio no nulo de $k[x]$. Entonces para cualquier $f \in k[x]$ existen $q, r \in k[x]$ tales que f se puede escribir como*

$$f = qg + r, \text{ con } r = 0 \text{ o } grd(r) < grd(g).$$

Además, r y q son únicos. Llamaremos a r resto y a q cociente.

El algoritmo introducido a continuación, conocido como el algoritmo de división en una variable, nos permitirá encontrar q y r.

Algoritmo 1: algoritmo de división en $k[x]$

Entrada: $f, g \in k[x]$ con $g \neq 0$
Resultado: q, r tales que $f = qg + r$ y $r = 0$ o $gr(r) < gr(g)$.
Inicialización: $q := 0, r := f$
mientras $r \neq 0$ ***y*** $gr(g) \leq gr(r)$ **hacer**
  $q := q + \frac{lt(r)}{lt(g)}$;
  $r := r - \frac{lt(r)}{lt(g)}g$;
fin

Ejemplo 1.3.5. Sean $f = 4x^3 + 2x^2 - 16$ y $g = 2X^2$ en $\mathbb{Q}[x]$ utilizaremos el algoritmo de división (1) para expresar f de la forma $f = qg + r$.

INICIALIZACIÓN: $q := 0, r := f$
Primera iteración $(r \neq 0$ y $gr(g) \leq gr(r))$

$$q := 0 + \frac{4x^3}{2x^2} = 2x$$
$$r := 4x^3 + 2x^2 - 16 - \frac{4x^3}{2x^2}2X^2 = 2x^2 - 16$$

Segunda iteración $(r \neq 0$ y $gr(g) \leq gr(r))$

$$q := 2x + \frac{2x^2}{2x^2} = 2x + 1$$
$$r := 2x^2 - 16 - \frac{2x^2}{2x^2}2X^2 = -16$$

Final del bucle $(gr(g) > gr(r))$

$$f = q \cdot g + r = (2x + 1)(2x) - 16$$

Aun que este algoritmo pueda parecer nuevo, es en realidad la forma algorítmica de la división larga de polinomios que se aprende en el instituto. Repitamos de nuevo la división anterior, expresándola esta vez con el formato de la división larga.

$$\begin{array}{rlll|ll} & 4x^3 & +2x^2 & -16 & 2x^2 & \\ \hline - & 4x^3 & & & 2x & +1 \\ \cline{1-4} & 0 & 2x^2 & -16 & & \\ - & & 2x^2 & & & \\ \cline{1-4} & & & -16 & & \end{array}$$

Efectivamente, obtenemos el mismo resultado. Por otro lado, resulta interesante ver que $I = \langle 4x^3 + 2x^2 - 16, 2x^2 \rangle = \langle -16, 2x^2 \rangle$ puesto que $-16 = (4x^3 + 2x^2 - 16) - (2x + 1) \cdot (2x^2)$. Esto no es algo particular de este ejemplo, por el teorema 1.3.3 tenemos que cualquier $f \in k[x]$ se puede expresar de la forma $f = q \cdot g + r$ por lo que el ideal generado por f y g, es decir $I = \langle f, g \rangle$, es el mismo que el generado por g y $r = f - q \cdot g$ con lo que tenemos $I = \langle f, g \rangle = \langle f - q \cdot g, g \rangle$. Esto nos resultará muy útil en los conceptos que se presentarán a continuación.

Definición 1.3.6 (Reducción)**.** Sean $f, g, r \in k[x]$ con $g \neq 0$, en cada iteración del algoritmo 1

diremos que r es una *reducción* de f por g, denotado por

$$f \xrightarrow{g} r.$$

Una sucesión de reducciones se denotará por

$$f \xrightarrow{g}_{+} r.$$

Ejemplo 1.3.6. La división del ejemplo anterior se puede expresar como una cadena de reducciones:

$$4x^3 + 2x^2 - 16 \xrightarrow{2x^2} 2x^2 - 16 \xrightarrow{2x^2} -16.$$

Teorema 1.3.4. *[4] Todo ideal de $k[x]$ tiene un conjunto generador de un solo elemento. Además dicho elemento es único salvo un múltiplo no nulo constante.*

Demostración. Sea I un ideal no nulo de $k[x]$. Sea f un polinomio no nulo del mínimo grado contenido en I. La hipótesis es $I = \langle f \rangle$. Es trivial ver que $\langle f \rangle \subset I$ puesto que I es un ideal. Por otro lado, sea $g \in I$, por el teorema 1.3.3 tenemos que $g = qf + r$, con $r = 0$ o $gr(r) < gr(f)$. Puesto que I es ideal, $qf \in I$, y por lo tanto $r = g - qf$. Si $r \neq 0$ tenemos que $grd(r) < gr(f)$ lo que contradice nuestra selección de f. Por lo que $r = 0$ y entonces $g = qf \in \langle f \rangle$. □

Gracias a la demostración del teorema 1.3.4 sabemos que el conjunto generador de un solo elemento de un ideal de $k[x]$ es el polinomio de grado mínimo contenido en el ideal. Sin embargo esto no nos facilita el cálculo de dicho polinomio, ya que esto requeriría comprobar el grado de todos los polinomios del ideal. Es por esto que introducimos el concepto de máximo común divisor de dos polinomios, que como veremos nos ayudará a encontrar el polinomio buscado.

Definición 1.3.7 (Máximo común divisor)**.** El *máximo común divisor* de los polinomios $f, g \in k[x]$ es un polinomio $h \in k[x]$ que cumple:

- h divide a f y g.
- Si $p \in k[x]$ divide a f y g, entonces p divide a h.
- h es mónico.

Se denotará a h por $mcd(f, g)$.

Proposición 1.3.2. *[3] Sea $f_1, f_2 \in k[x]$ con alguno no nulo. Entonces existe $mcd(f_1, f_2)$ y $\langle f_1, f_2 \rangle = \langle mcd(f_1, f_2) \rangle$.*

Demostración. Por el 1.3.4, existe $g \in k[x]$ tal que $\langle f_1, f_2 \rangle = \langle g \rangle$. Y dado que g es único salvo por un múltiplo constante, asumimos que este múltiplo es 1. Ahora demostraremos que $g = \langle f_1, f_2 \rangle$. Puesto que $f_1, f_2 \in \langle g \rangle$, g divide tanto a f_1 como a f_2. Sea $h \in k[x]$ otro polinomio que divida a f_1 y f_2. Dado que g está en el ideal $\langle f_1, f_2 \rangle$, tienen que exister $u_1, u_2 \in k[x]$ tales que $g = u_1 f_1 + u_2 f_2$. Por lo que h divide a g. □

Habiendo introducido el máximo común divisor de dos polinomios y conociendo algunas propiedades suyas, es fácil percatarse que el polinomio de mínimo grado que buscabamos es en realidad el mcd. Entonces si tenemos un algoritmo para calcular el mcd podremos encontrar el conjunto generador de un solo elemento del que hablabamos antes. Este algoritmo es el *algoritmo de Euclides*, y hace uso del siguiente lema.

Lema 1.3.1. *Sea $f_1, f_2 \in k[x]$ con alguno no nulo. Entonces $mcd(f_1, f_2) = mcd(f_1 - qf_2, f_2)$ para todo $q \in k[x]$.*

Demostración. Tenemos que $\langle f_1, f_2\rangle = \langle f_1 - qf_2, f_2\rangle$. Y por la proposición 1.3.2,

$$\langle mcd(f_1, f_2)\rangle = \langle f_1, f_2\rangle = \langle f_1 - qf_2, f_2\rangle = \langle mcd(f_1 - qf_2, f_2)\rangle.$$

□

Ahora ya tenemos los conocimientos necesarios para hacer uso del *Algoritmo de Euclides*. Que se presenta a continuación.

Algoritmo 2: Algoritmo de Euclides

Entrada: $f_1, f_2 \in k[x]$ con f_1 o $f_2 \neq 0$
Resultado: $f = mcd(f_1, f_2)$
Inicialización: $f := f_1, g := f_2$
mientras $g \neq 0$ **hacer**
  $f \xrightarrow{g}_+ r$;
  $f := g$;
  $g := r$;
fin
$f := \frac{1}{lc(f)} f$;

Ejemplo 1.3.7. Sea el ideal $I = \langle x^3 + 4x^2 + 1, x + 4\rangle$, si queremos encontrar otro conjunto de generadores de I que tenga un solo elemento nos basta con aplicar el algoritmo de Euclides para calcular el *mcd* de $x^4 + 2x^2 + 1$ y $2x^3 + 2x$:

INICIALIZACIÓN: $f := x^4 + 2x^2 + 1, g := 2x^3 + 2x$

Primera iteración $(g \neq 0))$

$$x^4 + 2x^2 + 1 \xrightarrow{2x^3+2x}_+ x^2 + 1$$
$$f := 2x^3 + 2x$$
$$g := x^2 + 1$$

Segunda iteración $(g \neq 0)$

$$2x^3 + 2x \xrightarrow{x^2+1}_+ 0$$
$$f := x^2 + 1$$
$$g := 0$$

Final del bucle $(g = 0)$

$$f := x^2 + 1$$

Con lo que tenemos $I = \langle x^3 + 4x^2 + 1, x + 4\rangle = \langle x^2 + 1\rangle$. Es lógico preguntarse que ocurre si tenemos un ideal generado por más de dos polinomios en $k[x]$, conceptualmente la idea es la misma, pero hay que generalizar un poco la definición de *mcd* y algunas de sus propiedades.

Definición 1.3.8. El *máximo común divisor* de los polinomios $f_1, ..., f_s \in k[x]$ es un polinomio h tal que:

- h divide a $f_1, ..., f_s$.
- Si $g \in k[x]$ divide a $f_1, ..., f_s$, entonces g divide a h.
- h es mónico.

Proposición 1.3.3. *Sean $f_1, ..., f_s$ polinomios de $k[x]$. Entonces*

- $\langle f_1, ...f_s \rangle = \langle mcd(f_1, ..., f_s) \rangle$.
- *Si $s \geq 3$, entonces $mcd(f_1, ..., f_s) = mcd(f_1, mcd(f_2, ...f_s))$.*

Ejemplo 1.3.8. Sea $I = \langle f_1, ..., f_s \rangle$ un ideal en $k[x]$ y $f \in k[x]$ un polinomio. Si quisieramos saber si $f \in I$ tendríamos que calcular un conjunto generador de I de un solo elemento, es decir $\langle mcd(f_1, ..., f_s) \rangle$ y después dividir f por dicho *mcd*. Es fácil ver que $f \in I$ si y solo si el resto de la división es 0.

1.4. Variedades e Ideales

En esta sección se procederá a enlazar los conceptos introducidos anteriormente con otros campos de las matemáticas y se intentará presentar la necesidad de un concepto matemático que resultarán ser las bases de Gröbner.

Definición 1.4.1 (Variedad)**.** Sea S un subconjunto de $k[x_1, ..., x_n]$, definimos

$$V(S) = \{(a_1, ..., a_n) \in k^n \mid f(a_1, ..., a_n) = 0 \text{ para todo } f \in S\}$$

Ejemplo 1.4.1. La variedad $V(y - x^2, y - 2)$ es la intersección de la parábola $y = x^2$ con la recta $y = 2$. Es decir, $V(y - x^2, y - 2) = \{(-\sqrt{2}, 2), (\sqrt{2}, 2)\}$ tal y como se muestra en la siguiente figura.

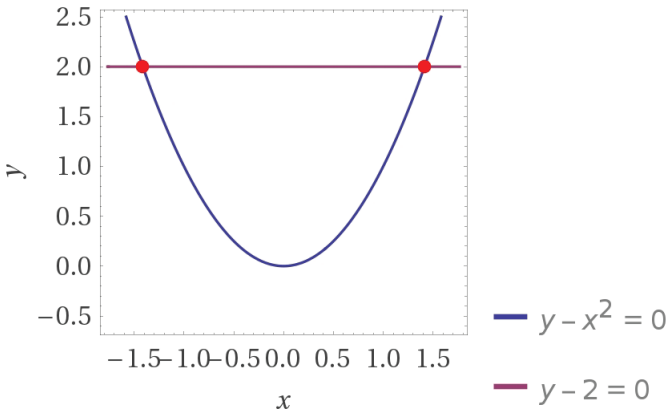


Figura 1.1: Ejemplo de variedad.

Sea $I = \langle f_1, ..., f_s \rangle$ consideramos $V(I)$, es decir, las soluciones del sistema de ecuaciones polinómicas

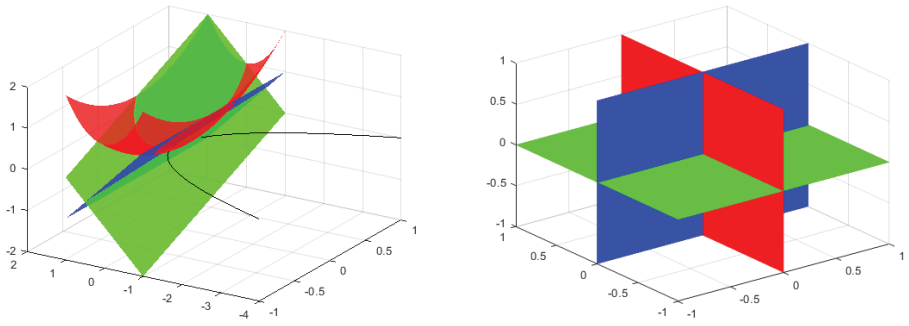
$$f = 0, f \in I. \tag{1.1}$$

Comparándolas con las soluciones de

$$f_1 = 0, f_2 = 0, ..., f_s = 0. \tag{1.2}$$

Se ve que una solución del sistema (1.1) es también solución de (1.2) ya que $\{f_1, ..., f_s\} \in I$. También si tenemos una solución de (1.1) tenemos que es una solución de (1.2) puesto que para todo $f \in I$ tenemos que $f = u_1 f_1 + \cdots + u_s f_s$, con $u_i \in k[x_1, ..., x_n]$. Por lo que $V(I) = V(f_1, ..., f_s)$.

Ejemplo 1.4.2. Sea $I = \langle x + y - z, 2x + y^2 - z, z + y, 4x^2 + y, x^2 + y^2 - z \rangle \subset k[x, y, z]$ tenemos que $I' = \langle x, y, z \rangle$ genera el mismo ideal, por lo que $I = I'$. Además, como vimos antes, $V(I) = V(x + y - z, 2x + y^2 - z, z + y, 4x^2 + y, x^2 + y^2 - z)$ y como $I = \langle x + y - z, 2x + y^2 - z, z + y, 4x^2 + y, x^2 + y^2 - z \rangle = \langle x, y, z \rangle$ tenemos que $V(I) = V(x + y - z, 2x + y^2 - z, z + y, 4x^2 + y, x^2 + y^2 - z) = V(x, y, z)$. Es fácil apreciar en la figura 1.2 que es más fácil encontrar las soluciones de la variedad $V(x, y, z)$ (figura 1.2b) que de la variedad generada por el primer conjunto que hemos definido (figura 1.2a).



(a) (b)

Figura 1.2: Superficies asociadas a la variedad

Es razonable pues concluir que una variedad está determinada por un ideal y no por un conjunto de polinomios. Antes hemos visto que una variedad V tiene el mismo conjunto de soluciones que la variedad determinada por el ideal que genera los polinomios de V. Y el teorema de la base de Hilbert (1.3.1), nos da la razón al considerar la variedad determinada por un ideal $I \subset k[x_1, ..., x_n]$. Y aunque un ideal no trivial I contenga infinitos polinomios, este puede ser determinado por un conjunto generador finito, que a su vez definirá la variedad $V(I)$ asociada al ideal. Recogemos este resultado en la siguiente proposición.

Proposición 1.4.1. *Sea I un ideal de $k[x_1, ..., x_n]$. Entonces, $V(I)$ es una variedad, además si $I = \langle f_1, ..., f_s \rangle$, se tiene que $V(I) = V(f_1, ..., f_s)$.*

Entonces si conseguimos encontrar un conjunto generador del ideal que cumpla unas características que nos sean convenientes, tendremos una mejor representación de la variedad. Lo interesante es que en $k[x]$ ya sabemos encontrar dicha representación, ilustraremos el procedimiento con el siguiente ejemplo.

Ejemplo 1.4.3. Sea la variedad determinada por el siguiente sistema

$$\begin{cases} x^5 - 3x^4 - 4x^3 = 0 \\ x^2 + 2x + 1 = 0 \\ 10x^2 + 10x = 0 \end{cases}$$

Consideremos el Ideal generado por $f_1 = x^5 - 3x^4 - 4x^3, f_2 = x^2 + 2x + 1, f_3 = 10x^2 + 10x$ es decir,$I = \langle f_1, f_2, f_3 \rangle$. Queremos encontrar un conjunto generador que sea más fácil de

resolver, como ya vimos podemos buscar un sistema generador de un solo elemento. Para ello calcularemos el *mcd* de los tres polinomios que generan I, sabiendo que $mcd(f_1, f_2, f_3) = mcd(mcd(f_1, f_2), f_3)$ y utlizando el algoritmo de Euclides.

$$\text{INICIALIZACIÓN: } f := x^5 - 3x^4 - 4x^3, g := x^2 + 2x + 1$$

$$\text{Primera iteración } (g \neq 0))$$

$$x^5 - 3x^4 - 4x^3 \xrightarrow{x^2+2x+1}_+ 5x + 5$$
$$f := x^2 + 2x + 1$$
$$g := 5x + 5$$

$$\text{Segunda iteración } (g \neq 0))$$

$$x^2 + 2x + 1 \xrightarrow{5x+5}_+ 0$$
$$f := 5x + 5$$
$$g := 0$$

$$\text{Final del bucle } (g = 0)$$

$$f := x + 1$$

Por lo que $I = \langle x + 1, f_3 \rangle$ y volviendo a utlizar el algoritmo de Euclides.

$$\text{INICIALIZACIÓN: } f := 10x^2 + 10x, g := x + 1$$

$$\text{Primera iteración } (g \neq 0))$$

$$10x^2 + 10x \xrightarrow{x+1}_+ 0$$
$$f := x + 1$$
$$g := 0$$

$$\text{Final del bucle } (g = 0)$$

$$f := x + 1$$

Con lo que hemos obtenido que $I = \langle f_1, f_2, f_3 \rangle = \langle x + 1 \rangle$ y por lo tanto la variedad también queda determinada por

$$x + 1 = 0,$$

cuya solución es mucho más sencilla, tenemos que $V(f_1, f_2, f_2) = V(I) = V(x + 1) = \{(-1)\}$.

Todo lo anterior se puede ver también de otra manera. Si en vez de considerar los puntos de k^n que son solución de un sistema de polinimios dado, consideremos todos los polinimios que tienen como solución a dichos puntos llegamos a la siguiente definición.

Definición 1.4.2. Sea V un un conjunto de puntos de k^n, diremos que el ideal $I(V)$ es el conjunto de polinimios de $k[x_1, ..., x_n]$ definido como

$$I(V) = \{f \in k[x_1, ..., x_n] | f(a_1, ..., a_n) = 0 \text{ para todo } (a_1, ...a_n) \in V\}.$$

Esta definición es muy interesante, ya que nos permite trazar un puente de las variedades a los ideales. Además por el teorema de la base de Hilbert 1.3.1 sabemos que el ideal $I(V)$ posee

un conjunto generador finito. De donde tenemos que la variedad está determinada por un conjunto finito de polinimios del ideal $I(V)$. En el capítulo 4 veremos con mayor detalle las repercusiones que tiene esto.

Continuando con el ejemplo anterior, resulta oportuno preguntarse si estos conceptos se pueden generalizar a ideales en $k[x_1, ..., x_n]$, de hecho ya conocemos un método para encontrar una representación más conveniente en el caso de que los polinomios sean lineales. Este método es la reducción a la forma canónica de matrices. Veámoslo con un par de ejemplos.

Ejemplo 1.4.4. Sea la variedad determinada por el siguiente sistema:

$$\begin{cases} 3x + 6y + z = 0 \\ 9x + 3z = 0 \end{cases} \tag{1.3}$$

Realizamos reducciones por filas en la matriz asociada al sistema:

$$\begin{pmatrix} 3 & 6 & 1 \\ 9 & 0 & 3 \end{pmatrix} \xrightarrow{f_1 - \frac{1}{3}f_2} \begin{pmatrix} 0 & 1 & 0 \\ 9 & 0 & 3 \end{pmatrix}.$$

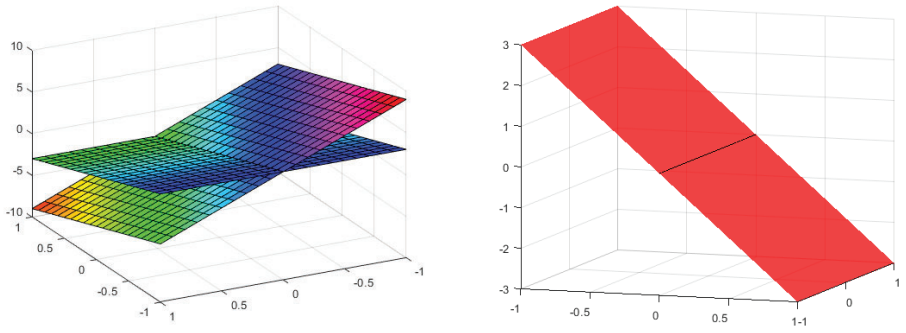
Por lo que las soluciones del sistema 1.3 son las mismas que las del siguiente

$$\begin{cases} 6y = 0 \\ 9x + 3z = 0 \end{cases} \tag{1.4}$$

Cuya obtención resulta mucho más sencilla. Sean $f_1 = 3x + 6y + z, f_2 = 9x + 3z$ y $f_3 = 6y$, veamos que el ideal $I = \langle f_1, f_2 \rangle$ es el mismo que el generado por f_2 y f_3. Tenemos que $f_3 = f_1 - \frac{1}{3}f_3$ por lo que $f_1 \in I$ por otro lado $f_1 = f_3 + \frac{1}{3}f_3$ lo cual implica que $f_1 \in \langle f_2, f_3 \rangle$. Lo más interesante es que podemos interpretar la simplificación hecha como el resto de una división. Dividimos f_1 por f_2, para ello escogemos el primer término de f_2, que es $9x$, para eliminar el primer término de f_1. Tras realizar esto vemos que no podemos reducir más f_1 por lo que el resto es f_3. Veámoslo visualmente expresado como una división larga.

$$\begin{array}{rccc|cc} & 3x & +y & +z & 9x & +3z \\ - & -3x & & -z & \frac{1}{3} & \\ \hline & & y & & & \end{array}$$

Es fácil observar que es muy similar a lo que haciamos en una sola variable, veamos ahora otro ejemplo un poco más complejo.



(a) Sistema 1.3. (b) Sistema 1.4.

Figura 1.3: Visualización de dos sistemas que determinan la misma variedad.

Ejemplo 1.4.5. Sean $f_1 = 2x + 6y + z, f_2 = 2y$ y $f_3 = x + y$ polinomios en $\mathbb{R}[x, y, z]$. Consideramos el ideal $I = \langle f_1, f_2, f_3 \rangle$ y la variedad $V(f_1, f_2, f_3)$ es decir las soluciones del sistema

$$\begin{cases} 2x + 6y + z = 0 \\ 2y = 0 \\ x + y = 0 \end{cases} \tag{1.5}$$

Aplicamos reducciones por filas:

$$\begin{pmatrix} 2 & 6 & 1 \\ 0 & 2 & 0 \\ 1 & 1 & 0 \end{pmatrix} \xrightarrow{f_1 - 2f_3} \begin{pmatrix} 0 & 4 & 1 \\ 0 & 2 & 0 \\ 1 & 1 & 0 \end{pmatrix} \xrightarrow{f_1 - 2f_2} \begin{pmatrix} 0 & 0 & 1 \\ 0 & 2 & 0 \\ 1 & 1 & 0 \end{pmatrix}.$$

Con lo que obtenemos un nuevo conjunto generador de I, $I = \langle f_1, f_2, f_3 \rangle = \langle f_2, f_3, y \rangle$. Al igual que en el ejemplo anterior podemos entender el proceso de reducción por filas como una división, con la excepción de que esta vez dividimos primero por f_3 y el resto de esta división la dividimos por f_2. Expresado como reducciones quedaría de la siguiente forma:

$$f_1 \xrightarrow{f_3} 4y + z \xrightarrow{f_2} z.$$

Pero todavía podriamos seguir reduciendo la matriz hasta alcanzar la forma escalanolada reducida por filas:

$$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 2 & 0 \\ 1 & 1 & 0 \end{pmatrix} \xrightarrow{f_3 - \frac{1}{2} f_2} \begin{pmatrix} 0 & 0 & 1 \\ 0 & 2 & 0 \\ 1 & 0 & 0 \end{pmatrix} \sim \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

De donde obtenemos el sistema

$$\begin{cases} x = 0 \\ y = 0 \\ z = 0 \end{cases} \tag{1.6}$$

La metodología llevada a cabo es muy similar a la que realizábamos para calcular el *mcd* de dos polinomios en $k[x]$, calculando en cada paso el polinomio resultante de dividir dos polinomios del sistema inicial. En el último paso convertimos todos los términos principales en mónicos, ya que esto lo exige la forma reducida por filas, obsérvese la equivalencia al *mcd* en el cual también imponiamos que este fuese mónico (coeficiente igual a 0).

Hemos visto como se resolvería si fuesen polinomios lineales pero ¿y en general?, para encontrar este conjunto generador expondremos en el capítulo 2 una generalización de la división en anillos de polinimios y en el capítulo 3 introduciremos lo equivalente al máximo común divisor, que serán las *bases de Gröbner*.

Capítulo 2

División en $k[x_1, ..., x_n]$.

2.1. Orden en anillos de polinomios

En una variable vimos que si queremos dividir $f_1 = 4x^4 + 2x^2 + x$ por $f_2 = x^2 + x$ escogemos el término principal de f_2, esto es x^2 y a partir de él reducimos. Ahora bien, en varias variables la situación cambia, dados $f = x^2y^2 + x^4 + y^2, g = xy + x^2 + y^2 \in \mathbb{R}[x, y]$,si dividimos f por g ¿qué término de g es el principal?, los tres poseen el mismo grado, y ¿qué variable tiene mayor prioridad x o y? Puesto que todavía no sabemos dar respuesta a estas preguntas, antes de introducir la división multivariable será necesario definir un orden en los polinomios.

Definición 2.1.1. Llamaremos a $x_1^{\beta_1} \cdots x_n^{\beta_n}$ *producto de potencias* y el conjunto de todos los productos de potencias se denotará por

$$\mathbb{T}^n = \{x_1^{\beta_1} ... x_n^{\beta_n} \mid \beta_i \in \mathbb{N}, i = 1, ...n\}.$$

Nota. Para simplificar la notación, denotaremos a $x_1^{\beta_1} \cdots x_n^{\beta_n}$ por x^β donde $\beta = (\beta_1, ..., \beta_n) \in \mathbb{N}^n$.

En la división de una sola variable seleccionábamos primero el término de mayor grado, en varias variables será conveniente también seleccionar un término con mayor prioridad, está prioridad vendrá dada por los *órdenes monomiales*. Y la selección que hacíamos al dividir en $k[x]$ es el resultado de utilizar un caso particular de un orden monomial.

Definición 2.1.2 (Orden monomial)**.** Un orden monomial en $\mathbb{T}^n$ es un orden total $<$ en $\mathbb{T}^n$ que satisface:

1. $1 < x^\beta$ para todo $x^\beta \in \mathbb{T}^n$, con $x^\beta \neq 1$.
2. Si $x^\alpha < x^\beta$, entonces $x^\alpha x^\gamma < x^\beta x^\gamma$ para todo $x^\gamma \in \mathbb{T}^n$.

Ahora que ya tenemos la definición, pasamos a presentar algunos de los órdenes más comunes, que además se usarán a lo largo del documento.

Definición 2.1.3 (Orden lexicográfico)**.** Fijamos un orden en las variables $x_1 > x_2 > ... > x_{n-1} > x_n$. Sean $\alpha = (\alpha_1, ..., \alpha_n)$, $\beta = (\beta_1, ..., \beta_n) \in \mathbb{N}^n$. Diremos que $x^\alpha <_{lex} x^\beta$ si:

> las primeras coordenadas α_i, β_i por la izquierda, diferentes, cumplen $\alpha_i < \beta_i$.

Denotaremos al *orden lexicográfico* por lex.

