Bases de Gröbner en Teoría de Códigos

Trabajo Fin de Máster

Autor:
Javier Suárez Quero

Tutor:
José Escoriza López

Máster en Matemáticas

Julio 2019
Universidad de Almería

Índice general

Abstract in English

The strong relationship that exists between Mathematics and Computer Science is undeniable. They are so mutually influenced that some mathematical branches were developed only to solve computing problems.

This text will study two mathematical branches specially related to informatics: Gröbner Basis, a tool discovered independently of computers, that turned out to be extremely useful in this area; and Coding Theory, a branch of Mathematics which appeared in the computers age to improve digital communication.

The purpose of this work is to introduce these theories as instructive as possible in order to be read and understood by those knowing of this topic for the first time.

Chapter 2 will study the origin and description of Gröbner Basis, paying special attention to the algorithms to compute them.

In parallel, chapter 3 will study Coding Theory, with a few examples that will clarify lineal, cyclic and BCH-codes.

Both theories will merge in chapter 4 as an algorithm to decode BCH-codes with Gröbner Bases is carefully studied.

Resumen en español

Es innegable la estrecha relación que existe entre las matemáticas y la informática. Están tan influenciadas mutuamente que incluso han surgido nuevas teorías en matemáticas a partir de problemas esencialmente informáticos.

En este texto se van a estudiar dos ramas de las matemáticas especialmente relacionadas con la informática: las Bases de Gröbner, una herramienta que se descubrió de forma independiente a la computación pero que se ha encontrado especialmente útil en procesos informáticos; y la Teoría de Códigos, una teoría que ha surgido en la era digital para mejorar las comunicaciones a distancia.

El objetivo de este trabajo es introducir estas dos teorías de forma especialmente didáctica para hacerlo accesible a iniciados en estas materias.

En el capítulo 2 se estudiarán las Bases de Gröbner, haciendo especial incapié en los algoritmos necesarios para calcularlas.

De forma paralela, en el capítulo 3 se verá la Teoría de Códigos, con varios ejemplos concretos que permitirán entender los códigos lineales, cíclicos y BCH.

En un cuarto capítulo se combinarán ambas teorías, viendo con detalle un algoritmo de decodificación de códigos BCH con Bases de Gröbner.

1 Introducción

Desde los primeros años en los que empieza a formarse un matemático es fácil advertir la estrecha relación entre las matemáticas y la informática. Las matemáticas no solo se han servido de la computación para hacer cálculos impracticables a mano, como el cálculo de los primos grandes; sino incluso para demostrar algunos resultados de los que no se conoce demostración teórica, como el *Teorema de los Cuatro Colores*.

De igual modo, la informática bebe mucho de las matemáticas sin más que ver que el lenguaje más básico que reconoce un ordenador es el binario y hasta el punto de que se han creado nuevas ramas de las matemáticas para solventar problemas informáticos, como la Criptografía; y se han potenciado mucho otras áreas por su aplicación computacional, como la Teoría de Aproximación.

Esta simbiosis ha provocado que hasta el más novato de los estudiantes de matemáticas oiga hablar de ciertos resultados y algoritmos que son útiles *por ser fáciles de implementar* o por *optimizar el cálculo computacional*. Sin embargo, a la hora de la verdad, estos algoritmos no siempre parecen tan sencillos, al menos para el aprendiz, ni se estudian expresamente.

Es por este motivo que la intención de este texto es mostrar que se pueden implementar estos algoritmos sin más conocimientos que los de un estudiante de matemáticas. Otro de los objetivos es explicar de la forma más didáctica posible dos ramas de las matemáticas que, sin necesitar de una base muy extensa, suelen ser desconocidas para la mayoría de iniciados. Para dar coherencia e interés al texto, estas dos áreas se unen al final dando lugar a una aplicación de las matemáticas a la informática.

Volviendo a la relación simbiótica entre estas dos disciplinas, vamos a presentar las dos ramas que vamos a estudiar en este texto por su relación con la informática:

En primer lugar, una teoría puramente algebraica que se desarrolló sin pensar en la informática, que cayó en desuso por la dificultad que entrañaban sus cálculos, pero que con el paso de los años ha resultado ser una herramienta extremadamente útil en muchos contextos computacionales: las **Bases de Gröbner**. Dedicaremos el segundo capítulo a estudiar qué son, de dónde vienen y cómo se calculan, incluyendo un algoritmo explícito programado con el software *Mathematica*. Al final del capítulo veremos una aplicación de las Bases de Gröbner que nos será de utilidad en el último capítulo: los **Ideales de Eliminación**.

Por otro lado, en el tercer capítulo estudiaremos una teoría que surge como respuesta a un problema de la informática, concretamente de la necesidad de subsanar errores producidos en la comunicación a través de medios digitales. Veremos qué es la **Teoría de Códigos**, cómo se utiliza un *código* y particularizaremos a los códigos lineales, cíclicos y por último a los **códigos BCH**.

Finalmente, en el cuarto capítulo haremos uso de los resultados vistos y uniremos ambas ramas estudiando un algoritmo con el que decodificar ciertos códigos BCH utilizando las Bases de Gröbner y los Ideales de Eliminación. Dicho algoritmo fue presentado por Xuemin Chen junto a tres compañeros del Instituto de Ingenieros Eléctricos y Electrónicos en el artículo *"Use of Gröbner Bases to decode Binary Cyclic Codes up to the True Minimum Distance"*[3].

2 Bases de Gröbner

Las bases de Gröbner fueron introducidas en 1965 por Bruno Buchberger [2] en el campo de los anillos de polinomios. La idea subyacente es la de generalizar el buen comportamiento de los ideales de polinomios en una variable.

Dado un cuerpo $\mathbb{K}$ y el anillo $\mathbb{K}[x]$, cualquier ideal I se puede generar a partir de un único elemento del anillo, el máximo común divisor de todos los elementos del anillo. Dado cualquier conjunto de generadores $\{f_1,\dots,f_s\} \subseteq \mathbb{K}[x]$ de I, se puede calcular usando el Algoritmo de Euclides un único polinomio $d = mcd(f_1,\dots,f_s)$ tal que $I =< f_1,\dots,f_s >=< d >$. Así, un polinomio cualquiera $f \in \mathbb{K}[x]$ estará en I si y solo si el resto de dividir f entre d es 0.

Esto ocurre porque $\mathbb{K}[x]$ es un **dominio de ideales principales** (**DIP**), no como $\mathbb{K}[x_1,\dots,x_n]$, lo que hace que el problema de la pertenencia a un ideal en el caso multivariable sea bastante más complicado. Sin embargo, no todo son malas noticias, el anillo $\mathbb{K}[x_1,\dots,x_n]$ es **noetheriano**, lo que significa que todos sus ideales son finitamente generados.

Las bases de Gröbner son una generalización de este máximo común divisor para un anillo de polinomios multivariable $\mathbb{K}[x_1,\dots,x_n]$ en el siguiente sentido: Una base de Gröbner de un ideal $I =< f_1,\dots,f_s >$ es un conjunto de polinomios $\{g_1,\dots,g_t\} \subset \mathbb{K}[x_1,\dots,x_n]$ tal que $I =< g_1,\dots,g_t >$ y que verifica que un polinomio $f \in \mathbb{K}[x_1,\dots,x_n]$ está en I si y solo si el resto al *dividir* f entre la base es 0. El concepto de esta *división* entre varios polinomios es uno de los pilares sobre los que se sustenta esta teoría.

2.1 Reducción de polinomios

Para empezar, hay que definir un orden monomial en $\mathbb{N}^n$, que nos permitirá señalar el *término líder* de un polinomio, $TL(f)$, identificando el monomio $x_1^{a_1}x_2^{a_2}\cdots x_n^{a_n} := x^\alpha$ con la n-upla $(a_1,a_2,\dots,a_n) := \alpha$.

Definición 2.1. Una relación de orden $>$ en $\mathbb{N}^n$ es un **orden monomial** si se verifica que:

1. Es un orden total: Para cualesquiera dos elementos distintos $\alpha,\beta \in \mathbb{N}^n$, tenemos que $\alpha > \beta$ o $\beta > \alpha$.

2. Es compatible con la suma: Para cualesquiera $\alpha,\beta,\gamma \in \mathbb{N}^n$, se verifica que

$$\alpha > \beta \quad \Rightarrow \quad \alpha+\gamma > \beta+\gamma$$

3. Es un buen orden: Todo subconjunto no vacío tiene mínimo y toda cadena descendente termina.

Ejemplo 2.2. *Los siguientes son algunos ejemplos de órdenes monomiales:*

- *Orden lexicográfico o alfabético (**lex**). Dados $\alpha,\beta \in \mathbb{N}^n$, tenemos que $\alpha >_{lex} \beta$ cuando $\alpha-\beta$ tiene el primer elemento no nulo por la izquierda positivo.*

- *Orden graduado lexicográfico (**grlex**). Dados $\alpha, \beta \in \mathbb{N}^n$, tenemos que $\alpha >_{grlex} \beta$ cuando $|\alpha| > |\beta|$, o bien cuando $|\alpha| = |\beta|$ y $\alpha >_{lex} \beta$* [1].
- *Orden graduado lexicográfico inverso (**grevlex**). Dados $\alpha, \beta \in \mathbb{N}^n$, tenemos que $\alpha >_{grevlex} \beta$ cuando $|\alpha| > |\beta|$, o bien cuando $|\alpha| = |\beta|$ y $\beta >_{lex} \alpha$.*

A partir del concepto de *orden monomial* definimos las siguientes nociones:

Definición 2.3. Dados un polinomio $f \in \mathbb{K}[x_1, \ldots, x_n]$ y un orden monomial $>$, llamaremos **grado del polinomio** a $gr(f) = \max\{\alpha \in \mathbb{N}^n : x^\alpha \text{ aparece en } f\}$. Al término correspondiente con el máximo exponente se le conoce como **término líder** (TL). Si tenemos en cuenta el coeficiente lo acompaña hablamos de **monomio líder** (ML). Dicho coeficiente es el **coeficiente líder** (CL). Cuando $CL(f) = 1$ decimos que f es un **polinomio mónico**.