Ejemplo 2.1.1. Sea x^α y x^β con $\alpha = (1,0,0)$ y $\beta = (0,1,1)$ y sea el orden *lex*. Entonces $x^\beta < x^\alpha$. Nótese que x^α y x^β también pueden ser expresados como $x^\alpha = x_1$ y $x^\beta = x_2x_3$

Ejemplo 2.1.2. En $k[x,y,z]$ con el orden *lex* y donde hemos fijado $x > y > z$ tenemos

$$1 < z < z^2 < z^3 < \cdots < y < yz < y^2 < \cdots < x < x^2 < x^2y < x^2yz \cdots .$$

Sin embargo si fijamos $y < x < z$ tenemos

$$1 < y < y^2 < \cdots < x < x^2 < x^2y < \cdots < z < yz < x^2yz < z^2 < z^3 < \cdots .$$

Nota. En $\mathbb{T}^n$ con el orden *lex* se pueden fijar $n!$ órdenes diferentes a las variables. Además, como se ve en el ejemplo 2.1.2 el orden *lex* depende del orden que se fije entre las variables, por ello siempre se deberá especificar un orden sobre las variables.

Definición 2.1.4 (Orden grado lexicográfico)**.** Asumiendo $x_1 > x_2 > ... > x_{n-1} > x_n$. Sean $\alpha = (\alpha_1, ..., \alpha_n),\ \beta = (\beta_1, ..., \beta_n) \in \mathbb{N}^n$. Decimos que $x^\alpha <_{grlex} x^\beta$ si:

$$\sum_{i=1}^{n} \alpha_i < \sum_{i=1}^{n} \beta_i, \qquad \text{o} \qquad \sum_{i=1}^{n} \alpha_i = \sum_{i=1}^{n} \beta_i \text{ y } x^\alpha <_{lex} x^\beta.$$

Denotaremos al *orden grado lexicográfico* por *grlex*.

Ejemplo 2.1.3. En $k[x,y,z]$ con el orden *grlex* y con $z < y < x$ tenemos

$$1 < z < y < x < z^2 < y^2 < x^2 < \cdots < xy^2 < xy^2z < \cdots < x^4 < \cdots .$$

Si embargo si las variables tuviesen el orden $x < y < z$ tendriamos

$$1 < x < y < z < y^2 < x^3 < x^2y < z^3 < z^4 < xy^3 < x^2yz < \cdots .$$

Definición 2.1.5 (Orden grado lexicográfico reverso)**.** Fijando el orden $x_1 > x_2 > ... > x_{n-1} > x_n$ entre las variables. Sean $\alpha = (\alpha_1, ..., \alpha_n),\ \beta = (\beta_1, ..., \beta_n) \in \mathbb{N}^n$. Decimos que $x^\alpha <_{grevlex} x^\beta$ si:

$$\sum_{i=1}^{n} \alpha_i < \sum_{i=1}^{n} \beta_i \qquad \text{o} \qquad \begin{array}{l} \sum_{i=1}^{n} \alpha_i = \sum_{i=1}^{n} \beta_i \text{ y las primeras coordenadas } \alpha_i, \beta_i \text{ de } \alpha \\ \text{y } \beta \text{ por la derecha, diferentes, cumplen } \alpha_i > \beta_i \end{array} .$$

Denotaremos al *orden grado lexicográfico reverso* por *grevlex*.

Ejemplo 2.1.4. En $k[x,y,z]$ con el orden *grevlex* y con $z < y < x$ tenemos

$$1 < z < y < x < y^2 < z^3 < \cdots < x^2y < \cdots < x^3 < xy^3 < \cdots .$$

Proposición 2.1.1. *[4] Los órdenes lex, grlex y grevlex son órdenes monomiales.*

Elegido un órden monomial en $k[x_1, ...x_n]$ tenemos que para todo $f \in k[x_1, ...x_n], f \neq 0$ podemos escribir f de la siguiente manera

$$f = a_1x^{\alpha_1} + a_2x^{\alpha_2} + ... + a_{n-1}x^{\alpha_{n-1}} + a_nx^{\alpha_n}$$

con $0 \neq a_i \in k, \alpha \in \mathbb{N}$ y $x^{\alpha_i} \in \mathbb{T}^n$ y $x^{\alpha_1} > x^{\alpha_2} > ... > x^{\alpha_{n-1}} > x^{\alpha_n}$.
Siempre que sea posible se utilizará esta forma.

En una variable definimos el término principal como aquel que lideraba el polinomio una vez los términos de este estaban ordenados según sus potencias. En $k[x_1, ..., x_n]$ la definición será equivalente, elegido un orden monomial, y escribiéndolo de la forma descrita anteriormente, se escogerá el primero.

Definición 2.1.6. Sea $f = a_1x^{\alpha_1} + a_2x^{\alpha_2} + ... + a_{n-1}x^{\alpha_{n-1}} + a_nx^{\alpha_n} \in k[x_1, ...x_n]$ ordenado con un orden monomial definimos:

- $lt(f) = a_1x^{\alpha_1}$, el *término principal* de f.
- $lp(f) = x^{\alpha_1}$, el *producto de potencias principal* de f.
- $lc(f) = a_1$ el *coeficiente principal* de f.

Ejemplo 2.1.5. Sea $f = 2x^2y^2z^3 + x^6 + 7y^5z - 3xy \in \mathbb{R}[x, y, z]$ donde fijamos el orden $x > y > z$ en las variables

Con el orden *lex* tenemos que el polinomio f ordenado es$f^6 + 2x^2y^2z^3 - 3xz + 7y^5z$

- $lt(f) = x^6$
- $lp(f) = x^6$
- $lc(f) = 1$

Con el orden *grlex* tenemos que el polinomio f ordenado es $2x^2y^2z^3 + x^6 + 7y^5z - 3xz$

- $lt(f) = 2x^2y^2z^3$
- $lp(f) = x^2y^2z^3$
- $lc(f) = 2$

CCon el orden *grevlex* tenemos que el polinomio f ordenado es $22x^2y^2z^3 + x^6 + 7y^5z - 3xz$

- $lt(f) = 2x^2y^2z^3$
- $lp(f) = x^2y^2z^3$
- $lc(f) = 2$

Ahora ya sabemos como seleccionar los términos al dividir dos polinomios de $k[x_1, ..., x_n]$ pero, ¡falta lo más importante!, la división multivariable en sí misma. Esta será introducida en la siguiente sección.

2.2. División en anillos de polinomios

Está sección está dedica a explicar la división en el caso de varias variables, aportando sendos ejemplos cuando así se estima oportuno. Primero se darán algunas definiciones y resultados necesarios para el algoritmo de división multivariable, que será presentado a continuación. Por último se tratará de enlazar mediante ejemplos lo visto al final del capítulo 1 con lo que se introducirá en el capítulo 3, es decir, las bases de Gröbner.

Proposición 2.2.1. *[3] Sean $x^\alpha, x^\beta \in \mathbb{T}^n$, si x^α divide a x^β entonces $x^\alpha \leq x^\beta$.*

Demostración. Por hipótesis existe un $x^\gamma \in \mathbb{T}^n$ tal que $X^\beta = x^\alpha x^\gamma$. Y por la primera condición de la definición de orden monomial, tenemos que $x^\gamma \geq 1$, y por la segunda condición tenemos que $x^\beta = x^\alpha x^\gamma \geq x^\alpha$. □

Definición 2.2.1 (Polinomio reducible)**.** Sean $f, g, h \in k[x_1, ..., x_n]$ con $g \neq 0$, decimos que f es *reducible* a h módulo g en un paso, y lo denotamos por

$$f \xrightarrow{g} h,$$

si $lp(g)$ divide a un término aX, con $0 \neq a \in k$ y $X \in \mathbb{T}^n$, de f y se cumple

$$h = f - \frac{aX}{lt(g)}g.$$

Ejemplo 2.2.1. Sean dos polinomios $f = x^2y + 10xy + 4y^2 + x$, $g = 2x + 2y \in \mathbb{Q}[x, y]$ con el orden *grlex* y fijando $x > y$. Utilizando la división larga de polinomios se siguen los siguientes pasos:

$$\begin{array}{rllll|ll} & x^2y & +10xy & +4y^2 & +x & 2x & +2y \\ \cline{6-7} - & x^2y & +xy^2 & & & \frac{1}{2}xy & \\ \cline{1-5} & -xy^2 & +10xy & +4y^2 & +x & & \end{array}$$

Donde el resto r es $r = -xy^2 + 10xy + 4y^2 + x = f\ \frac{lt(f)}{lt(g)}g$, y el cociente es la fracción $\frac{lt(f)}{lt(g)} = \frac{1}{2}xy$. Nótese la similaridad con la división en $k[x]$. La selección que se realiza lleva el mismo criterio. Bien, si ahora dividimos r por g, utilizando también los términos principales de los dos polinomios se tiene:

$$\begin{array}{rllll|ll} & -xy^2 & +10xy & +4y^2 & +x & 2x & +2y \\ \cline{6-7} - & -xy^2 & -y^3 & & & -\frac{1}{2}y^2 & \\ \cline{1-5} & y^3 & +10xy & +4y^2 & +x & & \end{array}$$

Y si hacemos esto sucesivamente, expresándolo todo una sola división, tenemos:

$$\begin{array}{rllll|lll} & x^2y & +10xy & +4y^2 & +x & 2x & +2y & \\ \cline{6-8} - & x^2y & +xy^2 & & & \frac{1}{2}xy & -\frac{1}{2}y^2 & +5y \quad +\frac{1}{2} \\ \cline{1-5} & -xy^2 & +10xy & +4y^2 & +x & & & \\ - & -xy^2 & -y^3 & & & & & \\ \cline{1-5} & y^3 & +10xy & +4y^2 & +x & & & \\ - & & 10xy & +10y^2 & & & & \\ \cline{1-5} & y^3 & -6y^2 & +x & & & & \\ - & & & x & +y & & & \\ \cline{1-5} & y^3 & -6y^2 & -y & & & & \end{array}$$

Que puede ser también expresada como la siguiente cadena de reducciones.

$$f \xrightarrow{g} -xy^2 + 10xy + 4y^2 + x \xrightarrow{g} y^3 + 10x + y4y^2 + x \xrightarrow{g} y^3 - 6y^2 + x \xrightarrow{g} y^3 - 6y^2 - y$$

La idea de reducir un polinomio por otro en varias variables puede resultar más o menos intuitiva una vez conocido la división en una sola variable, pero reducir un polinomio por un conjunto de polinomios es una idea un poco más alienígena. El algoritmo de división multivariable nos permitirá hacerlo, pero primero hemos de dar algunas definiciones más para entender lo que hace.

Definición 2.2.2 (Polinomio reducible por un conjunto)**.** Sean f, h y $f_1, ..., f_s \in k[x_1, ..., x_n]$ con $f_i \neq 0$ y sea $F = \{f_1, ..., f_s\}$, diremos que f es *reducible* a h módulo F, y lo denotaremos por

$$f \xrightarrow{F}_+ h$$

si existe una secuencia de índices $i_1, i_2, ..., i_t \in \{1, ..., s\}$ y una secuencia de polinomios $h_1, ..., h_{t-1} \in k_n[x_1, ..., x_n]$ tal que

$$f \xrightarrow{f_{i_1}} h_1 \xrightarrow{f_{i_2}} h_2 \xrightarrow{f_{i_3}} ... \xrightarrow{f_{i_{t-1}}} h_{t-1} \xrightarrow{f_{i_t}} h.$$

Definición 2.2.3 (Polinomio reducido)**.** Decimos que un polinomio r *está reducido respecto* de $F = \{f_1, ..., f_s\}$ si no es reducible por F.

Definición 2.2.4 (Resto)**.** Si $f \xrightarrow{F}_+ r$ y r está reducido respecto de F, se llama a r resto de f respecto de F.

Llegado este punto ya conocemos las herramientas fundamentales para poder introducir el algoritmo principal de este capítulo, el *algoritmo de división multivariable*. Cabe destacar que la heurística que usa es muy similar a lo que hacíamos en una sola variable, como ya se pudo anticipar en el ejemplo 2.2.1.

Algoritmo 3: Algoritmo de división multivariable

Entrada: $f, f_1, ..., f_s \in k[x_1, ..., x_n]$ con $f_i \neq 0$ $(1 \leq i \leq s)$
Resultado: $u_1, ..., u_s, r$ tales que $f = u_1 f_1 + ...u_s f_s + r$ y r está reducido respecto de $\{f_1, ..., f_s\}$ y $max(lp(u_1)lp(f_1), ..., lp(u_s)lp(f_s), lp(r)) = lp(f)$.
Inicialización: $u_1 := 0, u_2 := 0, ..., u_s := 0, r := 0, h := f$
mientras $h \neq 0$ **hacer**
  si *existe i tal que* $lp(f_i)$ *divide a* $lp(h)$ **entonces**
    elegir el mínimo i tal que $lp(f_i)$ divida a $lp(h)$;
    $u_i := u_i + \frac{lt(h)}{lt(f_i)}$;
    $h := h - \frac{lt(h)}{lt(f_i)} f_i$;
  en otro caso
    $r := r + lt(h)$;
    $h := h - lt(h)$;
  fin
fin

Teorema 2.2.1 (Teorema de división multivariable). *[3, pág. 30] Sea un conjunto de polinomios no nulos $F = \{f_1, ..., f_s\}$ y f en $k[x_1, ..., x_n]$ el algoritmo 3 (División multivariable) obtiene polinomios $u_1, ..., u_s, r \in k[x_1, ..., x_n]$ tal que*

$$f = u_1 f_1 + ... + u_s f_s + r$$

con r reducido respecto de F y

$$lp(f) = max(max(lp(u_1)lp(f_i)), lp(r)), \quad (1 \leq i \leq s)$$

Demostración. Se pueden encontrar demostraciones equivalentes en [4] y en [3]. □

Ejemplo 2.2.2. Vamos a volver a realizar el ejemplo 2.2.1 utilizando el algoritmo de división multivariable. Sea $f = x^2y + 10xy + 4y^2 + x, f_1 = 2x + 2y \in \mathbb{Q}[x, y]$ y $F = \{f_1\}$ usando el orden *grlex* con $x > y$.
INICIALIZACIÓN: $u_1 := 0, r := 0, h := x^2y + 10xy + 4y^2 + x$

Primera iteración $(h \neq 0)$

$lp(f_1) = x$ divide a $lp(h) = x^2y$
  $u_1 := 0 + \frac{x^2y}{2x} = \frac{1}{2}xy$
  $h := x^2y + 10xy + 4y^2 + x - \frac{1}{2}xy(2x + 2y) = -xy^2 + 10xy + 4y^2 + x$

Segunda iteración $(h \neq 0)$

$lp(f_1) = x$ divide a $lp(h) = xy^2$
$u_1 := \frac{1}{2}xy + \frac{-xy^2}{2x} = \frac{1}{2}xy - \frac{1}{2}y^2$
$h := -xy^2 + 10xy + 4y^2 + x + \frac{1}{2}y^2(2x + 2y) = y^3 + 10xy + 4y^2 + x$

Tercera iteración $(h \neq 0)$

$lp(f_1) = x$ no divide a $lp(h) = y^3$
$r := y^3$
$h := y^3 + 10xy + 4y^2 + x - y^3 = 10xy + 4y^2 + x$

Cuarta iteración $(h \neq 0)$

$lp(f_1) = x$ divide a $lp(h) = xy$
$u_1 := \frac{1}{2}xy - \frac{1}{2}y^2 + 5y$
$h := 10xy + 4y^2 + x - 10xy - 10y^2 = -6y^2 + x$

Quinta iteración $(h \neq 0)$

$lp(f_1) = x$ no divide a $lp(h) = y^2$
$r := y^3 - 6y^2$
s $h := x$

Sexta iteración $(h \neq 0)$

$lp(f_1) = x$ divide a $lp(h) = x$
$u_1 := \frac{1}{2}xy - \frac{1}{2}y^2 + 5y + \frac{1}{2}$
$h := x - x - y = y$

Séptima iteración $(h \neq 0)$

$lp(f_1) = x$ no divide a $lp(h) = y$
$r := y^3 - 6y^2 - y$
$h := y - y = 0$

Final del bucle $(h = 0)$

$f = u_1 \cdot f_1 + r = (\frac{1}{2}xy - \frac{1}{2}y^2 + 5y + \frac{1}{2})(2x + 2y) + y^3 - 6y^2 - y$

Es interesante observar que $f - r \in I = \langle f_1 \rangle$ por lo que si $r = 0$ tenemos que f está en I. De hecho si hubiésemos escogido un conjunto generador $F = \{f_1, ..., f_s\}$ y aplicasemos el agoritmo de división, por el teorema 1.3.1 obtendríamos $f = u_1 f_1 + ... + u_s f_s + r$ de donde se ve que si r fuese nulo, $f \in \langle f_1, ..., f_s \rangle$.

Ejemplo 2.2.3. Sea el polinomio $f = x^3y^3z + x^3y^2z + x^2y^3z + x^2y^2z \in \mathbb{R}[x, y, z]$, y el ideal $I = \langle f_1, f_2, f_3, f_4 \rangle \subseteq \mathbb{R}[x, y, z]$ con $f_1 = y + 1, f_2 = xy^2 + x, f_3 = yz, f_4 = x + z$. Con el orden gradolexicográfico y con $x > y > z$.Sea $F = \{f_1, f_2, f_3, f_4\}$. Si dividimos f por F usando el algoritmo de división tenemos:
INICIALIZACIÓN: $u_1 := 0, ..., u_4 := 0, r := 0, h := x^3y^3z + x^3y^2z + x^2y^3z + x^2y^2z$

Primera iteración $(h \neq 0)$

$lp(f_1) = y$ divide a $lp(h) = x^3y^3z$
$u_1 := x^3y^2z$
$h := x^2y^3z + x^2y^2z$

Segunda iteración $(h \neq 0)$

$lp(f_1) = y$ divide a $lp(h) = x^2y^3z$
$u_1 := x^3y^2z + x^2y^2z$
$h := 0$

Final del bucle $(h = 0)$

$f = u_1 \cdot f_1$
Dado que $f \xrightarrow{F}_+ 0$, vemos que $f = (x^3y^2z + x^2y^2z) \cdot f_1$ y claramente $f \in I$. Está por lo tanto claro que si el resto es cero f pertenece al ideal. Sin embargo que el resto sea diferente de cero no implica que el polinomio no pertenezca al ideal. Por ejemplo cambiando el orden de los polinomios de F a $F' = \{f_2, f_3, f_1, f_4\}$, y por tanto el orden en el que se escogen los polinomios en el algoritmo de división, tenemos lo siguiente:
Primera iteración $(h \neq 0)$

$lp(f_2) = xy^2$ divide a $lp(h) = x^3y^3z$
$u_1 := x^2yz$
$h := x^3y^2z + x^2y^3z - x^3yz + x^2y^2z$

Segunda iteración $(h \neq 0)$

$lp(f_2) = xy^2$ divide a $lp(h) = x^3y^2z$
$u_1 := x^2yz + x^2z$
$h := x^2y^3z - x^3yz + x^2y^2z - x^3z$

Tercera iteración $(h \neq 0)$

$lp(f_2) = xy^2$ divide a $lp(h) = x^2y^3z$
$u_1 := x^2yz + x^2z + xyz$
$h := -x^3yz + x^2y^2z - x^3z - x^2yz$

Cuarta iteración $(h \neq 0)$

$lp(f_3) = yz$ divide a $lp(h) = -x^3yz$
$u_2 := -x^3$
$h := x^2y^2z - x^3z - x^2yz$

Quinta iteración $(h \neq 0)$

$lp(f_2) = xy^2$ divide a $lp(h) = x^2y^2z$
$u_1 := x^2yz + x^2z + xyz + xz$

$h := -x^3z - x^2yz - x^2z$

Sexta iteración $(h \neq 0)$

$lp(f_4) = x$ divide a $lp(h) = -x^3z$
$u_4 := -x^2z$
$h := x^2yz + x^2z^2 - x^2z$

Séptima iteración $(h \neq 0)$

$lp(f_3) = yz$ divide a $lp(h) = -x^2yz$
$u_3 := -x^3 - x^2$
$h := x^2z^2 - x^2z$

Octava iteración $(h \neq 0)$

$lp(f_4) = x$ divide a $lp(h) = x^2z^2$
$u_4 := -x^2z + xz^2$
$h := -xz^3 - x^2z$

Novena iteración $(h \neq 0)$

$lp(f_4) = x$ divide a $lp(h) = -xz^3$
$u_4 := -x^2z + xz^2 - z^3$
$h := z^4 - x^2z$

Décima iteración $(h \neq 0)$

$lp(f_i) \in F, i = 1, ...4$ no divide a $lp(h) = z^4$
$r := z^4$
$h := -x^2z$

Undécima iteración $(h \neq 0)$

$lp(f_4) = x$ divide a $lp(h) = -x^2z$
$u_4 := x^2z + xz^2 - xz - z^3$
$h := xz^2$

Duodécima iteración $(h \neq 0)$

$lp(f_4) = x$ divide a $lp(h) = xz^2$
$u_4 := -x^2z + xz^2 - xz - z^3 + z^2$
$h := z^3$

Trigésima iteración $(h \neq 0)$

$lp(f_i) \in F, i = 1, ...4$ no divide a $lp(h) = -z^3$
$r := z^4 - z^3$
$h := 0$

Final del bucle $(h = 0)$

$f = u_1 \cdot f_2 + u_2 \cdot f_3 + u_4 \cdot f_4 + z^4 - z^3$

Con lo que tenemos $f \xrightarrow{F'}_+ z^4 - z^3$, pero como hemos visto $f \in I$. Con lo que vemos que la no nulidad del resto no determina la pertenencia.

Esto ya nos ocurría en $k[x]$, y para resolverlo usábamos el *mcd*, por ello surge la necesidad de una generalización del *mcd* a $k[x_1, ..., x_n]$. Esta generalización que introduciremos en el siguiente capítulo serán las *bases de Gröbner*. Y el *mcd* es el caso particularparticular de ellas en $k[x]$.

Capítulo 3

Bases de Gröbner

Al final del primer capítulo nos encontramos con la necesidad de un concepto generalizado del mcd, y en el segundo capítulo expusimos los instrumentos preliminares que nos permitirán, al fin, introducir las *bases de Gröbner*. Este capítulo está dedicado a la definición formal de las *bases de Gröbner* acompañadas de ejemplos. Junto con el algoritmo para hallarlas y algunas de sus propiedades más importantes.

3.1. Bases de Gröbner

Definición 3.1.1 (Base de Gröbner)**.** Un conjunto no nulo de polinomios $G = \{g_1, ..., g_t\}$ contenido en un ideal I se dice que es una *base de Gröbner* de I si para todo $f \in I$ no nulo, se tiene que existe $i \in \{1, ..., t\}$ tal que $lp(g_i)$ divide a $lp(f)$.

Ejemplo 3.1.1. Sea el ideal $I = \langle x, x^2 \rangle$. Es sencillo ver que $G = \{x\}$ es una base de Gröbner de I.

Definición 3.1.2 (Ideal de términos principales)**.** Dado un subconjunto S de $k[x_1, ..., x_n]$, definimos el *ideal de términos principales* de S como el ideal

$$Lt(S) = \langle lt(s) | s \in S \rangle$$

W. W. Adams y P. Loustaunau recogen en el siguiente teorema algunas propiedades equivalentes muy interesantes de las bases de Gröbner.

Teorema 3.1.1. *[3, pág. 32] Sea I un ideal no trivial de $k[x_1, ..., x_n]$, y $G = \{g_1, ..., g_t\} \subseteq I$ un conjunto de polinomios no nulos. Entonces, son equivalentes:*

1. *G es una base de Gröbner de I.*
2. *$f \in I$ si y solo si $f \xrightarrow{G}_+ 0$.*
3. *$f \in I$ si y solo si $f = \sum_{i=1}^{t} h_i g_i$ con $lp(f) = max_{1 \leq i \leq t}(lp(h_i) \cdot lp(g_i))$.*
4. *$Lt(G) = Lt(I)$.*

Demostración. La demostración también se proporciona en [3]. □

Ejemplo 3.1.2. Sea el ideal $I = \langle x^2 - 1, x + y, y^2 - 1\rangle$ tenemos que $G = \{x + y, y^2 - 1\}$ con el orden lexicográfico es una base de Gröbner de I. Se tiene que $Lt(G) = \langle x, y^2\rangle = Lt(I) = \langle x^2, x, y^2\rangle$, ya que $x^2 = x \cdot x \in Lt(G)$, por lo que el resto de propiedades se cumple. Es interesante ver que si tomamos el orden lexicográfico con $y > x$, entonces G dejará de ser una base de Gröbner de I, ya que $Lt(G) = \langle y, y^2\rangle \neq Lt(I) = \langle x^2, y, y^2\rangle$ y $x^2 \in Lt(I)$ pero no está en $Lt(G)$.

Corolario 3.1.1. *Si $G = \{g_1, ..., g_t\}$ es una base de Gröbner para un ideal I, entonces $I = \langle g_1, ..., g_n\rangle$.*

Demostración. Es claro que $\langle g_1, ..., g_n\rangle \subseteq I$, dado que por la definición de base de Gröbner $g_i, i \in \{1, .., n\}$ está en I. Por otro lado, para todo $f \in I$ por el teorema 3.1.1 se tiene que $f \xrightarrow{G}_+ 0$, y por lo tanto $f \in \langle g_1, ..., g_n\rangle$. □

Antes de presentar el siguiete corolario necesitaremos un lema que nos de algo más de información sobre la naturaleza de las bases de Gröbner.

Lema 3.1.1. *[3] Sea I un ideal no trivial generado por un conjunto S de polinomios, y sea $f \in k[x_1, ...x_n]$. Entonces $f \in I$ si y solo si para todo término ax^α, con $a \in k$, $x^\alpha \in \mathbb{T}^n$, de f existe otro término $bx^\beta \in S$, con $b \in k$, $x^\alpha \in \mathbb{T}^n$ tal que bx^β divida ax^α. Es más, existe un subconjunto finito $S_0 \subseteq S$ tal que $I = \langle S_0\rangle$.*

Demostración. Consultar [3, pág. 33] □

Corolario 3.1.2. *[3] Todo ideal no trivial de $k[x_1, ..., x_n]$ posee una base de Gröbner.*

Demostración. Por el lema 3.1.1, el ideal de términos principales $Lt(I)$ tiene un conjunto generador finito de la forma $\{lt(g_1), ..., lt(g_t)\}$ con $g_1, ..., g_t \in I$. Sea $G = \{g_1, ..., g_t\}$, entonces tenemos que $Lt(G) = Lt(I)$, por lo que G es una base de Gröbner de I. □

Definición 3.1.3 (Base de Gröbner)**.** Diremos que un subconjunto $G = \{g_q, ..., g_t\}$ de $k[x_1, ..., x_n]$ es una base de Gröbner si y solo si es una base de Gröbner para el ideal $\langle G\rangle$ que genera.

En el ejemplo 2.2.3 se dio el caso que un mismo polinomio podía resultar en dos polinomios distintos al ser reducido por el mismo conjunto de polinomios si se alteraba el orden en el que se escogian al dividir. El siguiente teorema nos asegura que si el conjunto que divide al polinomio es una base de Gröbner, entonces el resto de la división es único, no importando así el orden en el que se divida.

Teorema 3.1.2. *[4] Sea $G = \{g_1, ..., g_t\}$ un subconjunto de polinomios no nulos de $k[x_1, ..., x_n]$. Entonces G es una base de Gröbner si y solo si para todo $f \in k[x_1, ..., x_n]$, se tiene que el resto de la división de f entre G es único.*

Ejemplo 3.1.3. Sea el polinomio $f = x^3y^3z + x^3y^2z + x^2y^3z + x^2y^2z \in \mathbb{R}[x, y, z]$, y el conjunto $F = \{y + 1, xy^2 + x, yz, x + z\}$. En ejemplo 2.2.3 vimos que con el orden grlex y con $x > y > z$ teniamos que $f \xrightarrow{F}_+ 0$, pero si alterábamos el orden al dividir, es decir, la ordenación de F, obteníamos también $f \xrightarrow{F'}_+ z^4 - z^3$. De donde por el teorema 3.1.2 tenemos que F no es una base de Gröbner.

Ejemplo 3.1.4. En el ejemplo 1.4.3 vimos que calculando el mcd de $f_1 = x^5 - 3x^4 - 4x^3, f_2 = x^2 + 2x + 1, f_3 = 10x^2 + 10x$ obteníamos que el ideal $I = \langle f_1, f_2, f_3\rangle$ es equivalente al generado por $x + 1$, es decir $I = \langle mcd(f_1, f_2, f_3)\rangle = \langle x + 1\rangle$. Es fácil ver que $G = \{x + 1\}$ es una base de Gröbner, dado que por la definición de *mcd* tenemos que $x + 1$ divide a f_1, f_2 y f_3 de donde

tenemos que $x+1$ divide a cualquier $f \in I$, y por el teorema 3.1.1 (2) $G = \{x+1\}$ es base de Gröbner.
De igual manera en el caso lineal, al calcular la forma reducida por filas obtenemos una base de Gröbner.

Ahora ya conocemos la definición formal de una base de Gröbner, y sabemos encontrarlas en algunos casos concretos pero ¿y en general? Responderemos a esta pregunta en la siguiente sección introduciendo el algoritmo de Buchberger.

3.2. Algoritmo de Buchberger

En la sección anterior dijimos que $F = \{f_1, ..., f_t\}$ es una base de Gröbner del ideal $I = \langle F \rangle$ de $k[x_1, ..., x_n]$ si para todo $f \in I$ existe $f_i \in F$ con $(1 \leq i \leq t)$ tal que $lp(f_i)$ divida a $lp(f)$. En algunos casos esto es difícil de hallar, por ejemplo, sean $f_1 = x^2 - y, f_2 = x + y$, $F = \{f_1, f_2\}$ e $I = \langle f_1, f_2 \rangle$. Tenemos que $f_3 = f_1 - x \cdot f_2 = -xy - y$, por lo que $f_3 \in I$, sin embargo $f_3 \xrightarrow{F}_+ = y^2 - y$ es diferente de cero, esto se debe a que los términos principales de f_1 y f_2 se cancelan al calcular f_3. De otra manera, si $f \in I$ entonces $f = \sum_{i=1}^{t} h_i f_i$, con $h_i \in k[x_1, ..., x_n]$, y cuando el mayor $lp(h_i f_i) = lp(h_i) lp(f_i)$ se cancela, puede resultar que $lp(f)$ no sea divisible por F, como en el ejemplo anterior. Estos polinomios los llamaremos *S-polinomios*, la 's' viene del inglés *subtraction* (resta).

Definición 3.2.1 (S-polinomio)**.** Sean $f, g \in k[x_1, ..., x_n]$ dos polinomios no nulos. Sea $L = mcm(lp(f), lp(g))$. Diremos que el polinomio

$$S(f,g) = \frac{L}{lt(f)} f - \frac{L}{lt(g)} g$$

es el *S-polinomio* de f y g.

Ejemplo 3.2.1. Expandiendo el ejemplo introducido al comienzo de la sección, vemos que $S(f_1, f_2) = S(x^2 - y, x + y) = \frac{x^2}{x^2}(x^2 - y) - \frac{x^2}{x}(x + y) = -xy - y$. Y que los porductos de potencias principales $lp(f_1) = x^2 = lp(x f_2)$ se han anulado en $S(f_1, f_2)$.

Los S-polinomios también pueden ser entendidos de otra manera. En la división de f por $F = \{f_1, ..., f_s\}$ puede darse el caso de que algún término X de f sea divisible por el término principal de dos polinomios diferentes f_i y f_j de F, por lo que f es divisible por $L = mcm(f_i, f_j)$. Entonces al reducir f por f_i llegamos al polinomio $h_1 = f - \frac{X}{lt(f_i)} f_i$, y si reducimos por f_j obtenemos el polinomio $h_2 = f - \frac{X}{lt(f_j)} f_j$. Con lo que se llega la siguiente ambigüedad $h_1 - h_2 = \frac{X}{lt(f_i)} f_i - \frac{X}{lt(f_j)} f_j = \frac{X}{L} S(f_i, f_j)$.

Ejemplo 3.2.2. Sean $f = xyz + 1, f_1 = xz - y$, $f_2 = x + yz \in \mathbb{Q}[x, y, z]$ con $x > y > z$ y con el orden lexicográfico. Tenemos que $f \xrightarrow{f_1}_+ y^2 + 1 = f - y \cdot f_1$ y $f \xrightarrow{f_2}_+ -y^2 z^2 + 1 = f - yz \cdot f_2$, y entonces $-y^2 z^2 + y^2 = S(f_1, f_2)$. que $S(f_1, f_2)$ está reducido respecto de $\{f_1, f_2\}$.

Habiendo resuelto la ambigüedad que surgía al utilizar el algoritmo de división, procedemos ahora a presentar un teorema que nos dará los fundamentos para poder encontrar bases de Gröbner.

Teorema 3.2.1 (Buchberger). *Sea $G = \{g_1, ..., g_t\}$ un conjunto de polinomios no nulos de $k[x_1, ..., x_n]$. Entonces G es una base de Gröbner si y solo si para cada $i \neq j$ se cumple,*

$$S(g_i, g_j) \xrightarrow{G}_+ 0.$$

Corolario 3.2.1. *Sea $G = \{g_1, ..., g_t\}$ un conjunto de polinomios no nulos de $k[x_1, ..., x_n]$. Entonces G es una base de Gröbner si y solo si para cada $i \neq j$ $(1 \leq i, j \leq t)$se cumple,*

$$S(g_i, g_j) = \sum_{v=1}^{t} h_{ijv} g_v, \text{ donde } lp(S(g_i, g_j)) = max_{1 \leq v \leq t}(lp(h_{ijv})lp(g_v)).$$

El teorema de Buchberger nos proporciona las herramientas necesarias para hallar bases de Gröbner. Puesto que los S-polinomios han de ser reducibles por la base de Gröbner a cero, entonces calculamos los S-polinomios del conjunto generador dado y los reducimos, si alguno no es reducible a cero, añadimos el resto de la reducción al conjunto generador y repetimos hasta que todos los S-polinomios sean reducibles a cero. El algoritmo esbozado con palabras es *el algoritmo de Buchberger*.

Algoritmo 4: Algoritmo de Buchberger

Entrada: $F = \{f_1, ..., f_s\} \subseteq k[x_1, ..., x_n]$ con $f_i \neq 0 (1 \leq i \leq s)$
Resultado: Una base de Gröbner $G = \{g_1, ..., g_t\}$ de $\langle f_1, ..., f_s \rangle$
Inicialización: $G := F, \varrho := \{\{f_i, f_j\} | f_i \neq f_j \in G\}$
mientras $\varrho \neq \emptyset$ **hacer**
  Escoger cualquier $\{f, g\} \in \varrho$;
  $\varrho := \varrho - \{\{f, g\}|\}$;
  $S(f, g) \xrightarrow{G}_+ h$;
  si $h \neq 0$ **entonces**
    $\varrho := \varrho \cup \{\{u, h\} \forall u \in G\}$;
    $G := G \cup \{h\}$;
  en otro caso
  fin
fin

Ejemplo 3.2.3. Sean $f_1 = zx + y, f_2 = -x^2zy + y^2 \in \mathbb{R}[x, y, z]$ y $F = \{f_1, f_2\}$ el conjunto generador del ideal I, con el orden gradolexicográfico y con $z > y > x$. Si aplicamos el algoritmo de Buchberger para encontrar una base de Gröbner de I se realizarán los siguientes pasos:
INICIALIZACIÓN: $G := F, \varrho := \{\{zx + y, -x^2zy + y^2\}\}$

Primera iteración $(\varrho \neq \emptyset)$

$\varrho := \emptyset$
$S(xz + y, -x^2yz + y^2) \xrightarrow{G}_+ xy^2 + y^2 = h$
Dado que $h \neq 0$
  $\varrho := \{(xz + y, xy^2 + y^2), (-x^2yz + y^2, xy^2 + y^2)\}$
  $G := \{zx + y, x^2zy + y^2, xy^2 + y^2\}$

Segunda iteración $(\varrho \neq \emptyset)$

$\varrho := \{(-x^2yz + y^2, xy^2 + y^2)\}$
$S(xz + y, xy^2 + y^2) \xrightarrow{G}_+ -y^2z + y^3 = h$
Dado que$h \neq 0$
  $\varrho := \{(xz + y, -y^2z + y^3), (-x^2yz + y^2, -y^2z + y^3), (xy^2 + y^2, -y^2z + y^3)\}$
  $G := \{zx + y, x^2zy + y^2, xy^2 + y^2, -y^2z + y^3\}$

Tercera iteración ($\varrho \neq \emptyset$)

$\varrho := \{(xz + y, -y^2z + y^3), (-x^2yz + y^2, -y^2z + y^3), (xy^2 + y^2, -y^2z + y^3)\}$
$S(-x^2yz + y^2, xy^2 + y^2) \xrightarrow{G}_+ 0 = h$

Cuarta iteración ($\varrho \neq \emptyset$)

$\varrho := \{(-x^2yz + y^2, -y^2z + y^3), (xy^2 + y^2, -y^2z + y^3)\}$
$S(xz + y, -y^2z + y^3) \xrightarrow{G}_+ 0 = h$

Quinta iteración ($\varrho \neq \emptyset$)

$\varrho := \{(xy^2 + y^2, -y^2z + y^3)\}$
$S(-x^2yz + y^2, -y^2z + y^3) \xrightarrow{G}_+ 0 = h$

Sexta iteración ($\varrho \neq \emptyset$)

$\varrho := \emptyset$
$S(xy^2 + y^2, -y^2z + y^3) \xrightarrow{G}_+ 0 = h$

Final del bucle ($\varrho = \emptyset$)

$G = \{xz + y, x^2yz - y^2, xy^2 + y^2, y^2z - y^3\}$
Con lo que hemos obtenido G, que es una base de Gröbner de I por lo que $I = \langle f_1, f_2 \rangle = \langle xz + y, x^2yz - y^2, xy^2 + y^2, y^2z - y^3 \rangle$, y para todo $f \in I$ se tiene que f es reducible a cero por G.

Teorema 3.2.2. *[3] Dado $F = \{f_1, ..., f_s\}$ un conjunto generador de polinomios no nulos de un ideal I de $k[x_1, ..., x_n]$, el algoritmo de Buchberger (algoritmo 4) devuelve una base de Gröbner de I.*

Demostración. Empezaremos demostrando la correcta terminación del algoritmo. Supongamos que el algoritmo no termina, entonces según avanzan las iteraciones del algoritmo se irá teniendo una cadena de de conjuntos tal que

$$G_1 \subsetneq G_2 \subsetneq G_3 \subsetneq \cdots .$$

Y puesto que cada G_i se forma añadiendo un polinomio h a G_{i-1}, estando h reducido respecto a G_{i-1}. Tenemos que $lt(h) \notin Lt(G_{i-1})$, por lo que

$$Lt(G_1) \subsetneq Lt(G_2) \subsetneq Lt(G_3) \subsetneq \cdots .$$

Y esto es una cadena ascendendente que contradice el teorema 1.3.2.
Por lo que tenemos que $F \subseteq G \subseteq I$, de donde $I = \langle F \rangle \subseteq \langle G \rangle \subseteq I$. Por lo tanto G es un conjunto generador de I, es más, si g_i, g_j son dos polinomios diferentes de G, tenemos que $S(g_i, g_j) \xrightarrow{G}_+ 0$. Y por el teorema 3.2.1 G es una base de Gröbner de I. □

3.3. Bases de Gröbner reducidas

Llegado este punto ya sabemos hallar bases de Gröbner. Sin embargo estas bases pueden resultar ser de un tamaño mayor al necesario: véase el ejemplo 3.2.3 en el que obtuvimos la base de Gröbner $G = \{xz + y, x^2yz - y^2, xy^2 + y^2, y^2z - y^3\}$, resulta que $G' = \{xz + y, xy^2 + y^2, y^2z - y^3\}$ es otra base de Gröbner de menor tamaño. En general, el orden en el que se eligen los S-polinomios en el algoritmo de Buchberger y el orden al aplicar el algoritmo de división determina la base de Gröbner que se obtendrá. Por ello será de interés buscar una base de Gröbner de dimensión mínima, a poder ser única. Está sección está destinada a ello.

Lema 3.3.1. *[3] Sea $G = \{g_1, ..., g_t\}$ una base de Gröbner del ideal I. Si $lp(g_2)$ divide a $lp(g_1)$, entonces $\{g_2, ..., g_t\}$ es otra base de Gröbner de I.*

Demostración. Si dado un polinomio f, se tiene que $lp(f)$ es divisible por $lp(g_1)$, entonces también es divisible por $lp(g_2)$. Y por la definición de base de Gröbner tenemos que $\{g_2, ..., g_t\}$ es otra base de Gröbner de I. □

Corolario 3.3.1. *[3] Sea $G = \{g_1, ..., g_t\}$ una base de Gröbner del ideal I. Existe otra base de Gröbner, que llamaremos **minimal**, si eliminamos todos los g_i para los que exista $j \neq i$ tal que $lp(g_j)$ divida a $lp(g_i)$, y dividimos los g_i restantes por $lc(g_i)$.*

De este corolario se sustrae inmediatamente la siguiente definición.

Definición 3.3.1. Una base de Gröbner G se dice que es *minimal* si cumple:

1. $lc(g) = 1$ para todo $g \in G$.
2. Para todo $g \in G$, $lt(g) \notin Lt(G \setminus \{g\})$.

Las bases de Gröbner minimales no son únicas, pero todas las de un mismo ideal tienen la misma cardinalidad y los mismos términos principales.

Proposición 3.3.1. *[3] Si $G = \{g_1, ..., g_t\}$ y $F = \{f_1, ..., f_s\}$ son dos bases de Gröbner minimales del ideal I, entonces $s = t$, y reordenando, $lt(g_i) = lt(f_i)$ $(1 \leq i \leq t)$.*

Demostración. Ver [3, pág. 47] □

Si bien las bases de Gröbner minimales son efectivamente de cardinalidad mínima, no son únicas. Deberemos añadir más condiciones para conseguir esta unicidad. Aún que ahora mismo parezca que la búsqueda de esta base de Gröbner sea arbitraria, se verá que tiene muchas aplicaciones.

Definición 3.3.2. Diremos que una base de Gröbner G es una *base de Gröbner reducida* del ideal $I \subseteq k[x_1, ..., x_n]$ si cumple:

1. $lc(g) = 1$ para todo $g \in G$.
2. Todo $g \in G$ no es reducible por $G \setminus \{g\}$.

El siguiente corolario demostrará que las bases de Gröbner reducidas existen y nos proporcionará un método para hallarlas.

Corolario 3.3.2. *[3] Sea $G = \{g_1, ..., g_t\}$ una base de Gröbner minimal del ideal I. Dell siguiente proceso de reducciones resultará una base de Gröbner reducida de I:*

$$g_1 \xrightarrow{H_1} h_1, \text{ con } h_1 \text{ reducido respecto de } H_1 = \{g_2, ..., g_t\}$$

$$g_2 \xrightarrow{H_2} h_2, \text{ con } h_2 \text{ reducido respecto de } H_2 = \{h_1, g_3, ..., g_t\}$$

$$\vdots$$

$$g_t \xrightarrow{H_t} h_t, \text{ con } h_t \text{ reducido respecto de } H_t = \{h_1, h_2, ..., h_{t-1}\}$$

Y $H = \{h_1, ..., h_t\}$ es la base de Gröbner reducida de I.

Demostración. Dado que G es una base de Gröbner minimal, se tiene que $lp(h_i) = lp(g_i)$ para cada $i = 1, ..., t$. Por lo que H es también una base de Gröbner de I. Puesto que la división de g_i por $h_1, ..., h_{i-1}, g_{i+1}, ..., g_t$ se realiza eliminando términos de g_i mediante $lp(h_1), ..., lp(h_{i-1}), lp(h_{g+1}, ..., lp(h_{g_t})$, y puesto que$lp(h_j) = lp(g_j)$, para todo j, se sigue que H es una base de Gröbner reducida de I. □

La siguiente proposición nos garantiza la unicidad de la bases reducidas de Gröbner.

Proposición 3.3.2 (Buchberger). *Todo ideal no trivial de $k[x_1, ..., x_n]$ tiene una única base de Gröbner reducida para un orden monomial dado.*

Demostración. Ver [4, pág. 92]. □

Ejemplo 3.3.1. Continuaremos el ejemplo 3.2.3, teniamos $f_1 = xz + y, f_2 = x^2yz - y^2, f_3 = xy^2 + y^2, f_4 = y^2z - y^3$ y que $G = \{f_1, f_2, f_3, f_4\}$ es una base de Gröbner. Para calcular una base de Gröbner reducida lo primero que tendremos que hacer es hallar una base minimal de G. Tenemos que $lp(f_1)$ divide a $lp(f_2)$, por lo que eliminamos a f_2 de G. No podemos eliminar más polinomios de G, por lo que $G = \{f_1, f_3, f_4\}$ es una base de Gröbner minimal. Ahora calculamos la base de Gröbner reducida:

$$xz + y \xrightarrow{H_1}_+ h_1 = xz + y, \text{ con } H_1 = \{f_2, f_3\}$$

$$xy^2 + y^2 \xrightarrow{H_2}_+ h_2 = xy^2 + y^2, \text{ con } H_2 = \{h_1, f_3\}$$

$$y^2z - y^3 \xrightarrow{H_3}_+ h_3 = y^2z - y^3, \text{ con } H_3 = \{h_1, h_2\}$$

Por lo que $G' = \{h_1, h_2, h_3\}$ es una base de Gröbner reducida, en esta ocasión coincide con la minimal.

Con esto concluimos este capítulo, ahora ya conocemos como encontrar una base de Gröbner, y en particular una reducida. Esto tendrá aplicación en diversos campos de las matemáticas, alguunas de estas aplicaciones serán presentadas en el siguiente capítulo.

Capítulo 4

Aplicaciones de las bases de Gröbner

Este capítulo está estructurado en cuatro secciones. En la primera sección se darán algunas aplicaciones elementales de las bases de Gröbner, seguidas por el teorema de los ceros de Hilbert (el Nullstellensatz de Hilbert) en la segunda sección, siendo esto la teoría fundamental para la caracterización de la *k-coloreabilidad* de grafos que se presentará en la tercera sección. Por último, en la cuarta sección, se presentarán brevemente algunas otras aplicaciones.

4.1. Aplicaciones elementales

Las bases de Gröbner tienen muchas aplicaciones directas en el álgebra abastracta sin necesidad de introducir conceptos nuevos. En esta sección recogemos algunas consideradas elementales dada su inmediatez.

4.1.1. El problema de pertenencia a un ideal

El problema de pertenencia a un ideal consiste en lo siguiente:

> Dado un polinomio $f \in k[x_1, ..., x_n]$, determinar si f pertenece a un ideal $I = \langle f_1, ..., f_s \rangle \subseteq k[x_1, ..., x_n]$, y si así es, encontrar $h_1, ..., h_s \in k[x_1, ...x_n]$ tales que $f = h_1 f_1 + \cdots + h_s f_s$.

Este problema que otrora pudiera parecer complejo, tiene ahora fácil solución. Dado el ideal I para determinar si f pertenece a I se deberá calcular una base de Gröbner G de I, y por el teorema 3.1.1 se tiene que $f \in I$ si y sólo si

$$f \xrightarrow{G}_+ 0.$$

Y para hallar $h_1, ..., h_s$ basta con aplicar el algoritmo de división multivariable.

Ejemplo 4.1.1. Dado $I = \langle y + x, -xyz + yz^2, y^2x - y \rangle \subset \mathbb{R}[x_1, ..., x_n]$ y $f = 2xy^2 - 2z^2x - 2y - 2xyz$, queremos determinar si $f \in I$. Para ello calculamos una base de Gröbner de I con el orden *deglex* y con $z > y > x$ que resulta ser $G = \{xz, xy^2 - y, yz\}$. Y puesto que $f \xrightarrow{G}_+ 0$ se tiene que $f \in I$.

Nota. De ahora en adelante cuando se realice alguno de los algoritmos introducidos anteriormente no se explicitará la ejecución del mismo.

4.1.2. Determinar si dos ideales son iguales

Dados dos ideales I y J de $k[x_1, ..., x_n]$ queremos determinar si $I = J$. Para ello utilizaremos la proposición 3.3.2, que nos dice que la base reducida de Gröbner de un ideal es única respecto de un orden monomial dado. Por lo tanto I y J son iguales si y sólo si poseen la misma base reducida de Gröbner respecto del mismo orden monomial.

Ejemplo 4.1.2. Sean $I = \langle y^2, z^2 + xyz, xy + y\rangle$ y $J = \langle y^4, x^2 + xy^2, x + y\rangle$. Las bases de Gröbner reducidas de I y J son $G_1 = \{y + xy, xz^2 + z^2, y^2, -z^2 + yz, z^3\}$ y $G_2 = \{x + y, y^2\}$ respectivamente por lo que I no es igual a J.

Otra manera de dar respuesta a esta cuestión consiste en verificar que el conjunto generador de $I = \langle f_1, ..., f_s\rangle$ pertece a $J = \langle f_{1'}, ..., f_{s'}\rangle$ y viceversa. Con lo que se llega a lo siguiente: $I = J \iff \{f_1, ..., f_s\} \subseteq J$ y $\{f_{1'}, ..., f_{s'}\} \subseteq I$.

4.1.3. Encontrar representantes de las clases laterales de $k[x_1, ..., x_n]/I$

Sea $I = \langle f_1, ..., f_s\rangle \subseteq k[x_1, ..., x_n]$ un ideal, y $G = \{g_1, ..., g_t\}$ una base de Gröbner de I. Entonces el teorema 3.1.2 nos dice que para todo polinomio $f \in k[x_1, ..., x_n]$ el resto de dividir f por G es único. Utilizaremos I y G a lo largo de toda la sección.

Definición 4.1.1. Diremos que el resto $r \in k[x_1, ..., x_n]$ resultante de dividir f por G es la *forma normal* de f. Y escribiremos $N_G(f)$.

Proposición 4.1.1. *[3] Sean f, g polinomios de $k[x_1, ..., x_n]$. Entonces*

$$f \equiv g \pmod{I} \text{ si y sólo si } N_G(f) = N_G(g).$$

Demostración. Por el teorema de la división multivariable (2.2.1) existe un polinomio $q \in I$ tal que $f = q + N_G(f)$, por lo que $f - N_G(f) \in I$. Por lo tanto$f + I = N_G(f) + I$ en $k[x_1, ..., x_n]/I$. Por otro lado, para cualesquiera $c_1, c_2 \in k$, y $f_1, f_2 \in k[x_1, ..., x_n]$ se tiene que $c_1f_1 + c_2f_2 - (c_1N_G(f_1) + c_2N_G(f_2)) \in I$ y $c_1N_G(f_1) + c_2N_G(f_2)$ está reducido respecto de G. Entonces $N_G(c_1f_1 + c_2f_2) = c_1N_G(f_1) + c_2N_G(f_2)$, de donde sabemos que la aplicación $N_G : k[x_1, ..., x_n] \to k[x_1, ..., x_n]$ es k-lineal. Bien, ahora $f \equiv g \pmod{I}$ si y sólo si existe $q \in I$ tal que $f = q + g$. Siendo así $N_G(f) = N_G(q) + N_G(g)$, pero $N_G(q) = 0$ dado que $q \in I$, por lo que $N_G(f) = N_G(g)$. A la inversa, si $N_G(f) = N_G(g)$, entonces $f - g = (f - N_G(f)) - (g - N_G(g)) \in I$, de donde $f \equiv g \pmod{I}$. □

Ejemplo 4.1.3. Dado el ideal $I = \langle x + y, 4y^4\rangle \subset k[x_1, ..., x_n]$ y $f = 2xy \in k[x_1, ..., x_n]$, tenemos que una base de Gröbner de I es $G = \{x + y, y^4\}$ respecto del orden lexicográfico con $x > y$. Y $f \xrightarrow{G}_+ -2y^2$. Por lo que $N_G(2xy) = -2y^2$.

4.1.4. Encontrar una base del espacio vectorial $k[x_1, ..., x_n]/I$

En este sección mostraremos como encontrar una base del espacio vectorial $k[x_1, ..., x_n]/I$.

Proposición 4.1.2. *[3] Sea $I = \langle f_1, ..., f_s\rangle \subseteq k[x_1, ..., x_n]$ un ideal, y $G = \{g_1, ..., g_t\}$ una base de Gröbner de I. Entonces una base del espacio vectorial $k[x_1, ..., x_n]/I$. consiste en todas las clases de equivalencia de los productos de potencias de $\mathbb{T}^n$ tales que no sean divisibles por los productos de potencias principales de los polinomios de G.*

Demostración. (Ver [3]). □

Ejemplo 4.1.4. Dado el ideal $I = \langle x + y, x \rangle \subset \mathbb{R}[x_1, ..., x_n]$, tenemos que una base de Gröbner de I es $G = \{x, y\}$ respecto del orden lexicográfico con $x > y$. Por lo que una base del espacio vectorial $\mathbb{R}[x_1, ..., x_n]/I$ consistirá en las clases laterales de 1.

4.2. El Nullstellensatz de Hilbert

En la sección 1.4 vimos que existe una relación entre los subconjuntos de k^n y los de $k[x_1, ..., x_n]$. Y en la proposición 1.4.1 dijimos que si $I = \langle f_1, ..., f_s \rangle$ es un ideal de $k[x_1, ..., x_n]$ entonces la variedad $V(I) \subset k^n$ es el conjunto de soluciones del sistema formado por el conjunto generador de I, por lo que tenemos la siguiente aplicación:

$$\begin{array}{c} \mathit{Ideales} \to \mathit{Variedades} \\ \mathrm{I} \mapsto \mathrm{V(I)} \end{array} .$$

Por otro lado también dijimos que una variedad $V \subset k^n$ puede ser estudiada mediante el ideal

$$I(V) = \{f \in k[x_1, ..., x_n] \quad | \quad f(a_1, ..., a_n) = 0 \text{ para todo } (a_1, ...a_n) \in V\},$$

es decir, el ideal formado por todos los polinomio de $k[x_1, ..., x_n]$ que se anulan en V. Con lo que tenemos otra correspondencia en sentido inverso

$$\begin{array}{c} \mathit{Variedades} \to \mathit{Ideales} \\ \mathrm{V} \mapsto \mathrm{I(V)} \end{array} .$$

Esta sección está dedicada a estudiar esta relación bilateral con mayor detalle. Es relevante notar que, tal y como hemos presentado las correspondencias, existen ciertos problemas.

- Dado el ideal $I = \langle x^2 + 1 \rangle \subset \mathbb{R}[x]$, la variedad $V(I)$ es el conjunto vacio.
- Dos ideales diferentes pueden producir la misma variedad, por ejemplo, para los ideales $I_1 = \langle x, y \rangle$, $I_2 = \langle x^2 + y^2 \rangle \subset \mathbb{R}[x, y]$ se tiene que $V(I_1) = V(I_2) = \{(0, 0)\}$.

Por ello será conveniente perfeccionar esta correspondencia entre ideales y variedades. Para esto primero deberemos introducir algunos conceptos que serán usados a continuación.

Definición 4.2.1. [2, pág. 129] Sean k y K dos cuerpos tales que $k \subset K$ y las operaciones de K restringidas a los elementos de k coinciden con las operaciones de k. Entonces diremos que K es un *cuerpo de extensión* de k, o que k es un *subcuerpo* de K.

Ejemplo 4.2.1. El cuerpo $\mathbb{C}$ es un cuerpo de extensión de $\mathbb{R}$.

Nota. Dada una variedad $V(S)$ con $S \subseteq K[x_1, ..., x_n]$, escribiremos $V_K(S)$ denotando así que $V(S)$ es un subconjunto del cuerpo K^n.

Definición 4.2.2. Diremos que un cuerpo K es *algebraicamente cerrado* si para todo polinomio $f \in K[x_1, ..., x_n]$ con $gr(f) \geq 1$, se tiene que $f = 0$ tiene solución en K.

Definición 4.2.3. El *cierre algebraico* de un cuerpo k es un cuerpo de extensión de k algebraicamente cerrado. Denotaremos al cierre algebraico de k por $\bar{k}$.

Con estas definiciones a mano, si consideramos el cierre algebraico de $\mathbb{R}$, esto es $\mathbb{C}$, ahora tenemos:

- La variedad determinada por el ideal $I = \langle x^2 + 1 \rangle$, es decir $V_{\mathbb{C}}(I)$, en este caso no es el conjunto vacio, siendo $V_{\mathbb{C}}(I) = \{+i, -i\}$.

- Para los ideales $I_1 = \langle x, y\rangle, I_2 = \langle x^2 + y^2\rangle$ con $I_1, I_2 \subset \mathbb{C}$ tenemos por un lado que $V_{\mathbb{C}}(I_1) = \{(0,0)\}$ y por otro que $V_{\mathbb{C}}(I_2) = \{y = \pm ix\} \subset \mathbb{C}$, siendo $V_{\mathbb{C}}(I_1)$ claramente diferente de $V_{\mathbb{C}}(I_2)$.

Teorema 4.2.1 (Nullstellensatz de Hilbert (Versión débil)). *Sea un cuerpo k y su cierre algebraico $\bar{k}$. Sea un ideal $I \subseteq k[x_1, ..., x_n]$, entonces,*

$$V(I)_{\bar{k}} = \emptyset \Leftrightarrow I = k[x_1, ..., x_n].$$

Demostración. Ver [4, p. 177]. □

En este teorema hemos considerado primeramente un cuerpo cualquiera no necesariamente cerrado, para luego relacionar su cierre algebraico con las variedades. Si directamente trabajamos con cuerpos algebraicamente cerrados podemos obtener una relación más fuerte. Para ello necesitaremos la siguiente definición.