Ejemplo 2.4. *Considerando el orden lexicográfico y el polinomio $f(x, y, z) = y^3 - 2xy + z$, tenemos que:*

$$\begin{aligned} ML(f) &= -2xy \\ gr(f) &= (1, 1, 0) \\ TL(f) &= xy \\ CL(f) &= -2 \end{aligned}$$

Una vez fijado un orden monomial, vamos a definir una herramienta que generaliza cada uno de los pasos de la división de polinomios de una variable. La principal diferencia es que queremos que nuestro algoritmo *divida* un polinomio entre un conjunto de polinomios.

Definición 2.5. Fijado un orden monomial, sean $f, g, p \in \mathbb{K}[x_1, \ldots, x_n]$ con $f, p \neq 0$ y sea P un subconjunto de $\mathbb{K}[x_1, \ldots, x_n]$. Decimos que:

1. f se **reduce** a g **módulo** p **eliminando** t (que notaremos $f \xrightarrow[p]{} g\ [t]$), si para algún término t de f existe otro término s de forma que $s \cdot TL(p) = t$. Y tenemos que
$$g = f - \frac{a}{CL(p)} \cdot s \cdot p$$
donde a es el coeficiente de t en f.
2. f se **reduce** a g **módulo** p (que notaremos $f \xrightarrow[p]{} g$) si $f \xrightarrow[p]{} g\ [t]$ para algún término t de f.
3. f se **reduce** a g **módulo** P (que notaremos $f \xrightarrow[P]{} g$) si $f \xrightarrow[p]{} g$ para algún $p \in P$.
4. f es **reducible módulo** p si existe algún $g \in \mathbb{K}[x_1, \ldots, x_n]$ tal que $f \xrightarrow[p]{} g$.

[1] Donde $|\cdot|$ es la 1-norma definida por $|\alpha| = |(a_1, \ldots, a_n)| = \sum_{i=1}^{n} a_i$.

5. f es **reducible módulo** P si existe algún $g \in \mathbb{K}[x_1,\dots,x_n]$ tal que $f \xrightarrow[P]{} g$.

Además, notaremos de igual forma a la extensión reflexiva y transitiva de esta relación. Es decir, diremos que $f \xrightarrow[P]{} f$ y que $f \xrightarrow[P]{} g$ cuando tengamos que $f \xrightarrow[P]{} h$ y $h \xrightarrow[P]{} g$.

Esencialmente, lo que hace esta **reducción** a un polinomio f es obtener, a partir de un *divisor* p o *divisores* P, un polinomio más pequeño g de forma similar al cálculo del resto en la división de polinomios en una variable:

1. En primer lugar se busca un término t de f que sea divisible por el término líder de p (o por el de algún $p \in P$). No es necesario que sea el término líder de f.

2. Análogamente al caso de una variable cuando se divide el término líder del dividendo entre el del divisor, se calcula el término s que verifica $s \cdot TL(p) = t$.

3. Se multiplica p por este término s y por la constante necesaria para que coincidan los monomios de los términos t y $s \cdot TL(p)$. Es decir, si la constante que acompaña a t es a, habrá que multiplicar por $\frac{a}{CL(p)}$.

4. Por último, se calcula g restando a f el polinomio que hemos calculado:

$$g = f - \frac{a}{CL(p)} \cdot s \cdot p$$

Con este proceso, se elimina de f un término sustituyéndolo por otros menores y sin tocar los términos más grandes que t, por lo que se obtiene un polinomio menor $g < f$.

A partir de este nuevo concepto, la reducción de un polinomio en módulo otro u otros, surgen los siguientes conceptos relacionados:

Definición 2.6. Fijado un orden monomial, sean $f, g, p \in \mathbb{K}[x_1,\dots,x_n]$ con $f, p \neq 0$ y sea P un subconjunto de $\mathbb{K}[x_1,\dots,x_n]$, diremos que:

1. f está en **forma normal módulo** p (**módulo** P) si **no** es reducible módulo p (módulo P).

2. La **forma normal módulo** p (**módulo** P) de f es un polinomio g que está en forma normal que verifica $f \xrightarrow[p]{} g$ ($f \xrightarrow[P]{} g$). Lo notaremos $f \xrightarrow[p]{*} g$ ($f \xrightarrow[P]{*} g$).

3. Diremos que $f \xrightarrow[p]{} g\ [t]$ es una **líder-reducción** si $t = TL(f)$.

4. Cuando exista una líder-reducción de f módulo p ($p \in P$), diremos que f es **líder-reducible módulo** p (**módulo** P).

5. Notaremos $f \xleftrightarrow[P]{} g$ cuando sea cierto $f \xrightarrow[P]{} g$ o bien $f \xleftarrow[P]{} g$.

6. Escribiremos $f \downarrow_P g$ cuando exista un $h \in \mathbb{K}[x_1,\dots,x_n]$ tal que $f \xrightarrow[P]{} h$ y $g \xrightarrow[P]{} h$.

Tras estas definiciones surge una pregunta natural: ¿Existe la forma normal de cualquier polinomio?

Teorema 2.7. Fijado un orden monomial, sean $P \subset \mathbb{K}[x_1,\dots,x_n]$ y $f \in \mathbb{K}[x_1,\dots,x_n]$, entonces existe un polinomio g normal módulo P tal que $f \xrightarrow[P]{*} g$.

Demostración:
Como ya hemos visto, cuando se reduce un polinomio f se obtiene un polinomio más pequeño, digamos g_1. Si pudiéramos volver a reducir g_1 obtendríamos un g_2 más pequeño y así sucesivamente. De esta forma obtendríamos una cadena descendente de polinomios

$$f \xrightarrow[P]{} g_1 \xrightarrow[P]{} g_2 \xrightarrow[P]{} \dots \xrightarrow[P]{} g_i \xrightarrow[P]{} \dots$$

con

$$f > g_1 > g_2 > \dots > g_i > \dots$$

Como estamos considerando un orden monomial, sabemos que toda cadena descendente termina, por lo que existe un último elemento g más pequeño que todos los demás.
Si g fuera reducible módulo P, existiría un elemento más pequeño en esta cadena, por lo que tenemos que g está en forma normal módulo P y $f \xrightarrow[P]{*} g$.

■

Estos conceptos, que pueden ser tediosos de calcular a mano, son sin embargo fáciles de programar. Con ayuda del software *Mathematica* hemos creado dos herramientas que determinan si un polinomio está en forma normal o la forma normal de un polinomio cualquiera:

`NormQ[f,P]` Dados un polinomio f y un conjunto de polinomios P, devuelve `True` o `False` según si f está en forma normal módulo P o no.

`RedPol[f,P]` Dados un polinomio f y un conjunto de polinomios P, devuelve la forma normal de f en módulo P, g, así como los polinomios Q tales que $g = f - \sum p_i q_i$.

Ahora vamos a ver algunas propiedades de la reducción de polinomios:

Lema 2.8. Fijado un orden monomial, sean $f,g,h,h_1 \in \mathbb{K}[x_1,\dots,x_n]$, P un subconjunto de $\mathbb{K}[x_1,\dots,x_n]$ y m un monomio, se verifica que:

1. Si $f \in P$, entonces $hf \xrightarrow[P]{*} 0$.
2. Si $f \xrightarrow[P]{} g$, entonces $mf \xrightarrow[P]{} mg$.
3. En particular, si $f \xrightarrow[P]{*} 0$, entonces $mf \xrightarrow[P]{*} 0$.
4. Si $f - g = h$ y $h \xrightarrow[P]{} h_1$, entonces existen $f_1, g_1 \in \mathbb{K}[x_1,\dots,x_n]$ tales que $f_1 - g_1 = h_1$, $f \xrightarrow[P]{} f_1$ y $g \xrightarrow[P]{} g_1$.
5. Si $f - g \xrightarrow[P]{*} 0$, entonces $f \downarrow_P g$ y en particular $f \xleftrightarrow[P]{} g$.

```
In[1]:= NormQ[f_, P_] := Module[{TLP, Mf, i, j},
                         módulo
         If[f == 0, Return[True]];
         si         retorna  verdadero
         TLP = MonomialList[P][[All, 1]];  (* Términos Líder de P *)
               lista de monomios   todo
         Mf = MonomialList[f];             (* Monomios de f *)
              lista de monomios
         For[i = 1, i ≤ Length[Mf], i++,
         para cada       longitud
          For[j = 1, j ≤ Length[TLP], j++,
          para cada       longitud
            If[NumberQ[Denominator[Mf[[i]] / TLP[[j]]]],
            si  ¿número?  denominador
              factor = {j, Mf[[i]] / TLP[[j]]};
              Return[False] ];
              retorna  falso
           ];
         ];
         Return[True]];
         retorna  verdadero
```

Figura 2.1: Código fuente de la función `NormQ[f,P]`.

```
In[2]:= RedPol[f_, P_] := Module[{qP, g},
                          módulo
          qP = ConstantArray[0, Length[P]];
               arreglo constante   longitud
          g = f;
          While[! NormQ[g, P],
          mientras
           g = Expand[g - factor[[2]] * P[[factor[[1]]]]];
               expande factores
           qP[[factor[[1]]]] += factor[[2]];
          ];
          Return[{g, qP}]];
          retorna
```

Figura 2.2: Código fuente de la función `RedPol[f,P]`.