Definición 4.2.4. Sea I un ideal del anillo conmutativo R. Diremos que el *radical* de I es el conjunto

$$\sqrt{I} = \{r \in R \text{ tales que } r^k \in I \text{ para algún } k \in \mathbb{N}\}.$$

Ejemplo 4.2.2. A continuación presentamos dos ejemplos de radicales:

- El radical del ideal $I = \langle x^2\rangle$ es $\sqrt{I} = \langle x\rangle$.
- El radical del ideal de $4\mathbb{Z} = \{z \in \mathbb{Z} \text{ tales que } z \text{ es un múltiplo de } 4\} \subset \mathbb{Z}$ es $2\mathbb{Z}$.

Se comprueba fácilmente que efectivamente $\sqrt{I}$ es un ideal. Y además tenemos que $V_K(I) = V_K(\sqrt{I})$.

Teorema 4.2.2 (Nullstellensatz de Hilbert (Versión fuerte)). *Sea k un cuerpo y $\bar{k}$ su cierre algebraico, entonces para todo ideal $I \subsetneqq k[x_1, ..., x_n]$ se tiene*

$$I(V_{\bar{k}}) = \sqrt{I}.$$

Demostración. Ver [4, p. 183]. □

De este teorema se deduce que dos ideales I y J generan la misma variedad si y sólo si $\sqrt{I} = \sqrt{J}$. Enlacemos ahora estos conceptos con las bases de Gröbner. El resultado que se recoge en el siguiente teorema resulta natural del Hilbert Nullstellensatz en su versión débil (4.2.1).

Teorema 4.2.3. *[3] Sea $G = \{g_1, ..., g_t\}$ una base de Gröbner reducida del ideal $I = \langle f_1, ..., f_s\rangle \subseteq k[x_1, ...x]$ respecto de un orden monomial. Entonces*

$$V_{\bar{k}}(I) = \emptyset \text{ si y sólo si } G = \{1\}.$$

Demostración. Por el teorema 4.2.1 se tiene que $V_{\bar{k}}(I) = \emptyset$ si y solo si $1 \in I$, y dado que $G\{1\}$ es una base de Gröbner de I entonces $I = \langle 1\rangle$, y por lo tanto resulta trivial que $1 \in I$. □

Este teorema nos permite decir que dado un conjunto de polinomios que determinan una variedad, esta no tiene solución si 1 pertenece al ideal generado por los polinomios. Esto es claro ya que la variedad determinada por 1, esto es, $1 = 0$ no tiene solución.

Teorema 4.2.4. *[3] Sea $G = \{g_1, ..., g_t\}$ una base de Gröbner reducida del ideal $I = \langle f_1, ..., f_s\rangle \subseteq k[x_1, ...x]$ respecto de un orden monomial. Entonces son equivalentes:*

1. *La variedad $V_{\bar{k}}(I)$ es finita.*
2. *Para todo $i \in \{1, ..., n\} \subset \mathbb{N}$ existe un polinomio $g \in G$ tal que $lp(g) = x_i^v$ para algún $v \in \mathbb{N}$.*
3. *La dimensión del espacio vectorial $k[x_1, ..., x_n]/I$ es finita.*

Demostración. Consultar [3, p. 63]. □

Definición 4.2.5. Diremos que un ideal $I \subsetneq k[x_1, ..., x_n]$ es *cero-dimensional* si cumple alguna, y por lo tanto todas, de las equivalencias del teorema 4.2.4.

Ejemplo 4.2.3. Dado el ideal $I = \langle x + y, x^3 + y, y^2 + x \rangle \Subset \mathbb{R}[x, y]$, se tiene que $G = \{x + y, y^2 - y\}$ es una base de Gröbner de I respecto del orden lexicográfico con $x > y$. Y dado que las variables x e y aparecen en los productos de potencias principales de los polinomios de G, siendo x e y^2 respectivamente, entonces por el teorema 4.2.4 la variedad $V_{\mathbb{C}}(I)$ es finita.

Corolario 4.2.1. *[3, p. 65] Sea $I \subset k[x_1, ..., x_n]$ un ideal cero-dimensional, y sea $G = \{g_1, ..., g_t\}$ su base reducida de Gröbner respecto del orden lex con $x_1 < x_2 < \cdots x_n$. Entonces se pueden ordenar los polinomios de G, de manera que g_1 contenga solo a la variable x_1, g_2 a las variables x_1, x_2 y $lp(g_2)$ es una potencia de x_2, g_3 contenga a las variables x_1, x_2 y x_3 siendo $lp(g_3)$ una potencia de x_3, y así sucesivamente.*

Demostración. Es consecuencia inmediata de la segunda equivalencia del Teorema 4.2.4. Esto es, se pueden reordenar los g_j tales que $lp(g_j)$ es una potencia de x_j. Entonces por el orden lex, las únicas variables que pueden aparecer en g_j son $x_1, x_2, ..., x_j$. □

4.2.1. Resolución de sistemas de ecuaciones no lineales

En la última sección del primer capítulo vimos la semejanza entre la resolución de sistemas de ecuaciones lineales y encontrar el mcd de varios polinomios en $k[x]$. Luego al introducir las bases de Gröbner concluimos que el mcd es un método para encontrar bases de Gröbner en anillos de polinomios de una sola variable. Es consecuente preguntarse si con las bases de Gröbner se pueden resolver sistemas de ecuaciones no lineales, y la respuesta es sí.

Por el Corolario 4.2.1 se tiene que la base de Gröbner reducida de un ideal cero-dimensional tiene una forma *escalonada*, semejante a la forma escalonada reducida por filas de las matrices asociadas a los sistemas lineales. Entonces, si se quiere resolver un sistema de ecuaciones, se considera primeramente el ideal $I \subset k[x_1, ..., x_n]$ generado por los polinomios del sistema, por lo que la solución del sistema serán los puntos de la variedad $V(I)$. Además, por la Proposición 1.4.1 se tiene que se puede calcular $V(I)$ para cualquier conjunto generador de I, es decir, podemos usar la base reducida de Gröbner $G = \{g_1, ..., g_t\}$. Entonces, primeramente se ha de resolver la ecuación de una sola variable $g_1 = 0$. Entonces, para cada solución α de la ecuación se resuelve la ecuación $g_2(\alpha, x_2) = 0$. Continuando así hasta llegar a $g_n = 0$. Las soluciones obtenidas de esta manera son las únicas candidatas posibles. De haberlas, todavía falta por comprobar si las soluciones candidatas cumplen las ecuaciones $g_{n+1} = 0, ..., g_t = 0$, por lo que se comprueba. Y así ya tenemos las soluciones del sistema.

Ejemplo 4.2.4. Queremos resolver el siguiente sistema:

$$\begin{cases} x^2 - 1 = 0 \\ x + y = 0 \\ x + z = 0 \\ z^2 - 1 = 0 \\ y^2 - 1 = 0. \end{cases}$$

Por lo que consideramos el ideal $I = \langle x^2 - 1, x + y, x + z = 0, z^2 - 1, y^2 - 1 \rangle \subset \mathbb{R}[x, y, z]$, y calculamos su base reducida de Gröbner, que es $G = \{x + y, y^2 - 1, z - y\}$. Solucinamos primero la ecuación $y^2 - 1 = 0$, cuya solución se calcula fácilmente, siendo $y = \{1, -1\}$, sustituyendo en $x + y = 0$ se tiene que $x = \{1, -1\}$, y por último haciendo lo mismo en $z - y = 0$ se obtiene $z = \{1, -1\}$. Estas son todas las soluciones del sistema.

4.2.2. Pertenencia al radical de un ideal

Encontrar el radical de un ideal no es, en general, una tarea sencilla. Y dada su complejidad se ha decido no abarcarlo en este trabajo de fin de grado. Sin embargo determinar si un polinomio pertenece al radical de un ideal si es comparativamente sencillo. El siguiente teorema nos propociona un criterio de pertenencia.

Teorema 4.2.5. *[3] Sea el ideal $I = \langle f_1, ..., f_s \rangle \subseteq k[x_1, ..., x_n]$. Entonces $f \in \sqrt{I}$ si y solo si $1 \in \langle f_1, ..., f_s, 1 - wf \rangle \subseteq k[x_1, ..., x_n, w]$, donde w es una variable nueva.*

Demostración. (Consultar [3, p. 66]). □

Por lo que el problema de pertenencia al radical de un ideal tiene fácil respuesta. Dado un ideal $I = \langle f_1, ..., f_s \rangle \subseteq k[x_1, ..., x_n]$ y un polinomio $f \in k[x_1, ..., x_n]$ entonces $f \in \sqrt{(I)}$ si y sólo si considerando el ideal $I' = \langle f_1, ..., f_s, 1 - wf \rangle$, y calculando la base de Gröbner reducida G de I', se tiene que $1 \in G$.

Ejemplo 4.2.5. Dado $I = \langle x + y, x^2, y^2 \rangle$ ideal de $\mathbb{R}[x, y]$ y $f = x^4y^2 + 2x^3y^3 + x^2y^4 \in \mathbb{R}[x, y]$. Si consideramos el ideal $I' = \langle x + y, x^2, y^2, 1 - w(x^4y^2 + 2x^3y^3 + x^2y^4) \rangle$ se tiene que $G = \{1\}$ es una base de Gröbner reducida de I, y por lo tanto $f \in \sqrt{I}$.

4.3. K-coloreabilidad en grafos

Hay un problema muy conocido en la teoría de grafos, llamado *El problema de los tres colores*, que consiste en asignar tres colores distintos a los vértices de un grafo de tal manera que dos vértices adyacentes no tengan el mismo color. La versión con cuatro colores de este problema se planteó en el siglo XIX y hasta 1976 no se conseguió encontrar solución, con ayuda de un ordenador. Para plantear este problema en términos de ideales primero hemos de introducir algunas definiciones de teoría de grafos.

Definición 4.3.1. [10, pág.1] Un *grafo* es un par $G = (V, A)$, donde V un conjunto finito no vacío (a cuyos elementos llamaremos *vértices*) y A es una familia finita de pares no ordenados de vértices de V (a cuyos elementos llamaremos *aristas*).

Definición 4.3.2. [10] Una *coloración (propia)* de un grafo $\mathfrak{G}$ es una asignación de de colores a los vértices de $\mathfrak{G}$, a cada vértice un color, de forma que vértices adyacentes reciban colores

distintos. Si en la coloración (propia) se usan k colores distintos diremos que es una *k-coloración*. Si existe una k-coloración de G diremos que es *k-coloreable*.

Definición 4.3.3. [10] El mínimo K para el que un grafo $\mathfrak{G}$ es k-coloreable se llama número cromático de $\mathfrak{G}$, y se designa por $\chi(\mathfrak{G})$.

El problema de los tres colores consiste por lo tanto en encontrar una 3-coloración de un grafo $\mathfrak{G}$ dado. Resolvámoslo en el ámbito de las bases de Gröbner.

Sea $\mathfrak{G} = (V, A)$ con $|V| = n \in \mathbb{N}$ y sea $\xi = e^{\frac{2\pi i}{3}} \in \mathbb{C}$ una raíz cúbica de la unidad. Entonces representaremos a cada color de la 3-coloración de $\mathfrak{G}$ buscada por $1, \xi, \xi^2$. Ahora debemos asignar a cada vértice uno de l os colores. Si escogemos un conjunto $A = \{x_1, ..., x_n\}$ de variables, una por cada vértice de $\mathfrak{G}$, podemos representar la asignación mediante las siguientes n ecuaciones

$$x_i^3 - 1 = 0, 1 \leq i \leq n. \tag{4.1}$$

Con el sistema formado por las ecuaciones 4.1 hemos asignado a cada vértice un color, pero no hemos impuesto que si dos vértices son adyacentes tengan colores distintos. Dado que cada color es una raíz cúbica de la unidad tenemos que $x_i^3 = x_j^3$, y desarrollando, $(x_i - x_j)(x_i^2 + x_i x_j + x_j^2) = 0$, de donde deducimos que bien $x_i = x_j$ es decir son el mismo color, o $x_i \neq x_j$ y

$$x_i^2 + x_i x_j + x_j^2 = 0, \tag{4.2}$$

ecuación que deben cumplir los vértices adyacentes para que tengan diferente color.

Sea el ideal I de $\mathbb{C}[x_1, ..., x_n]$ generado por los polinomios en 4.1, y los de 4.2 para cada par de vértices conectados por una arista. Entonces si consideramos la variedad $V(I)$ sigue el siguiente teorema.

Teorema 4.3.1. *[3] Un grafo $\mathfrak{G}$ es 3-coloreable si y solo si $V(I) \neq \emptyset$.*

Del teorema se obtiene de inmediato lo siguiente.

Corolario 4.3.1. *Sea G una base reducida de Gröbner y $\mathfrak{G}$ un grafo. Entonces $\mathfrak{G}$ es 3-coloreable si y sólo si $1 \notin G$.*

Demostración. Consecuencia directa de aplicar el teorema 4.2.3 al teorema anterior. □

Ejemplo 4.3.1. Sea el grafo $\mathfrak{G}$ ilustrado en la siguiente figura.

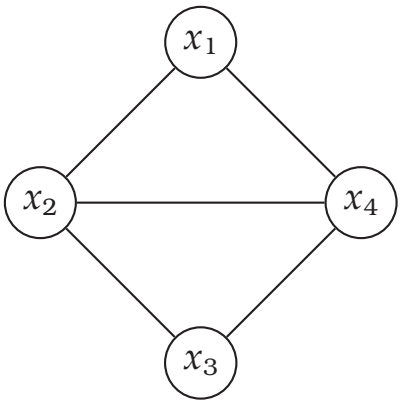


Figura 4.1: Grafo del ejemplo 4.3.1.

El sistema de polinomios que determina la 3-coloreabilidad es

$$\begin{cases} x_4^3 - 1 = 0 \\ x_2^3 - 1 = 0 \\ x_1^3 - 1 = 0 \\ x_3^3 - 1 = 0 \\ x_4^2 + x_4x_2 + x_2^2 = 0 \\ x_4^2 + x_4x_1 + x_1^2 = 0 \\ x_2^2 + x_2x_1 + x_1^2 = 0 \\ x_2^2 + x_2x_3 + x_3^2 = 0 \\ x_4^2 + x_4x_3 + x_3^2 = 0. \end{cases}$$

Si consideramos el ideal I generado por los polinomios del sistema, tenemos que una base reducida de Gröbner de I es $G = \{x_2^2 + x_2x_3 + x_3^2, x_3^3 - 1, x_4 + x_2 + x_3, x_1 - x_3\}$ y además $1 \notin G$ por lo que el grafo es 3-coloreable.

Además, para encontrar una coloración posible simplemente se ha de resolver el sistema determinado por los polinomios de la base de Gröbner. Esto es, $x_3^3 - 1 = 0 \to x_3 = 1$, de donde $x_1 - x_3 = x_1 - 1 = 0 \to x_1 = 1$, y sustituyendo $x_2^2 + x_2x_3 + x_3^2 = x_2^2 + x_2 = 0 \to x_2 = 0 \vee x_2 = -1$, por lo que $x_4 + x_2x_3 = x_4 + x_2 + 1 = 0 \to x_4 = -1$ y $x_2 = 0$ o $x_4 = 0$ y $x_2 = -1$. Es decir, los vértices x_1 y x_3 tienen el mismo color, y los vértices x_2 y x_4 tienen dos colores diferentes entre ellos y diferente al de x_1 y x_3.

Hasta ahora hemos visto como determinar si un grafo es 3-coloreable, pero ¿y en general? De Loera nos dice en [11] que si queremos comprobar si un grafo $\mathfrak{G} = (V, A)$ con $n = |V|$, es k-coloreable tenemos que considerar el ideal

$$I_{\mathfrak{G},k} = I_{n,k} + \langle x_i^{k-1} + x_i^{k-2}x_j + \cdots + x_ix_j^{k-2} + x_ix_j^{k-1} \mid \{i,j\} \in A \rangle,$$

donde

$$I_{n,k} = \langle x^k - 1 \mid i \in V \rangle.$$

Es fácil ver que si $k = 3$ entonces $I_{n,3}$ se corresponde con el ideal generados por los polinomios de la forma vista en 4.1. También es interesante que el ideal $I_{\mathfrak{G},3}$ se corresponde con el ideal $I_{n,3}$ pero añadiendo las restricciones de coloración dadas por los vértices adyacentes, siendo en este caso particular, los polinomios de la forma vista en 4.2.

En general, se pueden deducir los conjuntos generadores de los ideales $I_{\mathfrak{G},k}$ y $I_{n,k}$ de una manera similar a lo visto en 4.1 y 4.2, pero considerando las raices k de la unidad.

Teorema 4.3.2. *[11] El grafo $\mathfrak{G}$ es k-coloreable si y solo si $I_{\mathfrak{G},k}$ tiene alguna raíz en común.*

Demostración. La demostración se puede consultar en [11]. □

Corolario 4.3.2. *El grafo $\mathfrak{G}$ no es k-coloreable si y sólo si la base de Gröbner reducida de $I_{\mathfrak{G},k}$ es $G = \{1\}$.*

Ahora ya tenemos una manera de determinar la k-coloreabilidad de un grafo. Veamos un par de ejemplos.

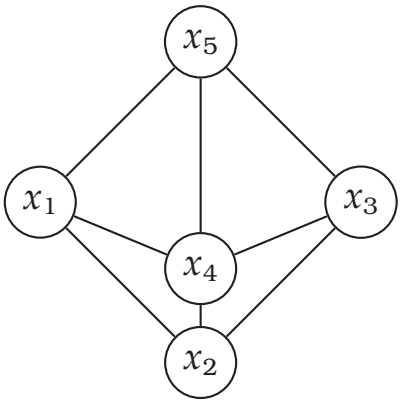


Figura 4.2: Grafo del ejemplo 4.3.2.

Ejemplo 4.3.2. Dado el grafo $\mathfrak{G}$ de la figura 4.3, queremos saber si es 4-coloreable, tenemos por un lado que

$$I_{5,4} = \langle x_5^4 - 1, x_2^4 - 1, x_1^4 - 1, x_4^4 - 1, x_3^4 - 1 \rangle,$$

y por otro que

$$I_{\mathfrak{G},4} = I_{5,4} + \langle x_3^3 + x_3^2 x_5 + x_3 x_5^2 + x_5^3, \cdots, x_4^3 + x_4^2 x_2 + x_4 x_2^2 + x_2^3 \rangle,$$

Y una base reducida de Gröbner de $I_{\mathfrak{G},4}$ es $G = \{x_1^3 + x_1^2 x_2 + x_1 x_2^2 + x_2^3, x_5^4 - 1, x_2^4 - 1, x_1^2 x_5 - x_1^2 x_2 + x_1 x_5^2 - x_1 x_2^2 + x_5^3 - x_2^3, x_3^2 + x_3 x_4 + x_3 x_2 - x_4 x_1 - x_1^2 - x_1 x_2, x_4^2 + x_4 x_1 + x_4 x_2 + x_1^2 + x_1 x_2 + x_2^2, x_3 x_5 - x_3 x_2 - x_1 x_5 + x_1 x_2, x_4 x_5 - x_4 x_2 + x_1 x_5 - x_1 x_2 + x_5^2 - x_2^2\}$ respecto del orden lexicográfico. Puesto que $1 \notin G$ se sigue que el grafo es 4-coloreable.

4.3.1. Una pequeña curiosidad

El juego del Sudoku es un juego de puzzles que es considerado ya un clásico, conquistando el mundo, y en particular los periódicos, al inicio de los años 2000. Lo curioso es que podemos interpretarlo como un un problema de coloración de grafos.

				3	5			
	1		2			9		
7		6				2		
6			5				3	
2				4				9
	3				1			5
		3				4		8
		4			6		7	
			3	1				

Figura 4.3: Ejemplo de un Sudoku.

Sea $\mathfrak{G}$ el grafo que representa al Sudoku, entonces

- $\mathfrak{G}$ tiene 81 vértices, uno por celda.
- Son nesarios 9 colores, uno por número.
- Sus aristas vienen definadas por las reglas de adyacencia del Sudoku, donde es necesario utlizar diferentes números, utilizamos diferentes colores.

Para resolver por ejemplo el sudoku de la figura 4.3 necesitaremos [6]:

- 81 variables x_{ij}, $1 \leq i, j \leq 81$, una por vértice.
- Renombrar las variables de las celdas resaltadas en rojo en la figura 4.4 a $y_1, ..., y_9$.
- El ideal $I_{\mathfrak{G},k}$.
- Los nueve polinomios $y_9^9 - 1, h_8(y_8, y_9), h_7(y_7, y_8, y_9), h_6(y_6, y_7, y_8, y_9), ..., h_1(y_1, ..., y_9) = y_1 + \cdots + y_9$, donde $h_j(y_j, ..., y_k) = y_j^{\alpha_j} \cdots y_k^{\alpha_k}$, con $j = 1, ..., k-1$.
- Los 16 polinomios $x_{31} - y_7, x_{33} - y_6, x_{37} - y_2, ...$

Entonces la base de Gröbner reducida del ideal generado por estos polinomios nos dirá como rellenar los cuadrados.

				3	5			
	1		2			9		
7		6						
6			5				3	
2				4				9
	3				1			5
		3				4		8
		4			6		7	
			3	1				

Figura 4.4: Variables elegidas.

4.4. Otras aplicaciones

Además de las aplicaciones presentadas en el capítulo, existen muchas otras en muy diversos campos. En esta sección se tiene recoger algunas de las más notables o inesperadas de ellas.

1. Procesamiento de señales e imágenes. [12][13]
2. Resolución de ecuaciones diofánticas. [14]
3. Robótica.[15]
4. Pertenencia de un elemento a la imagen de un morfismo.
5. Eliminación.[4] [3]
6. Encontrar el polinomio mínimo de un cuerpo.
7. Encontrar las ecuaciones de una curva en paramétricas.
8. Demostración automática de teoremas.[17]
9. Resolución de problemas de optimización. [16]

Y muchas otras aplicaciones que no han sido mencionadas. Con esto concluimos el capítulo 4, en el siguiente capítulo se explicará la implementación realiza de las bases de Gröbner junto con una aplicación gráfica que hace uso de ellas para resolver el problema de la k-coloreabilidad interactivamente.

Capítulo 5

Implementación

En este capítulo se mostrará y explicará la implementación realizada. Se ha escogido el lenguaje de progrmación C# para el desarrollo por dos razones fundamentales, por un lado la familiaridad del autor con el mismo, y por otro lado, el querer mostrar la no necesidad de lenguajes especializados en matemáticas para poder hacer uso de conceptos matemáticos que aparentemente pudieren ser difíciles de ser implementados en usos reales. Cabe destacar también que pese a que C# es un lenguaje orientado a objetos, tiene una trayectoría de acercamiento a la programación funcional (por muchos desconocida) que nos permite prescindir en ocasiones de cálculos y/o algoritmos tediosos que no forman parte de forma esencial de lo que se está desarrollando.

El trabajo realizado consiste en dos partes principales, la primera es una bibleoteca de clases que implementa todo lo visto en los capítulos 2 y 3, y por ende también parte del capítulo 1. La API (Application Programming Interface) de esta bibleoteca permite el uso las bases de Gröbner en cualquier aplicación que se quiera desarrollar. La segunda parte es una aplicación gráfica que permite determinar la *k-colorabilidad* de un grafo haciendo uso de esta biblioteca.

5.1. Biblioteca

Para poder implementar las bases de Gröbner ha habido primero que trasladar los conceptos subyacentes al lenguaje de programación elegido. Estos conceptos son los vistos en los capítulos 1 y 2, es decir, anillos e ideales de polinomios, cuerpos, orden en anillos de polinomios y los algoritmos preliminares. A continuación se muestra un diagrama de la estructura de clases resultante. En el se pueden apreciar las relaciones entre las diferentes clases y las propiedades y métodos que implmentan.

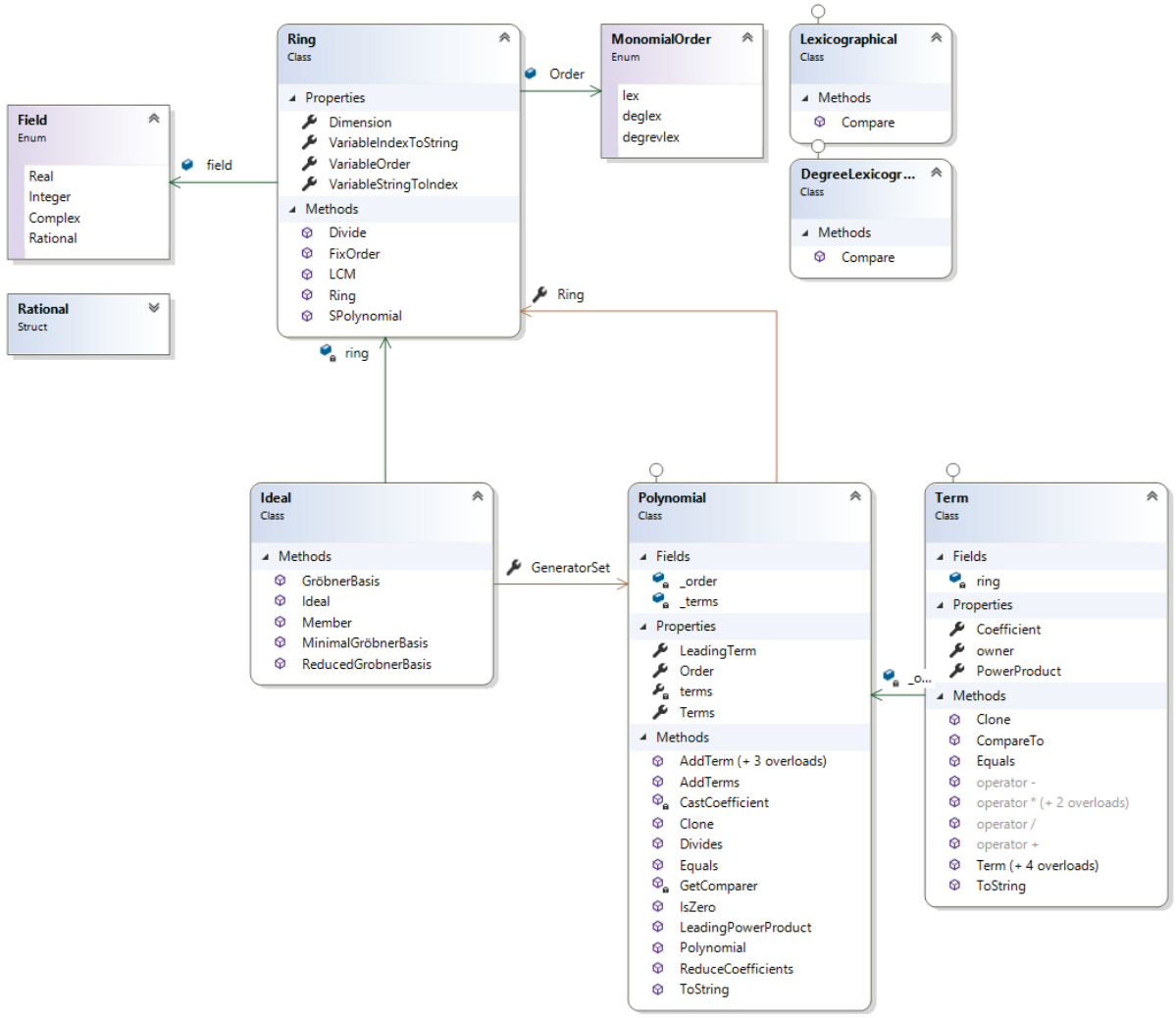


Figura 5.1: Diagrama de clases de la biblioteca.

A continuación explicaremos detalladamente cada clase junto con su implementación.

5.1.1. Ring

La clase *Ring* representa un anillo de polinomios, y como tal se encuentra dentro del paquete *PolynomialRings*.

Nota. De ahora en adelante cuando se hable de anillos, se entenderá que se habla de anillos de polinomios. Y todas las clases que se presenten estarán en el paquete *PolynomialRings*.

5.1.1.1. Campos

- *field*: Los coeficientes de los polinomios de un anillos de polinimios pertenecen al cuerpo que define al anillo. Por ello se ha dotado a la clase de un campo consistente en un cuerpo. Es solo de lectura y se asigna al declarar el anillo.
- *Order*: El orden monomial con el que se ordenan los polinomios del anillo. (Consultar 5.1.4)

5.1.1.2. Propiedades

- *Dimension*: El número de variables que tiene el anillo.
- *VariableOrder*: Orden que se fija entre las variables.
- *VariableIndexToString / VariableStringToIndex*: Diccionarios para traducir entre los nombres de las variables y enteros, ya que la implementación hace uso de la notación introducida en el capítulo 2 para productos de potencias.