A continuación vamos a relacionar esta reducción de polinomios con la *relación de congruencia* en el anillo inducida por ideales.

Definición 2.9. Sea P un subconjunto de $\mathbb{K}[x_1,\dots,x_n]$, notaremos por $\mathrm{Id}(P)$ al ideal generado por P. Es decir, al conjunto de todas las combinaciones lineales finitas $\sum h_i p_i$ con $h_i \in \mathbb{K}[x_1,\dots,x_n]$ y $p_i \in P$.

Definición 2.10. Sea I un anillo de $\mathbb{K}[x_1,\dots,x_n]$, llamaremos **relación de congruencia** modulo I a la relación $\equiv_I$ definida por:

$$f \equiv_I g \iff f - g \in I$$

Teorema 2.11. Sean $P \subseteq \mathbb{K}[x_1,\dots,x_n]$ y $f,g \in \mathbb{K}[x_1,\dots,x_n]$. Entonces $f \equiv_{\mathrm{Id}(P)} g \iff f \xleftrightarrow[P]{} g$.

Demostración:

$\Leftarrow|$ *Dados f y g tales que $f \xrightarrow[P]{} g$, por cada paso que haya que hacer en la reducción de f a g habrá un monomio m_i tal que al final obtenemos la igualdad*

$$f - \sum_i m_i p_i = g$$

para ciertos $p_i \in P$.

Por tanto se verifica que

$$f - g = \sum_i m_i p_i \in Id(P)$$

$\Rightarrow|$ *Partimos de dos elementos f y g tales que $f - g \in Id(P)$, es decir,*

$$f - g = \sum_i h_i p_i$$

para ciertos polinomios $h_i \in \mathbb{K}[x_1,\dots,x_n]$ y $p_i \in P$.

Por tanto tenemos que $f = g + \sum_i h_i p_i$ y es evidente que

$$f = g + \sum_i h_i p_i \xrightarrow[P]{} g$$

eliminando los términos que involucran los h_i.

■

Antes de pasar a otras propiedades de la reducción, presentamos la definición de conjunto *reducido* de polinomios y el programa implementado para calcular un conjunto reducido de polinomios que genere el mismo ideal que el conjunto dado.

Definición 2.12. Diremos que un conjunto de polinomios $P \subseteq \mathbb{K}[x_1,\dots,x_n]$ es **mónico** si todos sus elementos lo son.

$$P \text{ mónico} \iff \mathrm{CL}(p) = 1 \ \forall p \in P$$

Definición 2.13. Diremos que un conjunto de polinomios $P \subseteq \mathbb{K}[x_1,\dots,x_n]$ es **reducido** si todo $p \in P$ es mónico y está en forma normal módulo $P \backslash \{p\}$.

Para conseguir implementar un programa que calcule un conjunto reducido han hecho falta varias funciones auxiliares que listamos a continuación[2]:

`CoefLider[f]` Dado un polinomio f, devuelve el coeficiente líder.

`MonicQ[P]` Dado un conjunto de polinomios P, devuelve `True` o `False` según si P es mónico o no.

`Monico[P]` Dado un conjunto de polinomios P, lo devuelve transformado de forma que sea mónico.

`ReducQ[P]` Dado un conjunto de polinomios P, devuelve `True` o `False` según si P es reducido o no.

```
In[6]:= ReducQ[P_] := Module[{Q, Qsinp, p, i},
                      módulo
  Q = P;
  If[! MonicQ[Q], controlmonico = False;
  si                                falso
   Return[False], controlmonico = True];
   retorna  falso                verdadero
  For[i = 1, i ≤ Length[P], i++,
  para cada        longitud
   p = Q[[i]];
   Qsinp = Drop[Q, {i}];
           elimina
   If[! NormQ[p, Qsinp],
   si
    reducible = i;
    Return[False];
    retorna  falso
   ];
  ];
  Return[True]
  retorna  verdadero
 ]
```

Figura 2.3: Código fuente de la función `ReducQ[P]`.

`Reduccion[P]` Dado un conjunto de polinomios P, devuelve otro conjunto que genera el mismo ideal y que además es reducido.

[2]El código fuente de las funciones no incluido aquí se puede consultar en el archivo anexo.

```
In[7]:= Reduccion[P_] := Module[{Q, Qsinp, p, h, contador},
                         módulo
        Q = P;
        While[! ReducQ[Q],
        mientras
         If[! controlmonico, Q = Monico[Q]; Continue[]];
         si                                continúa iteración
         p = Q[[reducible]];
         Qsinp = Drop[Q, {reducible}];
                 elimina
         h = RedPol[p, Qsinp][[1]];
         If[! TrueQ[h == 0], Q = Append[Qsinp, h];, Q = Qsinp;];
         si    ¿verdadero?          añade
        ];
        Return[Q]
        retorna
       ]
```

Figura 2.4: Código fuente de la función `Reduccion[P]`.

2.2 ¿Qué es una base de Gröbner?

En la reducción de polinomios, un aspecto que aún no hemos estudiado es la unicidad. El algoritmo por el que se calcula la forma normal de un polinomio f respecto de un conjunto de polinomios P no dice nada sobre el orden en el que hay que reducir los monomios de f, ni sobre qué polinomio de P utilizar primero. ¿Se obtiene la misma forma normal independientemente de cómo utilicemos el algoritmo? La respuesta es que en general no, como podemos ver en el ejemplo 2.14. El objetivo de este capítulo es presentar y aprender a calcular las *Bases de Gröbner*, que son conjuntos de polinomios para los que la respuesta es afirmativa.

Ejemplo 2.14. *Consideremos el anillo de polinomios multivariable* $\mathbb{Q}[x,y,z]$ *y sean:*

$$\begin{aligned} f &= xyz - xy^2 + z \\ P &= \{p_1, p_2\} \\ p_1 &= xy + 1 \\ p_2 &= yz + 1 \end{aligned}$$

Con ayuda del algoritmo `RedPol` *calculamos la forma normal de* f *respecto de* P *de dos formas distintas, como se muestra en la figura 2.5, y podemos ver que según el orden en el que consideramos los polinomios de* P *obtenemos una forma normal u otra.*

Para empezar a estudiar esta unicidad, vamos a ver dos resultados con algunos casos en los que la forma normal es única.

Proposición 2.15. Sea $0 \neq p \in \mathbb{K}[x_1,\ldots,x_n]$, la formal normal calculada por $\xrightarrow[p]{}$ es única.

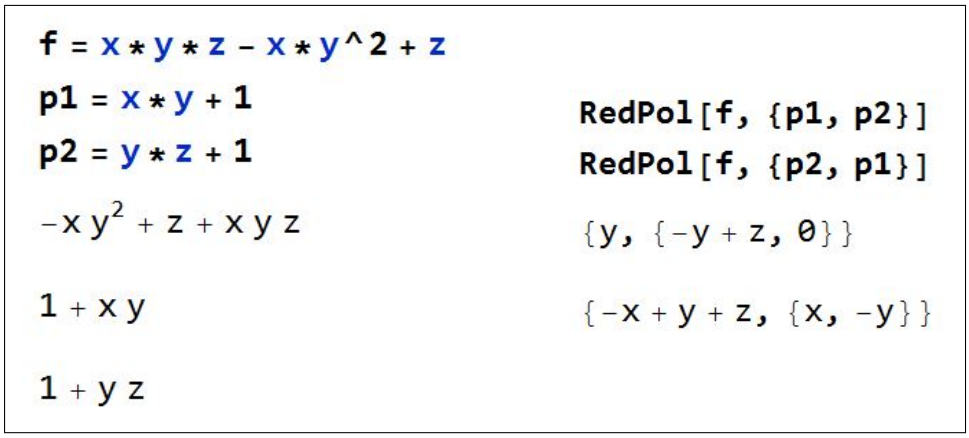

Figura 2.5: Ejemplo de cálculo de forma normal no única.

Corolario 2.16. Sea $P \subseteq \mathbb{K}[x_1,\ldots,x_n]$ un subconjunto no vacío tal que $\mathrm{Id}(P) = \mathrm{Id}(p)$ para algún $0 \neq p \in P$, entonces la formal normal calculada por $\xrightarrow[P]{}$ es única.

Además, para aquellos conjuntos de polinomios cuyo ideal se pueda generar con un solo polinomio, pero este no esté incluido, se puede transformar el conjunto añadiendo ese elemento de forma que genera el mismo ideal pero devuelve una forma normal única.

Sin embargo, no hay ningún resultado que nos garantice esto para aquellos ideales que no tienen un único generador. Es más, estos generadores no tendrían por qué ser únicos. La pregunta es natural: ¿Se puede transformar un conjunto de polinomios cualquiera en otro que genere el mismo ideal pero tenga asociadas formas normales únicas? La respuesta es que sí, y vamos a ver cómo.

Definición 2.17. Sea $P \subseteq \mathbb{K}[x_1,\ldots,x_n]$, entonces notaremos el conjunto de todos los términos líder de P como:

$$TL(P) = \{TL(p) \ : \ 0 \neq p \in P\}$$

Teorema 2.18. Dado $G \subseteq \mathbb{K}[x_1,\ldots,x_n]$, las siguientes afirmaciones son equivalentes:

(I) La formal normal calculada por $\xrightarrow[G]{}$ es única.

(II) $f \xrightarrow[G]{*} 0$ para todo $f \in \mathrm{Id}(G)$.