5.1.1.3. Métodos

- *Ring*: Constructor de la clase.
- *FixOrder*: Fija un orden dado entre las variables.
- *Divide*: División de dos polinomios pertenencientes al anillo. Devuelve los cocientes y el resto. Se entiende que la división se realiza en el anillo y por ello el método no se encuentra en la clase *Polynomial*. En un futuro puede que sea trasladado.
- *LCM*: Calcula el mínimo común múltiplo de dos polinomios dados del anillo.
- *SPolynomial*: Dado dos polinomios pertenecientes al anillo, devuelve su S-polinimio.

Se puede consultar el código completo en el anexo (Clase Ring).

Ejemplo 5.1.1. En este ejemplo se enseña la declaración de la clase *Ring* junto con el uso de algunos de sus métodos.

```
1     Ring r = new Ring(Field.Real, new string[] { "x", "y", "z" });
2     r.FixOrder(new string[] { "z", "y", "x" });
3     r.Order = MonomialOrder.deglex;
```

Se ha declarado un anillo de polinomios de tres variables x, y, z con coeficientes en $\mathbb{R}$, después se ha fijado el orden $z > y > x$ con el gradolexicográfico.

5.1.2. Term

Un polinomio se compone de términos, y por ello antes de implementar la clase *Polynomial* deberemos tener definidos los términos.

5.1.2.1. Campos

- *ring*: Anillo al que pertenece el término (privado y solo de lectura).
- *_owner*: Polinomio al que pertenece el término (privado).

5.1.2.2. Propiedades

- *Coefficient*: Coeficiente del término, del tipo dynamic, la cantidad de valores que puede tomar está restringida por diseño.
- *PowerProduct*: Producto de potencias del término.
- *Owner*: Propidad que permite el acceso al campo *_owner*.

5.1.2.3. Métodos

Además de algunos métodos necesarios para la implementación se ha realizado la sobrecarga de las operaciones usales para facilitar el uso.

- *Term*: Constructor, admite tres parámetros, el coeficiente, el producto de potencias y el anillo al que pertenece.
- *Clone*: Dado un término devuelve un clon del mismo.
- *CompareTo*: Compara el término con otro, según el orden del anillo, devuelve -1 si es mayor (se encuentra antes en el polinomio), 0 si son iguales y 1 si es menor.
- *Equals*: Devuelve un booleano que será *true* si el término con el que se compara es igual al que ha llamado al método.
- *ToString*: Convierte el polinomio a string, con un formato legible para humanos.
- *operator-*: Permite la resta de términos usando el símbolo $-$.
- *operator**: Permite la suma de términos usando el símbolo $+$.
- *operator-*: Permite la multiplicación de términos usando el símbolo $*$.
- *operator-*: Permite la división de términos usando el símbolo $/$.

Ejemplo 5.1.2. A continuación se encuentra un fragmento de código que muestra la declaración de dos términos con coeficientes complejos y una operación entre ellos.

```
1
2     Ring r = new Ring(Field.Complex, new string[] { "x", "y", "z" });
3
4     Term t1 = new Term(new Complex(1, 2), new int[] { 1, 1, 1 }, r);
5     Term t2 = new Term(new Complex(2, 0), new int[] { 1, 0, 0 }, r);
6     Term res = t1 * t2;
7
8     System.Console.WriteLine("La multiplicación de t1 y t2 da: " + res);
```

La salida de ejecutar el anterior código es:

```
1 La multiplicación de t1 y t2 da: (2, 4)x^2*y*z
```

Para consultar la implementación ver Clase Term.

5.1.3. Polynomial

Esta clase representa los polinomios de un anillo. Como vimos, un polinimio queda definido por los términos que lo componen.

5.1.3.1. Campos

- *_order*: Se utiliza para comprobar si el orden del anillo coincide con el del polinimio, y de no ser asi actualizarlo de ser necesario. (privado)
- *_term*: Colección de términos que determina el polinimio, privado ya que para añadir o eliminar términos se deberán usar los métodos expuestos para ello.

5.1.3.2. Propiedades

- *Ring*: Anillo al que pertenece el polinimio.
- *LeadingTerm*: El término principal del polinimio.
- *Order*: Orden monomial del polinimio, en todo momento coincide con el de su anillo.
- *Terms*: Devuelve una copia de lectura de los términos.

5.1.3.3. Métodos

- *Polynomial*: Constructor de la clase, acepta como argumentos un anillo y opcionalmente una serie de términos.
- *AddTern*: Método público que permite añadir un términos al polinimio.
- *AddTerms*: Método público que permite añadir n términos al polinimio.
- *Clone*: Método público que devuelve un clon del polinimio.
- *Divides*: Método público que devuelve si el polinimio que lo llama divide al polinomio del argumento que se le pasa.
- *Equals*: Método público que determina si dos polinimios son iguales.
- *IsZero*: Método público que determina si el polinimio es nulo.
- *LeadingPowerProduct*: Método público que devuelve el producto de potencias principal.
- *ReduceCoefficients*: Método público que divide todos los coeficientes de cada término del polinimio por el cociente principal.
- *ToString*: Convierte el polinimio a string, en un formato legible por humanos.
- *CastCoefficient*: Método privado, convierte el coeficiente de un cuerpo a otro.
- *GetComparer*: Método privado, devuelve el comparador necesario según el orden del anillo al que pertenece el polinimio.

Ejemplo 5.1.3. Ahora que hemos visto como están implementados los anillos y los polinimios veremos un ejemplo un poco más interesante. Sean $f_1 = xy + y^2, f_2 = x + y, f_3 = xy \in k[x, y]$ con el orden lexicográfico y con $x > y$ queremos comprobar si f_1 divide a f_2, y si f_2 divide a f_3.

```
1  Ring r = new Ring(Field.Real, new string[] { "x", "y" });
2
3
4  Polynomial f1 = new Polynomial(r);
5  f1.AddTerm(1, new int[] { 0, 2 });
6  f1.AddTerm(1, new int[] { 1, 1 });
7
8  Polynomial f2 = new Polynomial(r);
9  f2.AddTerm(1, new int[] { 0, 1 });
10 f2.AddTerm(1, new int[] { 1, 0 });
11
12 Polynomial f3 = new Polynomial(r);
13 f3.AddTerm(1, new int[] { 1, 1 });
14
```

```
15 Console.WriteLine("f1="+ f1);
16 Console.WriteLine("f2="+ f2);
17 Console.WriteLine("f3="+ f3);
18 Console.WriteLine("¿f1 divide a f2?="+ f1.Divides(f2));
19 Console.WriteLine("¿f2 divide a f3?="+ f2.Divides(f3));
```

La salida de ejecutar el anterior código es:

```
1 f1= x*y + y^2
2 f2= x + y
3 f3= x*y
4 ¿f1 divide a f2?=False
5 ¿f2 divide a f3?=True
```

5.1.4. MonomialOrder

MonomialOrder es una enumeración que puede tener los siguientes valores:

1. *lex*: representa el orden lexicográfico.
2. *deglex*: representa el orden gradolexicográfico.
3. *degrevlex*: representa el orden gradoreversolexicográfico.

Se ha decidido utilizar esta estructura en vez de herencia de clases en la que los ejemplos de ordenes monomiales heredasen de una clase abstracta por sencillez de implmentación y limpieza de código. Además es más rápida, lo cuál a efectos prácticos es despreciable ya que no se generaba un cuello de botella aquí.

Los casos concretos de ordenes monomiales implementan la interfaz genérica IComparer<Term>, permitiendo así comparar términos y ordenarlos.

5.1.5. Lexicographical

Dado la brevedad del código mostramos a continuación la implementación del orden visto en 2.1.3.

```
1  public class Lexicographical : IComparer<Term>
2  {
3      public int Compare(Term first, Term second)
4      {
5          if (first.PowerProduct.Length != second.PowerProduct.Length)
6              throw new Exception("One of the power products does not belong
                   to the given Ring!");
7          var ring = first.Owner.Ring;
8          for (int i = 0; i < first.PowerProduct.Length; i++)
9          {
10             if (first.PowerProduct[ring.VariableOrder[i]] >
                   second.PowerProduct[ring.VariableOrder[i]])
11                 return -1;
12             else if (first.PowerProduct[ring.VariableOrder[i]] <
                   second.PowerProduct[ring.VariableOrder[i]])
13                 return 1;
14         }
15         return 0;
```

```
16      }
17  }
```

5.1.6. DegreeLexicographical

Como en el caso anterior la implementación queda concisa y cabe a continuación, siendo una interpretación literal de lo visto en 2.1.4

```
1  public class DegreeLexicographical : IComparer<Term>
2  {
3      public int Compare(Term first, Term second)
4      {
5
6          if (first.PowerProduct.Length != second.PowerProduct.Length)
7              throw new Exception("One of the power products does not belong
                   to the given Ring!");
8
9          int fTotalGrade = first.PowerProduct.Sum();
10         int sTotalGrade = second.PowerProduct.Sum();
11
12         if (fTotalGrade < sTotalGrade)
13             return 1;
14         if (fTotalGrade > sTotalGrade)
15             return -1;
16
17         var lex = new Lexicographical();
18         return lex.Compare(first, second);
19
20     }
21 }
```

5.1.7. Field

Para limitar el número de cuerpos que puede utilizar el usuario, y que no use cualquier clase, se ha concebido el cuerpo como una enumeración que puede tener los siguientes valores:

1. *Reals*
2. *Complex*
3. *Rational*
4. *Integer*

Cada valor de esta enumeración se corresponde con un cuerpo, y dado que .NET no cuenta con una implementación de los racionales se ha creado una clase para ello.

5.1.8. Ideal

Esta clase representa un ideal de un anillo de polinimios y es la que incluirá todo lo relativo a las bases de Gröbner, al ser estas generadores de ideales.

5.1.8.1. Campos

- *ring*: Anillo en el que se encuentra el ideal.

5.1.8.2. Propiedades

- *GeneratorSet*: Conjunto de polinimios generador del ideal.

5.1.9. Métodos

- *Ideal*: Constructor de la clase, acepta como parámetros un anillo y el conjunto generador del ideal.
- *GröbnerBasis*: Método público que devuelve una base de Gröbner calculada a partir del conjunto generador (*GeneratorSet*).
- *MinimalGröbnerBasis*: Método público que devuelve una base de Gröbner minimal encontrada a partir del método anterior.
- *ReducedGröbnerBasis*: Método público que devuelve una base de Gröbner reducida.
- *Member*: Método público que acepta como parámetro un polinimio y determina si este polinimio pertenece al ideal.

Es muy interesante consultar el código implementado en el anexo (Clase Ideal). El algoritmo que se ha programado es el visto en 4, y cabe notar que existen versiones del mismo que haciendo uso de conceptos matemáticos más avanzados consiguen tener una mayor eficiencia. El trabajo realizado en este apartado consiste principalmente en reducir el número de $S-polinimios$ considerados, ya que el algoritmo de división multivariable cuando los polinimios son de grados relativamente grandes tiene un elevado coste computacional. Debido a este alto coste computacional, se ha dotado a todos los algoritmos de la biblioteca la capacidad de ser interrumpidos.

Ejemplo 5.1.4. En este ejemplo se ilustra como la declaración de ideales y la obtención de una base de Gröbner de un ideal.

```
1     Ring r = new Ring(Field.Real, new string[] { "x", "y" });
2
3     Polynomial f1 = new Polynomial(r);
4     f1.AddTerm(1, new int[] { 0, 2 });
5     f1.AddTerm(1, new int[] { 1, 1 });
6
7     Polynomial f2 = new Polynomial(r);
8     f2.AddTerm(1, new int[] { 0, 1 });
9     f2.AddTerm(1, new int[] { 1, 0 });
10
11    Polynomial f3 = new Polynomial(r);
12    f3.AddTerm(1, new int[] { 1, 1 });
13
14    Ideal I = new Ideal(new Polynomial[] { f1, f2, f3 }, r);
15    var gb = I.GröbnerBasis();
16    foreach (var p in gb)
17            Console.WriteLine(p);
```

La salida de ejecutar el anterior código es:

```
1     x*y + y^2
2     x + y
3     x*y
4     y^2
```

Para asegurar la correción del código se han realizado una bateria de cerca de 35 pruebas, intentando en la medida de lo posible replicar casos límites. Ahora que ya conocemos la implementación de la biblioteca y su uso podemos resolver alguno de los ejemplos de los capítulos anteriores computacionalmente.

Ejemplo 5.1.5. El ejemplo 2.2.2 quedaría programado de la siguiente manera:

```
        Ring r = new Ring(Field.Real, new string[] { "x", "y", "z" });

        r.FixOrder(new string[] { "x", "y", "z" });
        r.Order = MonomialOrder.deglex;

        Polynomial f1 = new Polynomial(r);
        f1.AddTerm(1, new int[] { 0, 1, 0 });
        f1.AddTerm(1, new int[] { 0, 0, 0 });

        Polynomial f2 = new Polynomial(r);
        f2.AddTerm(1, new int[] { 1, 0, 0 });
        f2.AddTerm(1, new int[] { 1, 2, 0 });

        Polynomial f3 = new Polynomial(r);
        f3.AddTerm(1, new int[] { 0, 1, 1 });

        Polynomial f4 = new Polynomial(r);
        f4.AddTerm(1, new int[] { 0, 0, 1 });
        f4.AddTerm(1, new int[] { 1, 0, 0 });

        Polynomial dividend = new Polynomial(r);
        dividend.AddTerm(1, new int[] { 3, 3, 1 });
        dividend.AddTerm(1, new int[] { 3, 2, 1 });
        dividend.AddTerm(1, new int[] { 2, 3, 1 });
        dividend.AddTerm(1, new int[] { 2, 2, 1 });

        var quotients = new Polynomial[] { f1, f2, f3, f4 };

        var res = r.Divide(dividend, quotients);
        Console.WriteLine("---------RESULTADO---------");
        Console.WriteLine("resto:" + res.Item2);

        Console.WriteLine("Sin embargo, al alterar el orden de la división
            a  { f2, f3, f1, f4 } se tiene que:");
        res = r.Divide(dividend, new Polynomial[] { f2, f3, f1, f4 });
        Console.WriteLine("resto:" + res.Item2);
```

Y al ejecutar el anterior código obtenemos:

```
        ---------RESULTADO---------
        resto: 0
        Sin embargo, al alterar el orden de la división a  { f2, f3, f1,
            f4 } se tiene que:
        resto: z^4 - z^3
```

Que en efecto coincide con el resultado del ejemplo 2.2.2.

Ejemplo 5.1.6. Dado el ideal del Ejemplo 3.2.3, calculamos la base de Gröbner reducida directamente de la siguiente manera:

```
        //R[x,y,z]
        Ring r = new Ring(Field.Real, new string[] { "x", "y", "z" });
        //Con el orden deglex
        r.Order = MonomialOrder.deglex;

        //Fijamos el orden z>y>x
        r.FixOrder(new string[] { "z", "y", "x" });

        //Polinomio f1
        Polynomial f1 = new Polynomial(r);
        f1.AddTerm(1, new int[] { 1, 0, 1 });
        f1.AddTerm(1, new int[] { 0, 1, 0 });
        //Polinomio f2
        Polynomial f2 = new Polynomial(r);
        f2.AddTerm(-1, new int[] { 2, 1, 1 });
        f2.AddTerm(1, new int[] { 0, 2, 0 });

        //Ideal generado por ambos polinomios
        Ideal I = new Ideal(new Polynomial[] { f1, f2 }, r);

        //Base de Gröbner asociada
        var gb = I.ReducedGrobnerBasis();

        //Imprimimos el resultado por consola
        Console.WriteLine("---------RESULTADO---------");
        foreach (var p in gb)
            Console.WriteLine(p);
```

Y al ejecutar el anterior código obtenemos:

```
        ---------RESULTADO---------
        x*z + y
        x*y^2 + y^2
        y^2*z - y^3
```

Que coincide con lo visto en el Ejemplo 3.3.1.

5.2. Aplicación gráfica de la k-colorabilidad.

La aplicación gráfica desarrollada para determinar la k-colorabilidad utiliza la biblioteca anterior, y una interesante biblioteca de grafos, llamada *MSAGL*, desarrollada por Microsoft, la cuál es de código abierto. Esta última biblioteca nos permite tener una manera de crear, guardar y editar grafos visualmente sin mayores complicaciones. Como contrapartida la documentación es escasa y en ocasiones se encuentra desfasada. Usando MSAGL es necesario realizar una conversión de la estructura que aporta de grafos a los anillos de polinimios, tal y como se describió en el capítulo 4. Además de ello ha sido necesario programar toda la interfaz gráfica para el problema de la k-colorabilidad. En la siguiente figura se muestra una visión global de la aplicación, después detallaremos su funcionamiento.

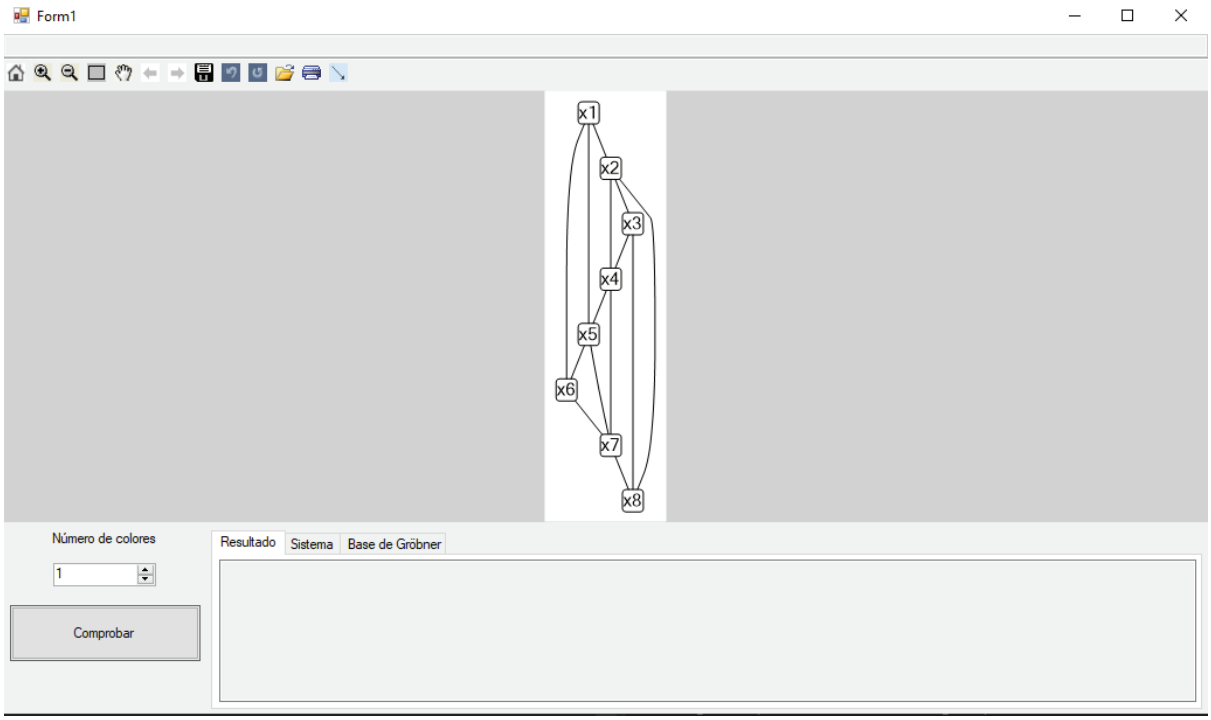


Figura 5.2: Aplicación gráfica.

5.2.1. Edición de grafos.

La zona de edición de grafos, se corresponde de dos partes, la primera es la barra de herramientas que permite hacer diferentes acciones, y la segunda la visualización gráfica del grado, que también es dinámica y permite otras accciones. Cada zona se corresponde con las secciones señaladas en la figura 5.3 como 1 y 2 respectivamente.

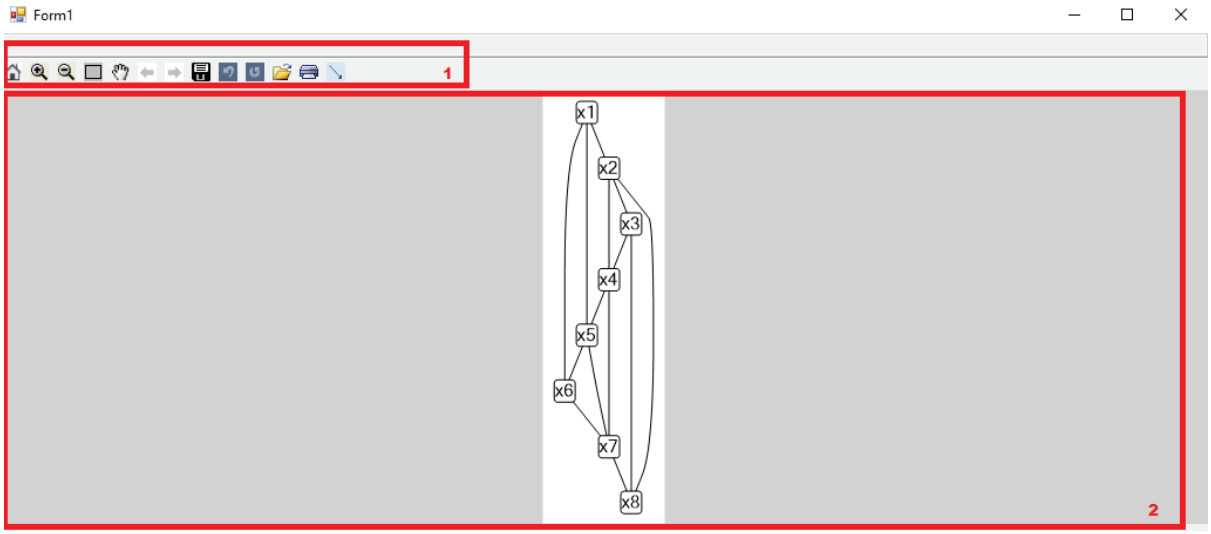


Figura 5.3: Partes de la la zona de edición de grafos.

5.2.1.1. Barra de herramientas.

Figura 5.4: Aplicación gráfica.

En la figura 5.4 se enumera cada posible acción, que se corrresponden con:

1. Centrar la vista: Centra la vista en el grafo.
2. Ampliar zoom en el grafo.
3. Alejar zoom del grafo.
4. Centra la vista en el recuadro dibujado.
5. Herramienta de mano: Permite moverse libremente por el grafo.
6. Deshacer cambio en la posición visual.
7. Rehacer cambio la posición visual.
8. Guardar grafo en el directorio elegido.
9. Deshacer cambio en la estructura del grafo.
10. Rehacer cambio en la estructura del grafo.
11. Abrir grafo guardado en un directorio.
12. Imprimir grafo con las opciones disponibles en el sistema operativo.
13. Modo inserción de aristas: permite insertar aristas entre dos nodos/vértices arrastrando el cursor.

5.2.1.2. Visualización gráfica del grafo

Este panel del grafo, tal y como su nombre indica nos permite tener una representación visual del mismo. El usuario puede desplazarse por la imagen y realizar zoom con la rueda del ratón. Además al realizar click derecho sobre cualquier parte se muestra el siguiente desplegable:

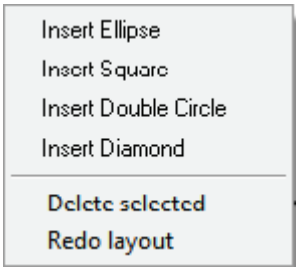


Figura 5.5: Desplegable.

Gracias a esto se pueden insertar y eliminar vértices. Para añadir aristas, primero hay que activar el modo de inserción de aristas en la barra de herramientas, una vez activado basta con arrastrar de un vértice a otro manteniendo el botón izquierdo del ratón pulsado.

5.2.2. Determinación de la k-colorabilidad.

Para determinar la k-colorabilidad del grafo que se muestra se dispone del siguiente panel:

Figura 5.6: Panel k-colorabilidad.

Se dispone de un cuadro de texto que solamente admite enteros positivos, cambiando este valor se selecciona la *k* de la k-colorabilidad a determinar. Y al pulsar con el botón izquierdo sobre *comprobar* comienza la computación de la base de Gröbner del sistema asociado al grafo. Para mostrar el estado actual del cálculo al usuario se sustituye el botón de comprobar por una barra de progreso, tal y como se ve en la figura **??**, además el usuario puede cancelar la operación en todo momento.

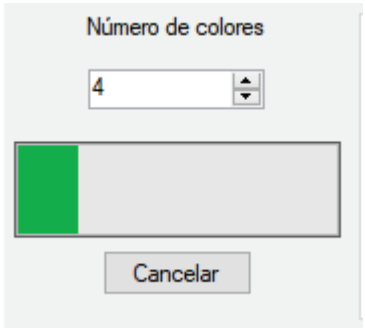


Figura 5.7: Barra de progreso.

Una vez determinada la k-colorabilidad dada para el grafo cargado, se arroja al usuario la siguiente información, dividida en diferentes pestañas:

- Resultado de la determinación de la k-colorabilidad.
- El sistema que determina la k-colorabilidad del grafo.
- La base de Gröbner reducida del anterior sistema.

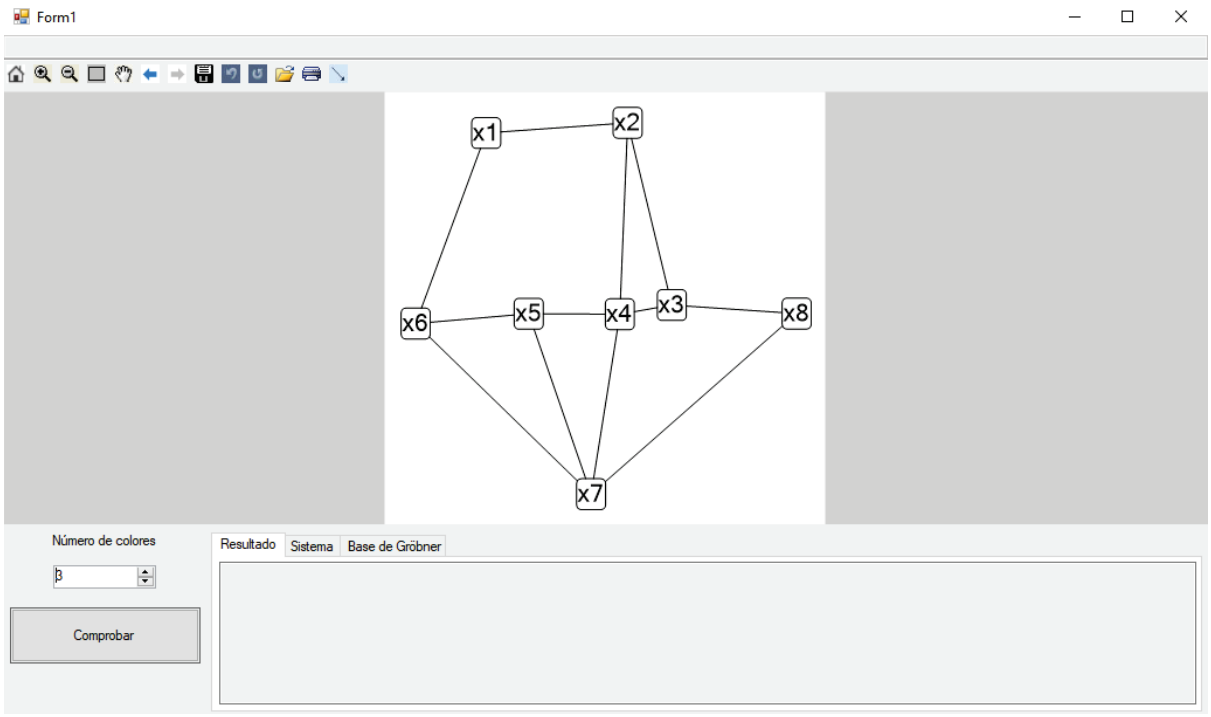


Figura 5.8: Ejemplo de funcionalidad.