(III) Todo $0 \neq f \in \mathrm{Id}(G)$ es reducible módulo G.

(IV) Todo $0 \neq f \in \mathrm{Id}(G)$ es líder-reducible módulo G.

(V) Para todo $s \in TL(\mathrm{Id}(G))$ existe un $t \in TL(G)$ tal que $t|s$.

(VI) Los polinomios $h \in \mathbb{K}[x_1,\ldots,x_n]$ que están en forma normal respecto de G forman un sistema único de representantes de la siguiente partición de $\mathbb{K}[x_1,\ldots,x_n]$:

$$\{\, f + \mathrm{Id}(G) : f \in \mathbb{K}[x_1,\ldots,x_n] \,\}$$

Demostración:

(I) $\Rightarrow$ (II)| *Sea $f \in Id(G)$, tenemos que $f - 0 \in Id(G)$ y utilizando el lema 2.8 y el teorema 2.11 tenemos que $f \downarrow_G 0$, por lo que existe un h tal que $f \xrightarrow[G]{} h$ y $0 \xrightarrow[G]{} h$. Como 0 siempre está en forma normal y esta es única, vemos que $h = 0$ y por tanto $f \xrightarrow[G]{} 0$.*

(II) $\Rightarrow$ (III)| *Es evidente que si todo polinomio del ideal se reduce a 0, en particular es reducible.*

(III) $\Rightarrow$ (IV)| *Supongamos que f es el polinomio más pequeño que es reducible pero no líder-reducible. Entonces existe un h tal que $f \xrightarrow[G]{} h$ con $TL(f) = TL(h)$. Por la minimalidad de f, h es líder-reducible, digamos $h \xrightarrow[G]{} h_1$ para algún $g \in G$, pero entonces tenemos que $TL(g) \mid TL(h) = TL(f)$ por lo que f sería líder-reducible, una contradicción.*

(IV) y (V) son reformulaciones de la misma idea.

(V) $\Rightarrow$ (VI)| *Supongamos que existen dos polinomios, f_1 y f_2, distintos y en forma normal, tales que*

$$f_1 + Id(G) = f_2 + Id(G)$$

Entonces tenemos que $f_1 - f_2 \in Id(G)$ y por tanto existe un $g \in G$ de forma que $TL(g) \mid TL(f_1 - f_2)$. Pero el término líder de $f_1 - f_2$ es un término de f_1 o de f_2, por lo que alguno de los dos es reducible, una contradicción.

(VI) $\Rightarrow$ (I)| *Sean dos polinomios f_1 y f_2 tales que $f_1 \xleftrightarrow[G]{} f_2$ Por el teorema 2.11 tenemos que $f_1 - f_2 \in Id(G)$ y por tanto*

$$f_1 + Id(G) = f_2 + Id(G)$$

Ahora sean h_1 y h_2 las formas normales de f_1 y f_2 respectivamente, tenemos que

$$h_1, h_2 \in f_1 + Id(G) = f_2 + Id(G)$$

y por la hipótesis (VI) concluimos que $h_1 = h_2$.

■

Definición 2.19. Un subconjunto $G \subseteq \mathbb{K}[x_1, \ldots, x_n]$ se dice que es una **base de Gröbner** si es finito, no incluye el 0 y satisface las condiciones del teorema 2.18.
Dado un ideal I de $\mathbb{K}[x_1, \ldots, x_n]$, una **base de Gröbner de** I es una base de Gröbner G tal que $\mathrm{Id}(G) = I$.

Se puede demostrar que existen bases de Gröbner para cualquier ideal[3], pero estas no son únicas. De hecho, a una base de Gröbner se le podría añadir un polinomio cualquiera del ideal y obtendríamos una nueva base. Para llegar a esa unicidad hay

[3] Hay pruebas sencillas de esta existencia, pero no son constructivas y por eso no se han incluido en este trabajo.

que introducir las **bases de Gröbner reducidas** en el sentido de la definición 2.13. Éstas últimas siempre existen y además son únicas, pero todavía no sabemos cómo se pueden calcular. Ni siquiera sabemos cómo reconocerlas con un número finito de comprobaciones. Este es el objetivo de la siguiente sección.

Por último, cabe recordar que las bases de Gröbner dependen del orden monomial escogido en cada caso. Una base de Gröbner calculada con el orden lexicográfico, en general, no va a ser base con respecto a otros órdenes, aunque siempre generará el ideal. Las bases de Gröbner que lo son para cualquier orden monomial se denominan **bases de Gröbner universales**.

2.3 Cálculo de las bases de Gröbner

Vamos a volver al ejemplo 2.14. Evidentemente P no es una base de Gröbner, ya que hemos comprobado que no calcula la forma normal de forma única, pero vamos a ver otra prueba viendo que hay polinomios en $\mathrm{Id}(P)$ que no son reducibles.