Ejemplo 5.2.1. Dado el programa en el estado que se muestra en la figura 5.8, si se comprueba la k-colorabilidad se obtiene la siguiente salida:

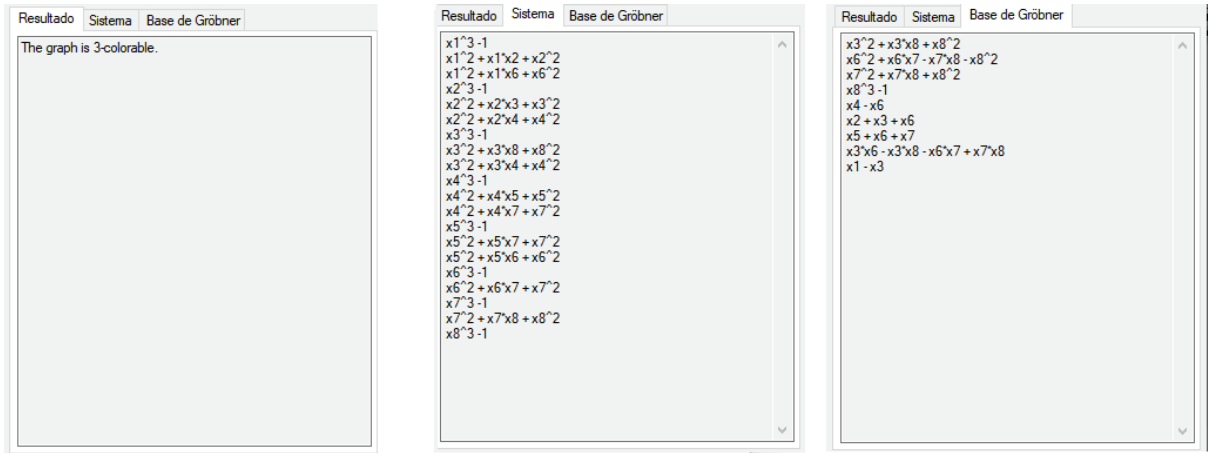


Figura 5.9: Salida del ejemplo 5.8.

El código fuente de todo lo explicado en este capítulo queda recogido en el anexo Clase de la

interfaz de usuario. Además todo el código se encuentra en el repositorio de Github `https://github.com/JuanGdelaCruz/GrobnerBasis`. Junto con númerosos ejemplos para realizar diferentes pruebas. Algunos de estos ejemplos se pueden consultar en 6.

Capítulo 6

Conclusiones

Las bases de Gröbner son una herramienta muy potente que permiten calcular mediante algoritmos, y por lo tanto usando un ordenador, una serie de problemas elementales del álgebra abstracta, problemas con repercursión en muchas otras áreas de las matemáticas. En ocasiones sorprendentes. Pero este potencial se ve en ocasiones oscurecido por tu alto coste computacional.

Este alto coste computacional se debe principalmente a dos cuellos de botella que surgen al usar los algoritmos que permiten hallarlas, siendo el primero el algoritmo de divisón multivariable (Algoritmo 3). En este el orden con el que se eligen los polinomios al dividir determina de forma notable el tiempo total de ejecución. Veamos un ejemplo.

Ejemplo 6.0.1. En el anillo de polinomios $k[x, y]$ sean los polinomios $f_1 = x^{1000}$, $f_2 = x + y$, $f_3 = x$, y el conjunto $F = \{f_2, f_3\}$. Cuando se realiza la división de f_1 por F con el orden lex y fijando $x > y$, se tiene que ambos f_2 y f_3 dividen a f_1, y por lo tanto el orden en el que se escogan determinará el tiempo de ejecución, esto se muestra en la tabla 6.1 donde se ve que se obtienen resultados muy dispares. Si se utiliza f_2 antes que f_3 el tiempo de ejecución es $18,9$ veces mayor que cuando se usa f_3 antes que f_2. Un humano sin embargo hubiera podido ver que f_3 simplifica f_1 de forma casi inmediata.

$F =$	$\{f_2, f_3\}$	$\{f_3, f_2\}$
Tiempo (ms)	945	50

Cuadro 6.1: Tiempo de ejecución del algoritmo 3.

Pero el verdadero cuello de botella viene dado por el algoritmo de Bucherberger (Algoritmo 4), en este se calculan todas las combinaciones de S-polinomios posibles, sin realizar ningún tipo de criba, esto implica que se ha de realizar el algoritmo de división para todos ellos, obteniendo en la mayoría de los casos 0 como resto, y no añadiéndolo por tanto al conjunto generador. Y ya que el algoritmo de división multivariable requiere gran potencia de cálculo, se traduce en un mayor tiempo de ejecución, empleado en gran parte en reducir estos polinomios que son prescindibles.

Se ha realizado mucho trabajo para reducir este coste computacional, por un lado no considerando todos los S-polinomios, y por otro intentando evitar usar el costoso algoritmo de división en la medida de lo posible. La versión que suelen implementar la mayoría de los paquetes matemáticos son los algoritmos *F4* y *F5* de Faugère. Estos usan los mismos principios

matemáticos que el algoritmo de Bucherberger pero haciendo uso de matrices de dispersión, y calculando bases de Gröbner incrementalmente, reduciendo así de forma considerable la complejidad del algoritmo, para mas información referise a [4].
Veamos con un ejemplo cuán alto es el coste computacional del algoritmo de Bucherberger.

Ejemplo 6.0.2. Revisitamos el ejemplo 4.3.1. Pero en esta ocasión lo que haremos es determinar la *k*-coloreabilidad para todos los *k* tales que la computación de la k-coloreabilidad tarda menos de 10 minutos. Y recogeremos el tiempo transcurrido en una tabla. Recordamos que determinar la k-coloreabilidad consiste en calcular una base de Gröbner para el ideal asociado al sistema que define al grafo. Se han realizado 4 ejecuciones por cada *k* para intentar minimizar cualquier error cometido por diferencias en el estado del ordenador.

k / Ejecución	1	2	3	5	6	6	7	8	9	10
Primera (ms)	4	8	24	86	3381	8674	16132	83160	104695	428410
Segunda (ms)	3	7	26	106	3379	8694	16229	83841	104939	434017
Tercera (ms)	2	10	25	87	3418	8780	16218	83260	105632	427438
Cuarta (ms)	3	8	28	88	3434	8690	16182	84614	105302	426425
Media (ms)	3	8,3	25,9	90	3417	8709,3	16190,2	83714,8	105140,8	429052,5

Cuadro 6.2: Tiempos de ejecución.

Como se puede observar en la tabla 6.2, el tiempo aumenta considerablemente.

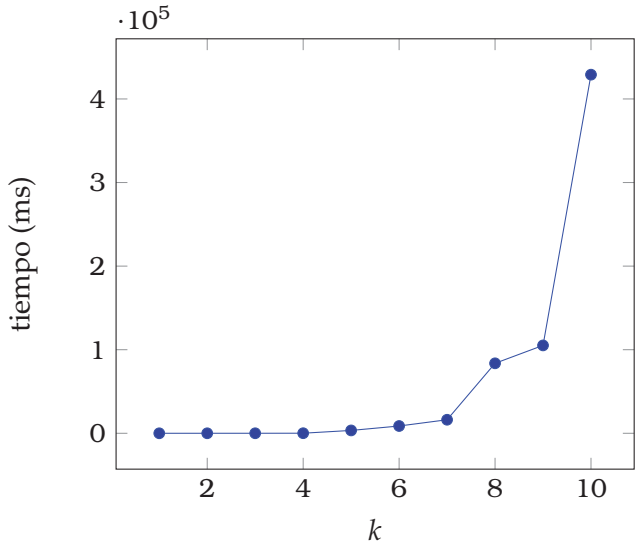


Figura 6.1: Gráfica de la tabla 6.2.

En la figura 6.1 se puede apreciar aun más fácilmente el incremento en el tiempo de ejecución según aumenta la complejidad del sistema. En concreto, al aumentar el número de colores elegidos, aumenta tanto el grado de los polinomios del sistema, como el número de términos en los polinomios que dan las restricciones.

Se han realizado diferentes estudios sobre la complejidad del algoritmo de Bucherberger, pero no es fácil determinarla, dada la gran cantidad de decisiones que se pueden tomar dentro

del él. Por ejemplo, la simple ordenación inicial del conjunto generador del ideal determina el número de S-polinomios que se consideran.

Ejemplo 6.0.3. Consideramos de nuevo el ideal asociado al grafo del ejemplo anterior. Además se ha realizado una modificación en el código de la biblioteca, reordenando aleatoriamente los polinomios del generador del ideal antes de calcular la base de Gröbner reducida. Entonces al determinar 4-coloreabilidad del grafo utilizando nuestra aplicación con la modificación del código mencionada, y tras realizar doce ejecuciones, se obtienen los resultados de la tabla 6.3. Como vemos, la cantidad de S-polinomios considerados oscila ostensiblemente, siendo la cantidad más baja considerada 117, y la más alta 399 ¡Esto son 282 S-polinomios tenidos en cuenta innecesariamente! Y la diferencia de tiempo de ejecución entre ambos casos es de $357ms$, cerca de 5 veces más lento.

Nº de S-polinomios considerados	135	117	174	240	399	289	135	154	135	342	195	240
Tiempo (ms)	93	87	122	208	444	214	115	116	96	284	161	194

Cuadro 6.3: Tiempo y S-polinomios considerados.

La figura 6.2 es un diagrama de dispersión con los datos de la tabla 6.3, en ella se puede percibir como un menor número de S-polinomios considerados implica un menor tiempo de ejecución.

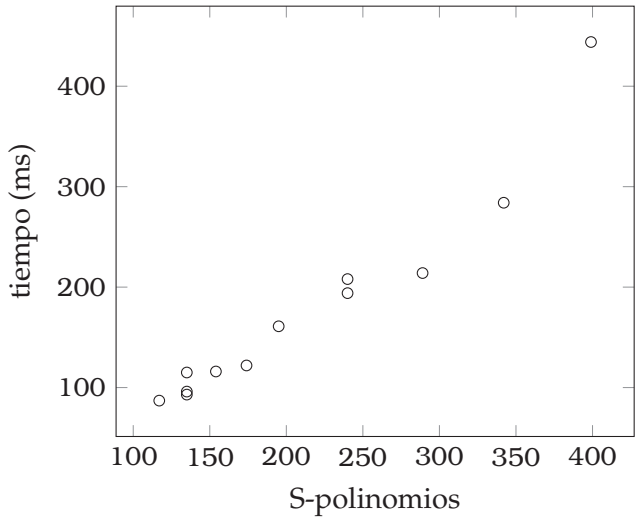


Figura 6.2: Diagrama de dispersión de la tabla 6.3.

Una pequeña reflexión del autor

En mi opinión este trabajo es un fiel reflejo de lo que pretende ser la carrera de matemáticas e informática, ya que por un lado, la curiosidad que me suscitó la asignatura de estructuras algebraicas me llevó a elegirlo, adquiriendo gracias a dicha elección una ingente cantidad de conocimiento nuevo, y por otro lado, mi educación en informática me ha permitido implementar toda la teoría vista de una manera limpia y eficiente en un lenguaje de programación que usualmente no es usado con este objetivo.

Sin duda alguna ha sido un camino con dificultades, en gran medida a mi previo escaso conocimiento de en cuanto al álgebra abstracta, pero a mi parecer con un gran aprendizaje realizado. Por ello quiero agradecer la rigurosidad empleada por mi tutor, propia de todo buen matemático, en la correción y sugerencias que me ha realizado a lo largo de la elaboración de este documento. Que sin duda alguna me han servido para escribir mejor las matemáticas.

Bibliografía

[1] T.W. Judson, *Abstract Algebra: theory and applications*, Stephen F. Austin State University, 2019.

[2] A. Mata, *Estructuras Algebraicas. Guía de Clase*, Fundación General de la U.P.M., Madrid, 2019.

[3] W.W. Adams and P. Loustaunau, *An introduction to Gröbner Bases*, American Mathematical Society, 1994.

[4] Cox, D., Little, J. and O'Shea, D. (2007). *Ideals, Varieties, and Algorithms. An Introduction to Computational Algebraic Geometry and Commutative Algebra.*.

[5] Hillar, C. J.and Windfeldt, T. (2008). *Algebraic characterization of uniquely vertex colorable graphs.* J. Comb. Theory, Ser. B, 98, 400-414.

[6] D.A. Cox, *Gröbner Bases Tutorial. Part II: A Sampler of Recent Developments*, ISSAC 2007 Tutorial, Amherst College, 2007.

[7] Chao, C.-Y.and Chen, Z. (1993). *On uniquely 3-colorable graphs.* Discret. Math., 112, 21-27.

[8] Atiyah, M. F., MacDonald, I. G. (1969).*Introduction to commutative algebra.* Addison-Wesley-Longman. ISBN: 978-0-201-40751-8

[9] Loera, J. A. D., Margulies, S., Pernpeintner, M., Riedl, E., Rolnick, D., Spencer, G., . . . Swenson, J. (2015). *Graph-Coloring Ideals: Nullstellensatz Certificates, Gröbner Bases for Chordal Graphs, and Hardness of Gröbner Bases*. In Proceedings of the 2015 ACM on International Symposium on Symbolic and Algebraic Computation (pp. 133–140).

[10] G. Hernández.(2014). *Grafos, Teoría y Algoritmos, 2ª edición.* Fundación General de la U.P.M., Madrid.

[11] Loera, Jesús A. De, Margulies, Susan, Pernpeintner, Michael, Riedl, Eric, Rolnick, David, Spencer, Gwen, Stasi, Despina and Swenson, Jon.(2018) *Gröbner Bases and Nullstellensätze for Graph-Coloring Ideals*.

[12] Bose, N. K. (2001). *GSpecial Issue on Applications of Gröbner Bases to Multidimensional Systems and Signal Processing.* Multidim. Syst. Sign. Process., 12, 215-216.

[13] Lin, Zhiping , Xu, Li and Wu, Qinghe. (2004). *Applications of Gröbner bases to signal and image processing: A survey. Linear Algebra and its Applications.* 391. 169-202. 10.1016/j.laa.2004.01.008.

[14] Cipu, Mihai. (2008). *Grobner Bases and Solutions to Diophantine Equations.* 77 - 80. 10.1109/SYNASC.2008.13.

[15] Abłamowicz, R. (2010). *Some Applications of Gröbner Bases in Robotics and Engineering*. *Geometric Algebra Computing*. ISBN 978-1-84996-107-3. Springer-Verlag London Limited, 2010, p. 495.

[16] Ziegler G.M. (1999) Gröbner Bases and Integer Programming. In: Cohen A.M., Cuypers H., Sterk H. (eds) Some Tapas of Computer Algebra. Algorithms and Computation in Mathematics, vol 4. Springer, Berlin, Heidelberg

[17] Buchberger, Bruno. (2005). *From Gröbner Bases to Automated Theorem Proving and Back.*

[18] Renschuch, Bodo & Roloff, Hartmut & Rasputin, Georgij & Abramson, Michael. (2003). *Contributions to constructive polynomial ideal theory XXIII: forgotten works of Leningrad mathematician N. M. Gjunter on polynomial ideal theory.* ACM SIGSAM Bulletin. 37.

[19] Abramson, Michael. (2009). *Historical background to Gröbner's paper*. ACM Sigsam Bulletin.

Anexo

En este capítulo se recoge el código fuente de la implementación realizada. La versión más reciente del mismo se puede encontrar disponible online en el siguiente enlace: `https://github.com/JuanGdelaCruz/GrobnerBasis`. Junto con números ejemplos para realizar diferentes pruebas.

Código

Biblioteca

Ring

Listing 1: Clase Ring

```
public class Ring
{

    public Dictionary<int, string> VariableIndexToString { get; private set; } = new Dictionary<int, string>();
    public Dictionary<string, int> VariableStringToIndex { get; private set; } = new Dictionary<string, int>();

    public Dictionary<int, int> VariableOrder { get; private set; } = new Dictionary<int, int>();

    public readonly Field Field;

    public MonomialOrder Order = MonomialOrder.lex;

    public int Dimension { get; private set; }

    public Ring(Field field, string[] vars)
    {
        this.Field = field;
        Dimension = vars.Length;
        for (int i = 0; i < vars.Length; i++)
        {
            VariableOrder.Add(i, i);
            VariableIndexToString.Add(i, vars[i]);
            VariableStringToIndex.Add(vars[i], i);
        }
    }

```

```
27      public Tuple<Polynomial[], Polynomial> Divide(Polynomial dividend,
            Polynomial[] divisors, CancellationToken token = default)
28      {
29          //Initialization
30          Polynomial remainder = new Polynomial(this);
31          Polynomial[] quotients = new Polynomial[divisors.Length];
32          for (int i = 0; i < quotients.Length; i++)
33              quotients[i] = new Polynomial(this);
34          //Begin algorithm
35          Polynomial h = dividend.Clone();
36
37          while (!h.IsZero())
38          {
39              if (token != CancellationToken.None &&
                    token.IsCancellationRequested)
40                  throw new OperationCanceledException();
41
42              var first = divisors.FirstOrDefault(f_i => f_i.Divides(h));
43
44
45              if (first != null)
46              {
47
48                  //CHECK if bottleneck: Using a dictionary for the
                        quotients might improve performance.
49                  int index = Array.IndexOf(divisors, first);
50                  var partialQuotient = h.LeadingTerm /
                        divisors[index].LeadingTerm;
51                  quotients[index].AddTerm(partialQuotient);
52
53                  foreach (Term term in divisors[index].Terms)
54                  {
55                      var subtract = -1 * partialQuotient * term;
56                      h.AddTerm(subtract);
57                  }
58
59              }
60              else
61              {
62                  var clone = h.LeadingTerm.Clone(remainder);
63                  remainder.AddTerm(clone);
64                  var subtract = h.LeadingTerm * -1;
65                  h.AddTerm(subtract);
66
67
68              }
69          }
70          return new Tuple<Polynomial[], Polynomial>(quotients, remainder);
71      }
72
73      public bool FixOrder(string[] order)
74      {
75          if (order == null || order.Length != Dimension)
76              return false;
77          for (int i = 0; i < order.Length; i++)
78          {
```

```
79              VariableOrder[i] = VariableStringToIndex[order[i]];
80          }
81          return true;
82      }
83
84      public Polynomial SPolynomial(Polynomial f, Polynomial g)
85      {
86
87          //S(f,g)= L/lt(f) * f - L/lt(g) *g   with L = lcm(lp(f),lp(g)).
88          Polynomial s = new Polynomial(this);
89          Term lcm = LCM(f.LeadingTerm, g.LeadingTerm);
90          // L / lt(f)
91          {
92
93              Term div = lcm / f.LeadingTerm;
94              // L / lt(f) *f
95              foreach (Term term in f.Terms)
96              {
97                  var add = div * term;
98                  s.AddTerm(add);
99              }
100         }
101         //-L / lt(g) * g
102         {
103             // L / lt(g)
104             Term div = lcm / g.LeadingTerm;
105             // - L / lt(g) * g
106             foreach (Term term in g.Terms)
107             {
108                 var sub = div * (-1 * term);
109
110                 s.AddTerm(sub);
111             }
112         }
113
114         return s;
115     }
116
117
118     public Term LCM(Term first, Term second)
119     {
120         if (first.PowerProduct.Length != Dimension ||
                second.PowerProduct.Length != Dimension)
121             return null;
122
123         int[] powerProduct = new int[Dimension];
124         for (int i = 0; i < Dimension; i++)
125         {
126             powerProduct[i] = Math.Max(first.PowerProduct[i],
                    second.PowerProduct[i]);
127         }
128         return new Term(1, powerProduct, this);
129     }
130 }
```

Term

Listing 2: Clase Term

```
1
2  namespace GröbnerBasis.PolynomialRings
3  {
4      public class Term : IEquatable<Term>, IComparable<Term>
5      {
6          public int[] PowerProduct { get; private set; }
7          private Polynomial _owner = null;
8          private readonly Ring ring;
9
10         public dynamic Coefficient { get; set; }
11
12         public Polynomial Owner
13         {
14             get => _owner;
15             set
16             {
17                 if (_owner == null)
18                     _owner = value;
19             }
20         }
21
22
23         public Term(double real, int[] power, Ring ring) : this(power,
               ring) => Coefficient = real;
24         public Term(long integer, int[] power, Ring ring) : this(power,
               ring) => Coefficient = integer;
25         public Term(Complex complex, int[] power, Ring ring) : this(power,
               ring) => Coefficient = complex;
26         public Term(Rational rational, int[] power, Ring ring) :
               this(power, ring) => Coefficient = rational;
27         private Term(int[] power, Ring ring) => (PowerProduct, this.ring)
               = (power, ring);
28
29
30
31         public bool Equals(Term other)
32         {
33             return PowerProduct.SequenceEqual(other.PowerProduct);
34         }
35
36         public override string ToString()
37         {
38             StringBuilder power = new StringBuilder(PowerProduct.Length *
                   4);
39             for (int i = 0; i < PowerProduct.Length; i++)
40             {
41                 if (PowerProduct[i] > 0)
42                 {
43                     power.Append(ring.VariableIndexToString[i]);
44                     if (PowerProduct[i] > 1)
45                         power.Append("^" + PowerProduct[i]);
46                     power.Append('*');
```

```
47                  }
48              }
49
50              if (power.Length > 0 && power[power.Length - 1] == '*')
51                  power.Remove(power.Length - 1, 1);
52
53              string coeff = "";
54              var c = (Complex)Coefficient;
55              if (power.Length > 0 && c.Imaginary == 0 && Math.Abs(c.Real)
                    == 1)
56              {
57                  coeff += (c.Real < 0 ? "- " : " ");
58              }
59              else
60                  coeff = Coefficient.ToString();
61
62              return coeff + power;
63          }
64
65
66
67          public int CompareTo(Term other)
68          {
69              IComparer<Term> comparer;
70              switch (Owner.Order)
71              {
72                  case MonomialOrder.deglex:
73                      comparer = new DegreeLexicographical();
74                      break;
75                  default:
76                      comparer = new Lexicographical();
77                      break;
78              }
79              return comparer.Compare(this, other);
80
81          }
82
83          public static Term operator +(Term a, Term b)
84          {
85              if (!a.Equals(b) || a.ring != b.ring)
86                  throw new ArithmeticException();
87              return new Term(a.Coefficient + b.Coefficient, a.PowerProduct,
                    a.ring);
88
89          }
90
91          public static Term operator -(Term a, Term b)
92          {
93              if (!a.Equals(b) || a.ring != b.ring)
94                  throw new ArithmeticException();
95              return new Term(a.Coefficient - b.Coefficient, a.PowerProduct,
                    a.ring);
96
97          }
98          public static Term operator *(Term p1, Term p2)
99          {
```

```
100             if (p1.PowerProduct.Length != p2.PowerProduct.Length)
101                 throw new ArithmeticException();
102             int[] pp = new int[p1.PowerProduct.Length];
103 
104             for (int i = 0; i < pp.Length; i++)
105             {
106                 pp[i] = p1.PowerProduct[i] + p2.PowerProduct[i];
107             }
108 
109             return new Term(p1.Coefficient * p2.Coefficient, pp, p1.ring);
110         }
111 
112         public static Term operator *(int integer, Term p1) => p1 *
                integer;
113         public static Term operator *(Term p1, int integer)
114         {
115             var term = p1.Clone(p1.Owner);
116             term.Coefficient *= integer;
117             return term;
118         }
119 
120         public static Term operator /(Term dividend, Term divisor)
121         {
122             if (divisor.ring != dividend.ring)
123                 throw new ArithmeticException();
124             int[] pp = new int[dividend.PowerProduct.Length];
125 
126             for (int i = 0; i < pp.Length; i++)
127             {
128                 pp[i] = dividend.PowerProduct[i] - divisor.PowerProduct[i];
129             }
130 
131             return new Term(dividend.Coefficient / divisor.Coefficient,
                    pp, dividend.ring);
132         }
133 
134 
135 
136         public Term Clone(Polynomial owner)
137         {
138             return new Term(Coefficient, (int[])PowerProduct.Clone(),
                    ring);
139         }
140 
141 
142     }
143 }
```

Polynomial

Listing 3: Clase Polynomial

```
1 
2 namespace GröbnerBasis.PolynomialRings
3 {
```

```
4       public class Polynomial : IEquatable<Polynomial>
5       {
6           private readonly List<Term> _terms = new List<Term>();
7
8           private List<Term> terms
9           {
10              get
11              {
12                  if (_order != Order)
13                  {
14                      _order = Order;
15                      _terms.Sort(GetComparer());
16
17                  }
18                  return _terms;
19              }
20          }
21
22
23          public Ring Ring { get; private set; }
24
25          public ReadOnlyCollection<Term> Terms
26          {
27              get { return terms.AsReadOnly(); }
28          }
29
30          private MonomialOrder _order;
31          public MonomialOrder Order => Ring.Order;
32
33          public Polynomial(Ring ring, Term[] terms = null)
34          {
35              Ring = ring;
36              _order = Order;
37              if (terms != null)
38                  AddTerms(terms);
39          }
40
41
42          public void AddTerm(Term term)
43          {
44              if (term == null)
45                  return;
46              var epsilon = 0.00001d;
47              if (Ring.Field == Field.Real)
48              {
49                  var round = Math.Round(term.Coefficient);
50                  if (Math.Abs(round - term.Coefficient) <= epsilon)
51                      term.Coefficient = round;
52              }
53
54              if (Math.Abs(term.Coefficient) <= epsilon)
55                  return;
56              Term contained = terms.SingleOrDefault(x =>
                    x.PowerProduct.SequenceEqual(term.PowerProduct));
57              if (contained != null)
58              {
```

```
59                  int index = terms.IndexOf(contained);
60                  terms[index] = contained + term;
61                  terms[index].Owner = this;
62
63                  if (Math.Abs(terms[index].Coefficient) <= epsilon)
64                  {
65                      terms.Remove(terms[index]);
66
67                  }
68              }
69              else
70              {
71                  //Might be interesting to insert the term directly at the
                        appropriate position .
72                  terms.Add(term);
73                  term.Owner = this;
74                  terms.Sort(GetComparer());
75              }
76          }
77
78
79          public void AddTerm(double coef, int[] powerProduct)
80          {
81              dynamic value = coef;
82              if (Ring.Field != Field.Real)
83                  value = CastCoefficient(coef);
84              Term term = new Term(value, powerProduct, Ring); AddTerm(term);
85          }
86
87          public void AddTerm(Complex coef, int[] powerProduct)
88          {
89              dynamic value = coef;
90              if (Ring.Field != Field.Complex)
91                  value = CastCoefficient(coef);
92              Term term = new Term(value, powerProduct, Ring);
93              AddTerm(term);
94          }
95
96          public void AddTerm(int coef, int[] powerProduct)
97          {
98              dynamic value = coef;
99              if (Ring.Field != Field.Integer)
100                 value = CastCoefficient(coef);
101             Term term = new Term(value, powerProduct, Ring);
102             AddTerm(term);
103         }
104
105         public void AddTerms(Term[] terms) => terms.ToList().ForEach(term
                => AddTerm(term));
106
107         public Term LeadingTerm => terms[0];
108
109         public int[] LeadingPowerProduct() => LeadingTerm.PowerProduct;
110
111
112
```

```
113         public bool IsZero()
114         {
115             return terms.Count == 0;
116         }
117 
118         public bool Divides(Polynomial other)
119         {
120             if (terms.Count == 0)
121                 return false;
122 
123 
124             bool divides = LeadingTerm.CompareTo(other.LeadingTerm) >= 0;
125 
126             var pp = other.LeadingTerm.PowerProduct;
127             for (int i = 0; i < pp.Length; i++)
128                 divides = divides && LeadingPowerProduct()[i] <= pp[i];
129 
130             return divides;
131         }
132 
133 
134         public void ReduceCoefficients()
135         {
136             var coef = LeadingTerm.Coefficient;
137             foreach (Term term in terms)
138             {
139                 term.Coefficient /= coef;
140             }
141         }
142 
143         public override string ToString()
144         {
145             string pstring = "";
146             for (int i = 0; i < Terms.Count; i++)
147             {
148                 if (i > 0)
149                 {
150                     var c = (Complex)Terms[i].Coefficient;
151                     pstring += (c.Real >= 0 ? " +" : " ");
152                 }
153                 pstring += Terms[i].ToString();
154             }
155             return pstring;
156         }
157 
158         public Polynomial Clone()
159         {
160             Polynomial copy = new Polynomial(Ring);
161 
162             foreach (Term term in terms)
163                 copy.AddTerm(term.Clone(copy));
164             return copy;
165         }
166 
167         private IComparer<Term> GetComparer()
168         {
```

```
169             switch (Order)
170             {
171                 default:
172                     return new Lexicographical();
173                 case MonomialOrder.deglex:
174                     return new DegreeLexicographical();
175 
176             }
177         }
178 
179         public bool Equals(Polynomial other)
180         {
181             return terms.SequenceEqual(other.Terms);
182         }
183 
184         private dynamic CastCoefficient(dynamic coefficent)
185         {
186             switch (Ring.Field)
187             {
188 
189                 case Field.Integer:
190                     return (long)coefficent;
191                 case Field.Complex:
192                     return (Complex)coefficent;
193                 case Field.Rational:
194                     return (Rational)coefficent;
195                 case Field.Real:
196                 default:
197                     return (double)coefficent;
198             }
199         }
200 
201 
202     }
203 }
```

Ideal

Listing 4: Clase Ideal

```
1  public class Ideal
2  {
3      public Polynomial[] GeneratorSet { get; private set; }
4      private Ring ring;
5 
6      public Ideal(Polynomial[] generator, Ring ring)
7      {
8          GeneratorSet = generator;
9          this.ring = ring;
10     }
11 
12     //BuchbergerÂ´s algorithm
13     public Polynomial[] GröbnerBasis(CancellationToken token = default,
           IProgress<int> progress = null)
14     {
```

```
15          List<Polynomial> G = GeneratorSet.Select(polynomial =>
                polynomial.Clone()).ToList();
16          var combinations = G.SelectMany((x, i) => G.Skip(i + 1), (x, y) =>
                Tuple.Create(x, y)).ToList();
17
18          //These two variables will be used to report the current progress
                of the algorithm
19          int combinationsProcessed = 0;
20          int totalCombinations = G.Count;
21
22          while (combinations.Count != 0)
23          {
24
25              if (progress != null)
26              {
27                  var value = (int) ((double)combinationsProcessed /
                        totalCombinations* 100);
28                  if (value > 100) value = 100;
29                  progress.Report(value);
30                  combinationsProcessed++;
31              }
32              if (token != CancellationToken.None &&
                    token.IsCancellationRequested)
33                  throw new OperationCanceledException();
34              var tuple = combinations.First();
35              combinations.Remove(tuple);
36
37              var sPol = ring.SPolynomial(tuple.Item1, tuple.Item2);
38              var division = ring.Divide(sPol, G.ToArray(),token);
39              Polynomial remainder = division.Item2;
40              if (!remainder.IsZero())
41              {
42                  var newCombinations = G.Select(x => Tuple.Create(x,
                        remainder));
43
44
45                  combinations.AddRange(newCombinations);
46                  G.Add(remainder);
47
48                  totalCombinations += newCombinations.Count();
49              }
50          }
51
52          G.ForEach(p => p.ReduceCoefficients());
53                      return G.ToArray();
54      }
55
56      public Polynomial[] MinimalGröbnerBasis(CancellationToken token =
            default, IProgress<int> progress = null)
57      {
58          var grobnerBasis = GröbnerBasis(token, progress);
59
60          var minimal = new List<Polynomial>(grobnerBasis);
61
62          for (int i = 0; i < grobnerBasis.Length; i++)
63          {
```

```
64              if (grobnerBasis[i] == null)
65                  continue;
66              for (int j = 0; j < grobnerBasis.Length; j++)
67              {
68                  if (grobnerBasis[j] == null)
69                      continue;
70                  if (i != j)
71                  {
72                      if (grobnerBasis[i].Divides(grobnerBasis[j]))
73                      {
74                          minimal.Remove(grobnerBasis[j]);
75                          grobnerBasis[j] = null;
76                      }
77                  }
78              }
79
80          }
81          return minimal.ToArray();
82      }
83
84      public Polynomial[] ReducedGrobnerBasis(CancellationToken token =
            default, IProgress<int> progress = null)
85      {
86          var minimal = MinimalGröbnerBasis(token,progress).ToList();
87          if (minimal == null)
88              return null;
89
90          for (int i = 0; i < minimal.Count; i++)
91          {
92              var g = minimal[i];
93              minimal.Remove(g);
94              var h = ring.Divide(g, minimal.ToArray()).Item2;
95              minimal.Insert(i, h);
96          }
97
98
99          return minimal.ToArray();
100     }
101
102
103     public bool Member(Polynomial polynomial)
104     {
105         return ring.Divide(polynomial,
                ReducedGrobnerBasis()).Item2.IsZero();
106     }
107
108 }
```

Aplicación gráfica de la k-colorabilidad.

Listing 5: Clase de la interfaz de usuario.

```
1
2       public partial class Form1 : Form
```

```
3       {
4
5           private CancellationTokenSource _cts;
6
7           public Form1()
8           {
9
10              graphEditor = new GraphEditor();
11              InitializeComponent();
12              graphEditor.AddNodeType("Ellipse", Shape.Ellipse,
                    Color.Transparent, Color.Black, 10, "user data", "New
                    Node");
13              graphEditor.AddNodeType("Square", Shape.Box,
                    Color.Transparent, Color.Black, 6, "user data", "");
14              graphEditor.AddNodeType("Double Circle", Shape.DoubleCircle,
                    Color.Transparent, Color.Black, 6, "user data", "New Node");
15              graphEditor.AddNodeType("Diamond", Shape.Diamond,
                    Color.Transparent, Color.Black, 6, "user data", "New Node");
16              graphEditor.Viewer.NeedToCalculateLayout = true;
17              CreateGraph();
18              graphEditor.Viewer.NeedToCalculateLayout = false;
19              SuspendLayout();
20
21              graphEditor.Viewer.LayoutAlgorithmSettingsButtonVisible =
                    false;
22              splitContainer1.Panel1.Controls.Add(graphEditor);
23              graphEditor.Dock = DockStyle.Fill;
24              ResumeLayout();
25
26          }
27
28
29          private void CreateGraph()
30          {
31              var g = new Graph();
32
33              //Default graph
34              g.AddEdge("x1", "x2").Attr.ArrowheadAtTarget = ArrowStyle.None;
35              g.AddEdge("x1", "x5").Attr.ArrowheadAtTarget = ArrowStyle.None;
36              g.AddEdge("x1", "x6").Attr.ArrowheadAtTarget = ArrowStyle.None;
37              g.AddEdge("x2", "x8").Attr.ArrowheadAtTarget = ArrowStyle.None;
38              g.AddEdge("x2", "x3").Attr.ArrowheadAtTarget = ArrowStyle.None;
39              g.AddEdge("x2", "x4").Attr.ArrowheadAtTarget = ArrowStyle.None;
40              g.AddEdge("x3", "x8").Attr.ArrowheadAtTarget = ArrowStyle.None;
41              g.AddEdge("x3", "x4").Attr.ArrowheadAtTarget = ArrowStyle.None;
42              g.AddEdge("x4", "x5").Attr.ArrowheadAtTarget = ArrowStyle.None;
43              g.AddEdge("x4", "x7").Attr.ArrowheadAtTarget = ArrowStyle.None;
44              g.AddEdge("x5", "x7").Attr.ArrowheadAtTarget = ArrowStyle.None;
45              g.AddEdge("x5", "x6").Attr.ArrowheadAtTarget = ArrowStyle.None;
46              g.AddEdge("x6", "x7").Attr.ArrowheadAtTarget = ArrowStyle.None;
47              g.AddEdge("x7", "x8").Attr.ArrowheadAtTarget = ArrowStyle.None;
48
49              graphEditor.Graph = g;
50
51          }
52
```

```
53
54
55
56          private async void CheckButton_Click(object sender, EventArgs e)
57          {
58              textBox2.Text = "";
59              textBox3.Text = "";
60
61              _cts = new CancellationTokenSource();
62              var token = _cts.Token;
63              int k = (int)numericUpDown1.Value;
64              progressBar1.Visible = true;
65              CheckButton.Visible = false;
66              cancel.Visible = true;
67
68              var progressHandler = new Progress<int>(value =>
69              {
70                  progressBar1.Value = value;
71              });
72
73              var progress = progressHandler as IProgress<string>;
74              try
75              {
76                  var result = await Task.Run(() =>
77                  {
78                      return Check(k,token, progressHandler);
79                   });
80                  foreach (Polynomial p in result.Item1)
81                      textBox2.Text += p.ToString() + "\r\n";
82
83                  foreach (Polynomial p in result.Item2)
84                      textBox3.Text += p.ToString() + "\r\n";
85                  textBox1.Text = (result.Item3 ? "The graph is " + k +
                        "-colorable." : "The graph is not " + k +
                        "-colorable.");
86
87              }
88              catch (OperationCanceledException)
89              {
90                  MessageBox.Show("Cancelled.");
91
92              }
93              progressBar1.Visible = false;
94              cancel.Visible = false;
95              CheckButton.Visible = true;
96
97          }
98
99
100
101         private  Tuple<Polynomial[],Polynomial [],bool> Check(int
                k,CancellationToken token,IProgress<int> progress)
102         {
103             var graph = graphEditor.Graph;
104             var variables = graph.Nodes.Select(x =>
                    x.Label.Text).ToArray();
```

```
105             foreach (String s in variables)
106                 Console.WriteLine(s);
107             Ring ring = new Ring(Field.Real, variables);
108             ring.FixOrder(variables);
109 
110             List<Polynomial> generator = new List<Polynomial>();
111             List<Node> nodes = graph.Nodes.ToList();
112 
113             float currentProgress = 0;
114             for (int i = 0; i < nodes.Count; i++)
115             {
116                 int[] powerProduct = new int[graph.NodeCount];
117                 powerProduct[i] = k;
118                 Polynomial first = new Polynomial(ring);
119                 first.AddTerm(1, powerProduct);
120                 first.AddTerm(-1, new int[nodes.Count]);
121 
122                 generator.Add(first);
123 
124                 var node = nodes[i];
125                 foreach (Edge edge in node.Edges)
126                 {
127 
128                     Polynomial second = new Polynomial(ring);
129 
130                     int index = nodes.IndexOf(edge.TargetNode);
131                     if (index == i)
132                         continue;
133                     powerProduct = new int[nodes.Count];
134                     powerProduct[i] = k - 1;
135                     second.AddTerm(1, powerProduct);
136                     for (int j = k - 2; j > 0; j--)
137                     {
138                         powerProduct = new int[nodes.Count];
139                         powerProduct[i] = j;
140                         powerProduct[index] = k - 1 - j;
141                         second.AddTerm(1, powerProduct);
142 
143                     }
144                     powerProduct = new int[nodes.Count];
145                     powerProduct[i] = 0;
146                     powerProduct[index] = k - 1;
147                     second.AddTerm(1, powerProduct);
148 
149                     generator.Add(second);
150                 }
151                 currentProgress = i / nodes.Count *100;
152                 progress.Report((int)currentProgress);
153             }
154 
155 
156             Ideal ideal = new Ideal(generator.ToArray(), ring);
157             Polynomial one = new Polynomial(ring);
158             one.AddTerm(1, new int[nodes.Count]);
159             var reducedBasis = ideal.ReducedGrobnerBasis(token,progress);
160 
```

```
161             return new Tuple<Polynomial[],Polynomial[],bool>
                    (ideal.GeneratorSet,reducedBasis, !reducedBasis.Any(i =>
                    i.Equals(one)));
162         }
163
164         private void ProgressChanged(object sender,
                ProgressChangedEventArgs e)
165         {
166             progressBar1.Value = e.ProgressPercentage;
167         }
168         private void Completed(object sender, RunWorkerCompletedEventArgs
                e)
169         {
170             progressBar1.Visible = false;
171             CheckButton.Text = "Comprobar";
172
173         }
174
175         private void cancel_Click(object sender, EventArgs e)
176         {
177             if (_cts != null)
178                 _cts.Cancel();
179         }
180
181         private void label1_Click(object sender, EventArgs e)
182         {
183
184         }
185
186
187         private void tableLayoutPanel1_Paint(object sender, PaintEventArgs
                e)
188         {
189
190         }
191
192         private void numericUpDown1_ValueChanged(object sender, EventArgs
                e)
193         {
194         }
195
196
197         private void label2_Click(object sender, EventArgs e)
198         {
199
200         }
201         private void Form1_Load(object sender, EventArgs e)
202         {
203
204         }
205
206         private void splitContainer1_Panel1_Paint(object sender,
                PaintEventArgs e)
207         {
208
209         }
```

```
210
211         private void splitContainer1_Panel1_Paint_1(object sender,
                PaintEventArgs e)
212         {
213
214         }
215
216         private void splitContainer1_Panel1_Paint_2(object sender,
                PaintEventArgs e)
217         {
218
219         }
220     }
```

Ejemplos

Listing 6: Ejemplos de uso de la Biblioteca.

```
1     static void Ejemplo1()
2     {
3         Ring r = new Ring(Field.Real, new string[] { "x", "y" });
4
5         r.FixOrder(new string[] { "y", "x" });
6
7
8         Polynomial f1 = new Polynomial(r);
9         f1.AddTerm(1, new int[] { 0, 2 });
10        f1.AddTerm(1, new int[] { 1, 1 });
11        f1.AddTerm(1, new int[] { 2, 0 });
12
13
14        Polynomial f2 = new Polynomial(r);
15        f2.AddTerm(1, new int[] { 0, 1 });
16        f2.AddTerm(1, new int[] { 1, 0 });
17
18        Polynomial f3 = new Polynomial(r);
19        f3.AddTerm(1, new int[] { 0, 1 });
20
21        Ideal I = new Ideal(new Polynomial[] { f1, f2, f3 }, r);
22
23        Console.WriteLine("__________Gröbner Basis_____________________");
24        var gb = I.GröbnerBasis();
25        foreach (var p in gb)
26            Console.WriteLine(p);
27
28        Console.WriteLine("__________Minimal Gröbner
              Basis_____________________");
29
30        var minimal = I.MinimalGröbnerBasis();
31        foreach (var p in minimal)
32            Console.WriteLine(p);
33
34        Console.WriteLine("_______________Reduced Gröbner
              Basis_______________");
```

```
35
36
37          var reduced = I.ReducedGrobnerBasis();
38          foreach (var p in reduced)
39              Console.WriteLine(p);
40
41          Console.WriteLine("__________Membership_____________________");
42          Console.WriteLine(I.Member(f1));
43      }
44
45      //Graph with 8 nodes
46      static void Ejemplo2()
47      {
48          Ring ring = new Ring(Field.Real, new string[] { "x1", "x2", "x3",
                "x4", "x5", "x6", "x7", "x8" });
49          ring.FixOrder(new string[] { "x1", "x2", "x3", "x4", "x5", "x6",
                "x7", "x8" });
50
51
52          Polynomial f1 = new Polynomial(ring);
53          f1.AddTerm(1, new int[] { 3, 0, 0, 0, 0, 0, 0, 0 });
54          f1.AddTerm(-1, new int[] { 0, 0, 0, 0, 0, 0, 0, 0 });
55
56          Polynomial f2 = new Polynomial(ring);
57          f2.AddTerm(1, new int[] { 0, 3, 0, 0, 0, 0, 0, 0 });
58          f2.AddTerm(-1, new int[] { 0, 0, 0, 0, 0, 0, 0, 0 });
59
60          Polynomial f3 = new Polynomial(ring);
61          f3.AddTerm(1, new int[] { 0, 0, 3, 0, 0, 0, 0, 0 });
62          f3.AddTerm(-1, new int[] { 0, 0, 0, 0, 0, 0, 0, 0 });
63
64          Polynomial f4 = new Polynomial(ring);
65          f4.AddTerm(1, new int[] { 0, 0, 0, 3, 0, 0, 0, 0 });
66          f4.AddTerm(-1, new int[] { 0, 0, 0, 0, 0, 0, 0, 0 });
67
68          Polynomial f5 = new Polynomial(ring);
69          f5.AddTerm(1, new int[] { 0, 0, 0, 0, 3, 0, 0, 0 });
70          f5.AddTerm(-1, new int[] { 0, 0, 0, 0, 0, 0, 0, 0 });
71
72          Polynomial f6 = new Polynomial(ring);
73          f6.AddTerm(1, new int[] { 0, 0, 0, 0, 0, 3, 0, 0 });
74          f6.AddTerm(-1, new int[] { 0, 0, 0, 0, 0, 0, 0, 0 });
75
76          Polynomial f7 = new Polynomial(ring);
77          f7.AddTerm(1, new int[] { 0, 0, 0, 0, 0, 0, 3, 0 });
78          f7.AddTerm(-1, new int[] { 0, 0, 0, 0, 0, 0, 0, 0 });
79
80          Polynomial f8 = new Polynomial(ring);
81          f8.AddTerm(1, new int[] { 0, 0, 0, 0, 0, 0, 0, 3 });
82          f8.AddTerm(-1, new int[] { 0, 0, 0, 0, 0, 0, 0, 0 });
83
84          Polynomial f9 = new Polynomial(ring);
85          f9.AddTerm(1, new int[] { 2, 0, 0, 0, 0, 0, 0, 0 });
86          f9.AddTerm(1, new int[] { 1, 1, 0, 0, 0, 0, 0, 0 });
87          f9.AddTerm(1, new int[] { 0, 2, 0, 0, 0, 0, 0, 0 });
88
```

```
89
90          Polynomial f10 = new Polynomial(ring);
91          f10.AddTerm(1, new int[] { 2, 0, 0, 0, 0, 0, 0, 0 });
92          f10.AddTerm(1, new int[] { 1, 0, 0, 0, 1, 0, 0, 0 });
93          f10.AddTerm(1, new int[] { 0, 0, 0, 0, 2, 0, 0, 0 });
94
95          Polynomial f11 = new Polynomial(ring);
96          f11.AddTerm(1, new int[] { 2, 0, 0, 0, 0, 0, 0, 0 });
97          f11.AddTerm(1, new int[] { 1, 0, 0, 0, 0, 1, 0, 0 });
98          f11.AddTerm(1, new int[] { 0, 0, 0, 0, 0, 2, 0, 0 });
99
100
101         Polynomial f12 = new Polynomial(ring);
102         f12.AddTerm(1, new int[] { 0, 2, 0, 0, 0, 0, 0, 0 });
103         f12.AddTerm(1, new int[] { 0, 1, 0, 0, 0, 0, 0, 1 });
104         f12.AddTerm(1, new int[] { 0, 0, 0, 0, 0, 0, 0, 2 });
105
106         Polynomial f13 = new Polynomial(ring);
107         f13.AddTerm(1, new int[] { 0, 2, 0, 0, 0, 0, 0, 0 });
108         f13.AddTerm(1, new int[] { 0, 1, 1, 0, 0, 0, 0, 0 });
109         f13.AddTerm(1, new int[] { 0, 0, 2, 0, 0, 0, 0, 0 });
110
111         Polynomial f14 = new Polynomial(ring);
112         f14.AddTerm(1, new int[] { 0, 2, 0, 0, 0, 0, 0, 0 });
113         f14.AddTerm(1, new int[] { 0, 1, 0, 1, 0, 0, 0, 0 });
114         f14.AddTerm(1, new int[] { 0, 0, 0, 2, 0, 0, 0, 0 });
115
116         Polynomial f15 = new Polynomial(ring);
117         f15.AddTerm(1, new int[] { 0, 0, 2, 0, 0, 0, 0, 0 });
118         f15.AddTerm(1, new int[] { 0, 0, 1, 0, 0, 0, 0, 1 });
119         f15.AddTerm(1, new int[] { 0, 0, 0, 0, 0, 0, 0, 2 });
120
121         Polynomial f16 = new Polynomial(ring);
122         f16.AddTerm(1, new int[] { 0, 0, 2, 0, 0, 0, 0, 0 });
123         f16.AddTerm(1, new int[] { 0, 0, 1, 1, 0, 0, 0, 0 });
124         f16.AddTerm(1, new int[] { 0, 0, 0, 2, 0, 0, 0, 0 });
125
126         Polynomial f17 = new Polynomial(ring);
127         f17.AddTerm(1, new int[] { 0, 0, 0, 2, 0, 0, 0, 0 });
128         f17.AddTerm(1, new int[] { 0, 0, 0, 1, 1, 0, 0, 0 });
129         f17.AddTerm(1, new int[] { 0, 0, 0, 0, 2, 0, 0, 0 });
130
131         Polynomial f18 = new Polynomial(ring);
132         f18.AddTerm(1, new int[] { 0, 0, 0, 2, 0, 0, 0, 0 });
133         f18.AddTerm(1, new int[] { 0, 0, 0, 1, 0, 0, 1, 0 });
134         f18.AddTerm(1, new int[] { 0, 0, 0, 0, 0, 0, 2, 0 });
135
136         Polynomial f19 = new Polynomial(ring);
137         f19.AddTerm(1, new int[] { 0, 0, 0, 0, 2, 0, 0, 0 });
138         f19.AddTerm(1, new int[] { 0, 0, 0, 0, 1, 0, 1, 0 });
139         f19.AddTerm(1, new int[] { 0, 0, 0, 0, 0, 0, 2, 0 });
140
141         Polynomial f20 = new Polynomial(ring);
142         f20.AddTerm(1, new int[] { 0, 0, 0, 0, 2, 0, 0, 0 });
143         f20.AddTerm(1, new int[] { 0, 0, 0, 0, 1, 1, 0, 0 });
144         f20.AddTerm(1, new int[] { 0, 0, 0, 0, 0, 2, 0, 0 });
```

```
145
146         Polynomial f21 = new Polynomial(ring);
147         f21.AddTerm(1, new int[] { 0, 0, 0, 0, 0, 2, 0, 0 });
148         f21.AddTerm(1, new int[] { 0, 0, 0, 0, 0, 1, 1, 0 });
149         f21.AddTerm(1, new int[] { 0, 0, 0, 0, 0, 0, 2, 0 });
150
151         Polynomial f22 = new Polynomial(ring);
152         f22.AddTerm(1, new int[] { 0, 0, 0, 0, 0, 2, 0, 0 });
153         f22.AddTerm(1, new int[] { 0, 0, 0, 0, 0, 1, 0, 1 });
154         f22.AddTerm(1, new int[] { 0, 0, 0, 0, 0, 0, 0, 2 });
155
156         Ideal I = new Ideal(new Polynomial[] { f1, f2, f3, f4, f5, f6, f7,
                f8, f9, f10 }, ring);
157         Console.WriteLine("__________Gröbner Basis__________________");
158         var gb = I.GröbnerBasis();
159         foreach (var p in gb)
160             Console.WriteLine(p);
161
162         Console.WriteLine("_________Minimal Gröbner
                Basis_____________________");
163
164         var minimal = I.MinimalGröbnerBasis();
165         foreach (var p in minimal)
166             Console.WriteLine(p);
167
168         Console.WriteLine("_____________Reduced Gröbner
                Basis_____________");
169
170
171         var reduced = I.ReducedGrobnerBasis();
172         foreach (var p in reduced)
173             Console.WriteLine(p);
174
175         Console.WriteLine("__________Membership__________________");
176         Console.WriteLine(I.Member(f1));
177
178     }
179
180
181     static void Ejemplo3()
182     {
183         Ring r = new Ring(Field.Real, new string[] { "x", "y" });
184
185         r.FixOrder(new string[] { "y", "x" });
186
187
188         Polynomial f1 = new Polynomial(r);
189         f1.AddTerm(1, new int[] { 1, 1 });
190         f1.AddTerm(-1, new int[] { 1, 0 });
191
192
193         Polynomial f2 = new Polynomial(r);
194         f2.AddTerm(1, new int[] { 0, 1 });
195         f2.AddTerm(-1, new int[] { 2, 0 });
196
197         Polynomial f3 = new Polynomial(r);
```

```
198         f3.AddTerm(1, new int[] { 3, 0 });
199         f3.AddTerm(-1, new int[] { 1, 0 });
200
201
202
203         Ideal I = new Ideal(new Polynomial[] { f2, f3 }, r);
204         Console.WriteLine("__________Gröbner Basis__________________");
205         var gb = I.GröbnerBasis();
206         foreach (var p in gb)
207             Console.WriteLine(p);
208
209         Console.WriteLine("_________Minimal Gröbner
                Basis_____________________");
210
211         var minimal = I.MinimalGröbnerBasis();
212         foreach (var p in minimal)
213             Console.WriteLine(p);
214
215         Console.WriteLine("_____________Reduced Gröbner
                Basis_____________");
216
217
218         var reduced = I.ReducedGrobnerBasis();
219         foreach (var p in reduced)
220             Console.WriteLine(p);
221
222         Console.WriteLine("__________Membership__________________");
223         Console.WriteLine(I.Member(f1));
224     }
225
226     //Graph with 3 nodes and 3-colorable
227     static void Ejemplo4()
228     {
229         Ring r = new Ring(Field.Real, new string[] { "x", "y", "z" });
230
231         r.FixOrder(new string[] { "x", "y", "z" });
232
233
234         Polynomial f1 = new Polynomial(r);
235         f1.AddTerm(1, new int[] { 3, 0, 0 });
236         f1.AddTerm(-1, new int[] { 0, 0, 0 });
237
238
239         Polynomial f2 = new Polynomial(r);
240         f2.AddTerm(1, new int[] { 0, 3, 0 });
241         f2.AddTerm(-1, new int[] { 0, 0, 0 });
242
243         Polynomial f3 = new Polynomial(r);
244         f3.AddTerm(1, new int[] { 0, 0, 3 });
245         f3.AddTerm(-1, new int[] { 0, 0, 0 });
246
247         Polynomial f4 = new Polynomial(r);
248         f4.AddTerm(1, new int[] { 0, 2, 0 });
249         f4.AddTerm(1, new int[] { 0, 1, 1 });
250         f4.AddTerm(1, new int[] { 0, 0, 2 });
251
```

```
252         Polynomial f5 = new Polynomial(r);
253         f5.AddTerm(1, new int[] { 2, 0, 0 });
254         f5.AddTerm(1, new int[] { 1, 0, 1 });
255         f5.AddTerm(1, new int[] { 0, 0, 2 });
256 
257         Polynomial f6 = new Polynomial(r);
258         f6.AddTerm(1, new int[] { 2, 0, 0 });
259         f6.AddTerm(1, new int[] { 1, 1, 0 });
260         f6.AddTerm(1, new int[] { 0, 2, 0 });
261 
262         Ideal I = new Ideal(new Polynomial[] { f1, f2, f3, f6, f5, f4 },
                r);
263         Console.WriteLine("__________Gröbner Basis_____________________");
264         var gb = I.GröbnerBasis();
265         foreach (var p in gb)
266             Console.WriteLine(p);
267 
268         Console.WriteLine("_________Minimal Gröbner
                Basis_____________________");
269 
270         var minimal = I.MinimalGröbnerBasis();
271         foreach (var p in minimal)
272             Console.WriteLine(p);
273 
274         Console.WriteLine("_______________Reduced Gröbner
                Basis_______________");
275 
276 
277         var reduced = I.ReducedGrobnerBasis();
278         foreach (var p in reduced)
279             Console.WriteLine(p);
280 
281         Console.WriteLine("__________Membership_____________________");
282         Console.WriteLine(I.Member(f1));
283     }
284 
285 
286     //Division Ejemplo for a case in the previous Ejemplo
287     static void Ejemplo5()
288     {
289         Ring r = new Ring(Field.Real, new string[] { "x", "y", "z" });
290 
291         r.FixOrder(new string[] { "x", "y", "z" });
292 
293 
294         Polynomial f1 = new Polynomial(r);
295         f1.AddTerm(1, new int[] { 3, 0, 0 });
296         f1.AddTerm(-1, new int[] { 0, 0, 0 });
297 
298 
299         Polynomial f2 = new Polynomial(r);
300         f2.AddTerm(1, new int[] { 0, 3, 0 });
301         f2.AddTerm(-1, new int[] { 0, 0, 0 });
302 
303         Polynomial f3 = new Polynomial(r);
304         f3.AddTerm(1, new int[] { 0, 0, 3 });
```

```
305         f3.AddTerm(-1, new int[] { 0, 0, 0 });
306
307         Polynomial f4 = new Polynomial(r);
308         f4.AddTerm(1, new int[] { 0, 2, 0 });
309         f4.AddTerm(1, new int[] { 0, 1, 1 });
310         f4.AddTerm(1, new int[] { 0, 0, 2 });
311
312         Polynomial f5 = new Polynomial(r);
313         f5.AddTerm(1, new int[] { 2, 0, 0 });
314         f5.AddTerm(1, new int[] { 1, 0, 1 });
315         f5.AddTerm(1, new int[] { 0, 0, 2 });
316
317         Polynomial f6 = new Polynomial(r);
318         f6.AddTerm(1, new int[] { 2, 0, 0 });
319         f6.AddTerm(1, new int[] { 1, 1, 0 });
320         f6.AddTerm(1, new int[] { 0, 2, 0 });
321
322
323         Polynomial dividend = new Polynomial(r);
324         dividend.AddTerm(-1, new int[] { 2, 1, 0 });
325         dividend.AddTerm(-1, new int[] { 1, 2, 0 });
326         dividend.AddTerm(-1, new int[] { 0, 0, 0 });
327
328
329         Console.WriteLine("-----------------------------------");
330         var res = r.Divide(dividend, new Polynomial[] { f1, f2, f3, f6,
                f5, f4 });
331         Console.WriteLine(res.Item2);
332         Console.WriteLine("-----------------------------------");
333
334
335     }
336     //Graph with 4 nodes and 3-colorable
337     static void Ejemplo6()
338     {
339         Ring r = new Ring(Field.Real, new string[] { "x", "y", "z", "t" });
340
341         r.FixOrder(new string[] { "x", "y", "z", "t" });
342
343
344         Polynomial f1 = new Polynomial(r);
345         f1.AddTerm(1, new int[] { 3, 0, 0, 0 });
346         f1.AddTerm(-1, new int[] { 0, 0, 0, 0 });
347
348
349         Polynomial f2 = new Polynomial(r);
350         f2.AddTerm(1, new int[] { 0, 3, 0, 0 });
351         f2.AddTerm(-1, new int[] { 0, 0, 0, 0 });
352
353         Polynomial f3 = new Polynomial(r);
354         f3.AddTerm(1, new int[] { 0, 0, 3, 0 });
355         f3.AddTerm(-1, new int[] { 0, 0, 0, 0 });
356
357         Polynomial f4 = new Polynomial(r);
358         f4.AddTerm(1, new int[] { 0, 2, 0, 0 });
359         f4.AddTerm(1, new int[] { 0, 1, 1, 0 });
```

```
360         f4.AddTerm(1, new int[] { 0, 0, 2, 0 });
361
362         Polynomial f5 = new Polynomial(r);
363         f5.AddTerm(1, new int[] { 2, 0, 0, 0 });
364         f5.AddTerm(1, new int[] { 1, 0, 1, 0 });
365         f5.AddTerm(1, new int[] { 0, 0, 2, 0 });
366
367         Polynomial f6 = new Polynomial(r);
368         f6.AddTerm(1, new int[] { 2, 0, 0, 0 });
369         f6.AddTerm(1, new int[] { 1, 1, 0, 0 });
370         f6.AddTerm(1, new int[] { 0, 2, 0, 0 });
371
372
373
374         Polynomial f7 = new Polynomial(r);
375         f7.AddTerm(1, new int[] { 0, 0, 0, 3 });
376         f7.AddTerm(-1, new int[] { 0, 0, 0, 0 });
377
378         Polynomial f8 = new Polynomial(r);
379         f8.AddTerm(1, new int[] { 2, 0, 0, 0 });
380         f8.AddTerm(1, new int[] { 1, 0, 0, 1 });
381         f8.AddTerm(1, new int[] { 0, 0, 0, 2 });
382
383
384         Ideal I = new Ideal(new Polynomial[] { f1, f2, f3, f6, f5, f4, f7,
                f8 }, r);
385         Console.WriteLine("__________Gröbner Basis__________________");
386         var gb = I.GröbnerBasis();
387         foreach (var p in gb)
388             Console.WriteLine(p);
389
390         Console.WriteLine("_________Minimal Gröbner
                Basis_____________________");
391
392         var minimal = I.MinimalGröbnerBasis();
393         foreach (var p in minimal)
394             Console.WriteLine(p);
395
396         Console.WriteLine("_____________Reduced Gröbner
                Basis_____________");
397
398
399         var reduced = I.ReducedGrobnerBasis();
400         foreach (var p in reduced)
401             Console.WriteLine(p);
402
403         Console.WriteLine("__________Membership__________________");
404         Console.WriteLine(I.Member(f1));
405     }
406
407
408     //Graph with 4 nodes and 3-colorable
409     static void Ejemplo7()
410     {
411         Ring r = new Ring(Field.Real, new string[] { "x", "y", "z", "t" });
412
```

```
413         r.FixOrder(new string[] { "x", "y", "z", "t" });
414
415
416         Polynomial f1 = new Polynomial(r);
417         f1.AddTerm(1, new int[] { 3, 0, 0, 0 });
418         f1.AddTerm(-1, new int[] { 0, 0, 0, 0 });
419
420
421         Polynomial f2 = new Polynomial(r);
422         f2.AddTerm(1, new int[] { 0, 3, 0, 0 });
423         f2.AddTerm(-1, new int[] { 0, 0, 0, 0 });
424
425         Polynomial f3 = new Polynomial(r);
426         f3.AddTerm(1, new int[] { 0, 0, 3, 0 });
427         f3.AddTerm(-1, new int[] { 0, 0, 0, 0 });
428
429         Polynomial f4 = new Polynomial(r);
430         f4.AddTerm(1, new int[] { 0, 2, 0, 0 });
431         f4.AddTerm(1, new int[] { 0, 1, 1, 0 });
432         f4.AddTerm(1, new int[] { 0, 0, 2, 0 });
433
434         Polynomial f5 = new Polynomial(r);
435         f5.AddTerm(1, new int[] { 2, 0, 0, 0 });
436         f5.AddTerm(1, new int[] { 1, 0, 1, 0 });
437         f5.AddTerm(1, new int[] { 0, 0, 2, 0 });
438
439         Polynomial f6 = new Polynomial(r);
440         f6.AddTerm(1, new int[] { 2, 0, 0, 0 });
441         f6.AddTerm(1, new int[] { 1, 1, 0, 0 });
442         f6.AddTerm(1, new int[] { 0, 2, 0, 0 });
443
444
445
446         Polynomial f7 = new Polynomial(r);
447         f7.AddTerm(1, new int[] { 0, 0, 0, 3 });
448         f7.AddTerm(-1, new int[] { 0, 0, 0, 0 });
449
450         Polynomial f8 = new Polynomial(r);
451         f8.AddTerm(1, new int[] { 2, 0, 0, 0 });
452         f8.AddTerm(1, new int[] { 1, 0, 0, 1 });
453         f8.AddTerm(1, new int[] { 0, 0, 0, 2 });
454
455         Polynomial f9 = new Polynomial(r);
456         f9.AddTerm(1, new int[] { 0, 0, 2, 0 });
457         f9.AddTerm(1, new int[] { 0, 0, 1, 1 });
458         f9.AddTerm(1, new int[] { 0, 0, 0, 2 });
459
460
461
462         Ideal I = new Ideal(new Polynomial[] { f1, f2, f3, f6, f5, f4, f7,
                f8, f9 }, r);
463         Console.WriteLine("__________Gröbner Basis__________________");
464         var gb = I.GröbnerBasis();
465         foreach (var p in gb)
466             Console.WriteLine(p);
467
```

```
468         Console.WriteLine("_________Minimal Gröbner
                Basis_____________________");
469
470         var minimal = I.MinimalGröbnerBasis();
471         foreach (var p in minimal)
472             Console.WriteLine(p);
473
474         Console.WriteLine("_______________Reduced Gröbner
                Basis_______________");
475
476
477         var reduced = I.ReducedGrobnerBasis();
478         foreach (var p in reduced)
479             Console.WriteLine(p);
480
481         Console.WriteLine("__________Membership___________________");
482         Console.WriteLine(I.Member(f1));
483     }
484
485     //Graph with 4 nodes and not 3-colorable
486     static void Ejemplo8()
487     {
488         Ring r = new Ring(Field.Real, new string[] { "x", "y", "z", "t" });
489
490         r.FixOrder(new string[] { "x", "y", "z", "t" });
491
492
493         Polynomial f1 = new Polynomial(r);
494         f1.AddTerm(1, new int[] { 3, 0, 0, 0 });
495         f1.AddTerm(-1, new int[] { 0, 0, 0, 0 });
496
497
498         Polynomial f2 = new Polynomial(r);
499         f2.AddTerm(1, new int[] { 0, 3, 0, 0 });
500         f2.AddTerm(-1, new int[] { 0, 0, 0, 0 });
501
502         Polynomial f3 = new Polynomial(r);
503         f3.AddTerm(1, new int[] { 0, 0, 3, 0 });
504         f3.AddTerm(-1, new int[] { 0, 0, 0, 0 });
505
506         Polynomial f4 = new Polynomial(r);
507         f4.AddTerm(1, new int[] { 0, 2, 0, 0 });
508         f4.AddTerm(1, new int[] { 0, 1, 1, 0 });
509         f4.AddTerm(1, new int[] { 0, 0, 2, 0 });
510
511         Polynomial f5 = new Polynomial(r);
512         f5.AddTerm(1, new int[] { 2, 0, 0, 0 });
513         f5.AddTerm(1, new int[] { 1, 0, 1, 0 });
514         f5.AddTerm(1, new int[] { 0, 0, 2, 0 });
515
516         Polynomial f6 = new Polynomial(r);
517         f6.AddTerm(1, new int[] { 2, 0, 0, 0 });
518         f6.AddTerm(1, new int[] { 1, 1, 0, 0 });
519         f6.AddTerm(1, new int[] { 0, 2, 0, 0 });
520
521
```

```
522
523         Polynomial f7 = new Polynomial(r);
524         f7.AddTerm(1, new int[] { 0, 0, 0, 3 });
525         f7.AddTerm(-1, new int[] { 0, 0, 0, 0 });
526
527         Polynomial f8 = new Polynomial(r);
528         f8.AddTerm(1, new int[] { 2, 0, 0, 0 });
529         f8.AddTerm(1, new int[] { 1, 0, 0, 1 });
530         f8.AddTerm(1, new int[] { 0, 0, 0, 2 });
531
532         Polynomial f9 = new Polynomial(r);
533         f9.AddTerm(1, new int[] { 0, 0, 2, 0 });
534         f9.AddTerm(1, new int[] { 0, 0, 1, 1 });
535         f9.AddTerm(1, new int[] { 0, 0, 0, 2 });
536
537
538         Polynomial f10 = new Polynomial(r);
539         f10.AddTerm(1, new int[] { 0, 2, 0, 0 });
540         f10.AddTerm(1, new int[] { 0, 1, 0, 1 });
541         f10.AddTerm(1, new int[] { 0, 0, 0, 2 });
542
543
544         Ideal I = new Ideal(new Polynomial[] { f1, f2, f6, f3, f9, f4, f5,
                f7, f10, f8 }, r);
545
546         Console.WriteLine("__________Generator__________________");
547         var sys = I.GeneratorSet;
548         foreach (var p in sys)
549             Console.WriteLine(p);
550
551
552         Console.WriteLine("__________Gröbner Basis_____________________");
553         var gb = I.GröbnerBasis();
554         foreach (var p in gb)
555             Console.WriteLine(p);
556
557         Console.WriteLine("_________Minimal Gröbner
                Basis_____________________");
558
559         var minimal = I.MinimalGröbnerBasis();
560         foreach (var p in minimal)
561             Console.WriteLine(p);
562
563         Console.WriteLine("_____________Reduced Gröbner
                Basis_____________");
564
565
566         var reduced = I.ReducedGrobnerBasis();
567         foreach (var p in reduced)
568             Console.WriteLine(p);
569
570         Console.WriteLine("__________Membership__________________");
571         Console.WriteLine(I.Member(f1));
572     }
573
574     static void Ejemplo9()
```

```
575     {
576         Ring ring = new Ring(Field.Real, new string[] { "x1", "x2", "x3",
                "x4", "x5", "x6", "x7", "x8" });
577         ring.FixOrder(new string[] { "x1", "x2", "x3", "x4", "x5", "x6",
                "x7", "x8" });
578
579
580         Polynomial f1 = new Polynomial(ring);
581         f1.AddTerm(1, new int[] { 3, 0, 0, 0, 0, 0, 0, 0 });
582         f1.AddTerm(-1, new int[] { 0, 0, 0, 0, 0, 0, 0, 0 });
583
584         Polynomial f2 = new Polynomial(ring);
585         f2.AddTerm(1, new int[] { 0, 3, 0, 0, 0, 0, 0, 0 });
586         f2.AddTerm(-1, new int[] { 0, 0, 0, 0, 0, 0, 0, 0 });
587
588         Polynomial f3 = new Polynomial(ring);
589         f3.AddTerm(1, new int[] { 0, 0, 3, 0, 0, 0, 0, 0 });
590         f3.AddTerm(-1, new int[] { 0, 0, 0, 0, 0, 0, 0, 0 });
591
592         Polynomial f4 = new Polynomial(ring);
593         f4.AddTerm(1, new int[] { 0, 0, 0, 3, 0, 0, 0, 0 });
594         f4.AddTerm(-1, new int[] { 0, 0, 0, 0, 0, 0, 0, 0 });
595
596         Polynomial f5 = new Polynomial(ring);
597         f5.AddTerm(1, new int[] { 0, 0, 0, 0, 3, 0, 0, 0 });
598         f5.AddTerm(-1, new int[] { 0, 0, 0, 0, 0, 0, 0, 0 });
599
600         Polynomial f6 = new Polynomial(ring);
601         f6.AddTerm(1, new int[] { 0, 0, 0, 0, 0, 3, 0, 0 });
602         f6.AddTerm(-1, new int[] { 0, 0, 0, 0, 0, 0, 0, 0 });
603
604         Polynomial f7 = new Polynomial(ring);
605         f7.AddTerm(1, new int[] { 0, 0, 0, 0, 0, 0, 3, 0 });
606         f7.AddTerm(-1, new int[] { 0, 0, 0, 0, 0, 0, 0, 0 });
607
608         Polynomial f8 = new Polynomial(ring);
609         f8.AddTerm(1, new int[] { 0, 0, 0, 0, 0, 0, 0, 3 });
610         f8.AddTerm(-1, new int[] { 0, 0, 0, 0, 0, 0, 0, 0 });
611
612         Polynomial f9 = new Polynomial(ring);
613         f9.AddTerm(1, new int[] { 2, 0, 0, 0, 0, 0, 0, 0 });
614         f9.AddTerm(1, new int[] { 1, 1, 0, 0, 0, 0, 0, 0 });
615         f9.AddTerm(1, new int[] { 0, 2, 0, 0, 0, 0, 0, 0 });
616
617
618         Polynomial f10 = new Polynomial(ring);
619         f10.AddTerm(1, new int[] { 0, 2, 0, 0, 0, 0, 0, 0 });
620         f10.AddTerm(1, new int[] { 0, 1, 1, 0, 0, 0, 0, 0 });
621         f10.AddTerm(1, new int[] { 0, 0, 2, 0, 0, 0, 0, 0 });
622
623         Polynomial f11 = new Polynomial(ring);
624         f11.AddTerm(1, new int[] { 0, 0, 2, 0, 0, 0, 0, 0 });
625         f11.AddTerm(1, new int[] { 0, 0, 1, 1, 0, 0, 0, 0 });
626         f11.AddTerm(1, new int[] { 0, 0, 0, 2, 0, 0, 0, 0 });
627
628
```

```
629         Polynomial f12 = new Polynomial(ring);
630         f12.AddTerm(1, new int[] { 0, 0, 0, 2, 0, 0, 0, 0 });
631         f12.AddTerm(1, new int[] { 0, 0, 0, 1, 1, 0, 0, 0 });
632         f12.AddTerm(1, new int[] { 0, 0, 0, 0, 2, 0, 0, 0 });
633 
634         Polynomial f13 = new Polynomial(ring);
635         f13.AddTerm(1, new int[] { 0, 0, 0, 0, 2, 0, 0, 0 });
636         f13.AddTerm(1, new int[] { 0, 0, 0, 0, 1, 1, 0, 0 });
637         f13.AddTerm(1, new int[] { 0, 0, 0, 0, 0, 2, 0, 0 });
638 
639         Polynomial f14 = new Polynomial(ring);
640         f14.AddTerm(1, new int[] { 0, 0, 0, 0, 0, 2, 0, 0 });
641         f14.AddTerm(1, new int[] { 0, 0, 0, 0, 0, 1, 1, 0 });
642         f14.AddTerm(1, new int[] { 0, 0, 0, 0, 0, 0, 2, 0 });
643 
644         Polynomial f15 = new Polynomial(ring);
645         f15.AddTerm(1, new int[] { 0, 0, 0, 0, 0, 0, 2, 0 });
646         f15.AddTerm(1, new int[] { 0, 0, 0, 0, 0, 0, 1, 1 });
647         f15.AddTerm(1, new int[] { 0, 0, 0, 0, 0, 0, 0, 2 });
648 
649         Ideal I = new Ideal(new Polynomial[] { f1, f2, f3, f4, f5, f6, f7,
                f8, f9, f10, f11, f12, f13, f14, f15 }, ring);
650         Console.WriteLine("__________Gröbner Basis__________________");
651         var gb = I.GröbnerBasis();
652         foreach (var p in gb)
653             Console.WriteLine(p);
654 
655         Console.WriteLine("_________Minimal Gröbner
                Basis_____________________");
656 
657         var minimal = I.MinimalGröbnerBasis();
658         foreach (var p in minimal)
659             Console.WriteLine(p);
660 
661         Console.WriteLine("_____________Reduced Gröbner
                Basis_____________");
662 
663 
664         var reduced = I.ReducedGrobnerBasis();
665         foreach (var p in reduced)
666             Console.WriteLine(p);
667 
668         Console.WriteLine("__________Membership__________________");
669         Console.WriteLine(I.Member(f1));
670     }
671     static void Ejemplo10()
672     {
673         Ring r = new Ring(Field.Real, new string[] { "x1", "x2", "x3",
                "x4" });
674 
675         r.FixOrder(new string[] { "x2", "x3", "x4", "x1" });
676 
677         // x2*x3 -x2*x4 + x3^2 -x4^2
678         Polynomial f1 = new Polynomial(r);
679         f1.AddTerm(1, new int[] { 0, 1, 1, 0 });
680         f1.AddTerm(-1, new int[] { 0, 1, 0, 1 });
```

```
681         f1.AddTerm(1, new int[] { 0, 0, 2, 0 });
682         f1.AddTerm(-1, new int[] { 0, 0, 0, 2 });
683 
684         //x3^2 + x3*x1 + x1^2
685         Polynomial f2 = new Polynomial(r);
686         f2.AddTerm(1, new int[] { 0, 0, 2, 0 });
687         f2.AddTerm(-1, new int[] { 1, 0, 1, 0 });
688         f2.AddTerm(-1, new int[] { 1, 0, 1, 0 });
689 
690         Console.WriteLine(f1 + " divided by " + f2);
691         Console.WriteLine("__________________STEPS____________________");
692 
693         var div = r.Divide(f1, new Polynomial[] { f2 });
694         Console.WriteLine("__________________RESULT____________________");
695 
696         Console.WriteLine("Remainder: " + div.Item2);
697         Console.WriteLine("Quotients:");
698 
699         for (int i = 0; i < div.Item1.Length; i++)
700             Console.WriteLine(i.ToString() + ") " + div.Item1[i]);
701 
702 
703     }
704 
705     static void Ejemplo11()
706     {
707         Ring r = new Ring(Field.Real, new string[] { "x1", "x2", "x3",
                "x4" });
708 
709         r.FixOrder(new string[] { "x2", "x3", "x4", "x1" });
710 
711         // x2*x3 -x2*x4 + x3^2 -x4^2
712         Polynomial f1 = new Polynomial(r);
713         f1.AddTerm(1, new int[] { 0, 1, 1, 0 });
714         f1.AddTerm(-1, new int[] { 0, 1, 0, 1 });
715         f1.AddTerm(1, new int[] { 0, 0, 2, 0 });
716         f1.AddTerm(-1, new int[] { 0, 0, 0, 2 });
717 
718         //x3^2 + x3*x1 + x1^2
719         Polynomial f2 = new Polynomial(r);
720         f2.AddTerm(1, new int[] { 0, 0, 2, 0 });
721         f2.AddTerm(-1, new int[] { 1, 0, 1, 0 });
722         f2.AddTerm(-1, new int[] { 1, 0, 1, 0 });
723 
724         Console.WriteLine(f1 + " divided by " + f2);
725         Console.WriteLine("__________________STEPS____________________");
726 
727         var div = r.Divide(f1, new Polynomial[] { f2 });
728         Console.WriteLine("__________________RESULT____________________");
729 
730         Console.WriteLine("Remainder: " + div.Item2);
731         Console.WriteLine("Quotients:");
732 
733         for (int i = 0; i < div.Item1.Length; i++)
734             Console.WriteLine(i.ToString() + ") " + div.Item1[i]);
735 
```

```
736
737     }
738
739
740
741     static void Ejemplo12()
742     {
743         Ring r = new Ring(Field.Real, new string[] { "x1", "x2", "x3",
                "x4" });
744
745         r.FixOrder(new string[] { "x2", "x3", "x4", "x1" });
746
747         // x2*x3 -x2*x4 + x3^2 -x4^2
748         Polynomial f1 = new Polynomial(r);
749         f1.AddTerm(-0.666666666666667, new int[] { 4, 0, 0, 2 });
750         f1.AddTerm(0.666666666666667, new int[] { 1, 0, 0, 2 });
751         f1.AddTerm(-0.333333333333333, new int[] { 3, 0, 0, 0 });
752         f1.AddTerm(0.333333333333333, new int[] { 0, 0, 0, 0 });
753
754         //x3^2 + x3*x1 + x1^2
755         Polynomial f2 = new Polynomial(r);
756         f2.AddTerm(1, new int[] { 3, 0, 0, 0 });
757         f2.AddTerm(-1, new int[] { 0, 0, 0, 0 });
758
759         Console.WriteLine(f1 + " divided by " + f2);
760         Console.WriteLine("_____________________STEPS_____________________");
761
762         var div = r.Divide(f1, new Polynomial[] { f2 });
763         Console.WriteLine("_____________________RESULT_____________________");
764
765         Console.WriteLine("Remainder: " + div.Item2);
766         Console.WriteLine("Quotients:");
767
768         for (int i = 0; i < div.Item1.Length; i++)
769             Console.WriteLine(i.ToString() + ") " + div.Item1[i]);
770
771
772     }
773
774
775     //Division Ejemplo for a case in the previous Ejemplo
776     static void Ejemplo13()
777     {
778         Ring r = new Ring(Field.Real, new string[] { "x", "y", "z" });
779
780         r.FixOrder(new string[] { "x", "y", "z" });
781         r.Order = MonomialOrder.deglex;
782
783         Polynomial f1 = new Polynomial(r);
784         f1.AddTerm(1, new int[] { 0, 1, 0 });
785         f1.AddTerm(1, new int[] { 0, 0, 0 });
786
787
788         Polynomial f2 = new Polynomial(r);
789         f2.AddTerm(1, new int[] { 1, 0, 0 });
790         f2.AddTerm(1, new int[] { 1, 2, 0 });
```

```
791
792         Polynomial f3 = new Polynomial(r);
793         f3.AddTerm(1, new int[] { 0, 1, 1 });
794
795         Polynomial f4 = new Polynomial(r);
796         f4.AddTerm(1, new int[] { 0, 0, 1 });
797         f4.AddTerm(1, new int[] { 1, 0, 0 });
798
799
800
801         Polynomial dividend = new Polynomial(r);
802         dividend.AddTerm(1, new int[] { 3, 3, 1 });
803         dividend.AddTerm(1, new int[] { 3, 2, 1 });
804         dividend.AddTerm(1, new int[] { 2, 3, 1 });
805         dividend.AddTerm(1, new int[] { 2, 2, 1 });
806
807
808         var quotients = new Polynomial[] { f1, f2, f3, f4 };
809
810
811         var res = r.Divide(dividend, quotients);
812         Console.WriteLine("---------RESULTADO---------");
813         Console.WriteLine("resto:" + res.Item2);
814
815         Console.WriteLine("Sin embargo, al alterar el orden de la división
                a  { f2, f3, f1, f4 } obtenemos:");
816         res = r.Divide(dividend, new Polynomial[] { f2, f3, f1, f4 });
817         Console.WriteLine("resto:" + res.Item2);
818
819
820     }
821
822     static void Ejemplo14()
823     {
824         Ring r = new Ring(Field.Real, new string[] { "x", "y", "z" });
825
826         r.FixOrder(new string[] { "z", "y", "x" });
827         r.Order = MonomialOrder.deglex;
828
829         Polynomial f1 = new Polynomial(r);
830         f1.AddTerm(1, new int[] { 1, 1, 0 });
831         f1.AddTerm(-1, new int[] { 0, 1, 0 });
832
833         Polynomial f2 = new Polynomial(r);
834         f2.AddTerm(1, new int[] { 0, 2, 0 });
835         f2.AddTerm(-1, new int[] { 1, 0, 0 });
836
837
838
839
840         Polynomial dividend = new Polynomial(r);
841         dividend.AddTerm(-1, new int[] { 1, 0, 0 });
842         dividend.AddTerm(1, new int[] { 1, 2, 0 });
843
844
845         var res = r.Divide(dividend, new Polynomial[] { f1, f2 });
```

```
846         Console.WriteLine("------------------RESULTADO---------------");
847 
848         Console.WriteLine("resto:" + res.Item2);
849         Console.WriteLine("---------------------------------");
850 
851 
852     }
853     static void Ejemplo15()
854     {
855         Ring r = new Ring(Field.Real, new string[] { "x", "y", "z" });
856 
857         r.FixOrder(new string[] { "x", "y", "z" });
858 
859         Polynomial dividend = new Polynomial(r);
860         dividend.AddTerm(-1, new int[] { 0, 2, 2 });
861         dividend.AddTerm(-1, new int[] { 0, 2, 0 });
862 
863         Polynomial f1 = new Polynomial(r);
864         f1.AddTerm(1, new int[] { 1, 0, 1 });
865         f1.AddTerm(-1, new int[] { 0, 1, 0 });
866 
867         Polynomial f2 = new Polynomial(r);
868         f2.AddTerm(1, new int[] { 0, 1, 1 });
869         f2.AddTerm(1, new int[] { 1, 0, 0 });
870 
871 
872         Console.WriteLine("------------------RESULTADO---------------");
873         Console.WriteLine(r.Divide(dividend, new Polynomial[] { f1,f2
                }).Item2);
874 
875         Console.WriteLine("---------------------------------");
876 
877 
878     }
879 
880     static void Ejemplo16()
881     {
882         Ring r = new Ring(Field.Real, new string[] { "x", "y", "z" });
883         r.Order = MonomialOrder.deglex;
884 
885         r.FixOrder(new string[] { "z","y","x" });
886 
887 
888         Polynomial f1 = new Polynomial(r);
889         f1.AddTerm(1, new int[] { 1, 0, 1 });
890         f1.AddTerm(1, new int[] { 0, 1, 0 });
891 
892         Polynomial f2 = new Polynomial(r);
893         f2.AddTerm(-1, new int[] { 2, 1, 1 });
894         f2.AddTerm(1, new int[] { 0, 2, 0 });
895 
896 
897         Polynomial f3 = new Polynomial(r);
898         f3.AddTerm(1, new int[] { 1, 2, 0 });
899         f3.AddTerm(-1, new int[] { 0, 1, 0 });
900 
```

```
901         Ideal I = new Ideal(new Polynomial[] { f1, f2}, r);
902 
903         var gb = I.ReducedGrobnerBasis();
904 
905         Console.WriteLine("__________Gröbner Basis_____________________");
906         foreach (var p in gb)
907             Console.WriteLine(p);
908 
909         Console.WriteLine("---------------------------------");
910         var lex = new DegreeLexicographical();
911         Console.WriteLine(gb[1].LeadingTerm);
912         Console.WriteLine(lex.Compare(gb[1].LeadingTerm, gb[1].Terms[1]));
913 
914     }
915 
916 
917     static void Ejemplo17()
918     {
919         Ring r = new Ring(Field.Real, new string[] { "x", "y", "z" })
920         {
921             Order = MonomialOrder.deglex
922         };
923 
924         r.FixOrder(new string[] { "z", "y", "x" });
925 
926 
927         Polynomial f1 = new Polynomial(r);
928         f1.AddTerm(1, new int[] { 0, 1, 0 });
929         f1.AddTerm(1, new int[] { 1, 0, 0 });
930 
931         Polynomial f2 = new Polynomial(r);
932         f2.AddTerm(-1, new int[] { 1, 1, 1 });
933         f2.AddTerm(1, new int[] { 0, 1, 2 });
934 
935 
936         Polynomial f3 = new Polynomial(r);
937         f3.AddTerm(1, new int[] { 1, 2, 0 });
938         f3.AddTerm(-1, new int[] { 0, 1, 0 });
939 
940         Ideal I = new Ideal(new Polynomial[] { f1, f2, f3}, r);
941 
942         Console.WriteLine("__________Conjunto
               generador_____________________");
943         foreach (var p in I.GeneratorSet)
944             Console.WriteLine(p);
945 
946         Console.WriteLine("__________Gröbner Basis_____________________");
947         var gb = I.ReducedGrobnerBasis();
948         foreach (var p in gb)
949             Console.WriteLine(p);
950 
951         Console.WriteLine("__________Membership_____________________");
952         //2xy^2 - 2z^2x - 2y - 2xyz
953         Polynomial f = new Polynomial(r);
954         f.AddTerm(2, new int[] { 1, 2, 0 });
955         f.AddTerm(-2, new int[] { 1, 1, 1 });
```

```
956             f.AddTerm(-2, new int[] { 0, 1, 0 });
957             f.AddTerm(-2, new int[] { 1, 0, 2 });
958 
959             Console.WriteLine(f);
960             Console.WriteLine(I.Member(f));
961 
962         }
```

Este documento esta firmado por

Firmante	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
Fecha/Hora	Sun Jul 05 23:48:26 CEST 2020
Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
Numero de Serie	630
Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)