Recordamos que

$$\mathrm{Id}(P) =< \{p_1, p_2\} >=< \{xy+1, yz+1\} >$$

y consideramos el polinomio

$$f = z \cdot p_1 - x \cdot p_2 = z - x \ \in \mathrm{Id}(P)$$

Es evidente que f está en el ideal y se puede comprobar fácilmente que no es reducible módulo P, por lo que P no puede ser base de Gröbner.

La forma en la que hemos encontrado el polinomio *problemático* ha sido calculando el mínimo común múltiplo de los monomios líder de dos polinomios (xyz), multiplicar cada polinomio por el monomio necesario para llegar a ese mínimo y restarlos para obtener un nuevo polinomio en el que no aparecerían esos términos líder.

No hay nada que garantice que los polinomios obtenidos de esta forma vayan a ser reducibles (como, de hecho, hemos comprobado que no ocurre). Lo interesante es que cuando todos los polinomios obtenidos de esta forma se reducen a 0, ocurre lo mismo con todos los polinomios del ideal y por tanto el conjunto de generadores del ideal es una base de Gröbner. En esta idea se basan los algoritmos con los que se calculan las bases de Gröbner.

El siguiente lema formaliza esta idea:

Lema 2.20. Dado un subconjunto finito $G \subseteq \mathbb{K}[x_1, \ldots, x_n]$ en el que $0 \notin G$, si se verifica que para todo par de elementos $g_1, g_2 \in G$ y todo par de monomios m_1, m_2 tales que $ML(m_1 g_1) = ML(m_2 g_2)$ tenemos que

$$m_1 g_1 - m_2 g_2 \xrightarrow[G]{} 0$$

entonces G es una base de Gröbner.

Ahora vamos a definir ese polinomio potencialmente *problemático* y una nueva caracterización de las bases de Gröbner.

Definición 2.21. Sean s y t dos términos

$$s = x_1^{k_1} \cdot \dots \cdot x_n^{k_n} \quad \text{y} \quad t = x_1^{l_1} \cdot \dots \cdot x_n^{l_n}$$

definimos el **mínimo común múltiplo** como

$$mcm(s,t) = x_1^{m_1} \cdot \dots \cdot x_n^{m_n} \quad \text{donde} \quad m_i = max(k_i, l_i) \quad \forall\ 1 \leq i \leq n$$

Definición 2.22. Sean g_1, g_2 dos polinomios no nulos de $\mathbb{K}[x_1, \dots, x_n]$; t_1, t_2 sus términos líderes; a_1, a_2 los coeficientes líderes y s_1, s_2 los términos que verifican que $s_1 t_1 = s_2 t_2 = mcm(t_1, t_2)$. Entonces el **S-polinomio** de g_1 y g_2 se define como

$$S(g_1, g_2) = a_2 s_1 g_1 - a_1 s_2 g_2$$

Teorema 2.23. Un subconjunto finito $G \subseteq \mathbb{K}[x_1, \dots, x_n]$ en el que $0 \notin G$ es una base de Gröbner si y solo si $S(g_1, g_2) \xrightarrow[G]{*} 0$ para cualesquiera $g_1, g_2 \in G$.

Demostración:

$\Rightarrow|$ *Es claro que $S(g_1, g_2)$ es un elemento del ideal generado por G y por tanto, se reduce a 0 módulo G.*

$\Leftarrow|$ *Por el lema 2.20, es suficiente probar que los polinomios de la forma $m_1 g_1 - m_2 g_2$ se reducen a 0, con $g_1, g_2 \in G$, $g_1 \neq g_2$ y m_1, m_2 monomios tales que*

$$ML(m_1 g_1) = ML(m_2 g_2) \tag{2.1}$$

Sean t_1 y t_2 los términos líderes de g_1 y g_2 respectivamente; a_1 y a_2 sus coeficientes líderes y descomponemos $m_1 = b_1 u_1$ y $m_2 = b_2 u_2$ de forma que $b_1, b_2 \in \mathbb{K}$ y u_1, u_2 son términos. Así, podemos ver la ecuación 2.1 como

$$b_1 a_1 u_1 t_1 = b_2 a_2 u_2 t_2 \tag{2.2}$$

Ahora sean s_1 y s_2 los términos tales que $s_1 t_1 = s_2 t_2 = mcm(t_1, t_2)$.
Podemos ver que $u_1 t_1 = u_2 t_2$ es un múltiplo común de t_1 y t_2, por lo que existe un término v tal que

$$u_1 t_1 = u_2 t_2 = v \cdot mcm(t_1, t_2) = v s_1 t_1 = v s_2 t_2 \quad \Rightarrow \quad u_1 = v s_1 \quad y \quad u_2 = v s_2$$

Además sabemos por 2.2 que $b_1 a_1 = b_2 a_2$, con lo que tenemos que

$$\begin{aligned} m_1 g_1 - m_2 g_2 &= b_1 u_1 g_1 - b_2 u_2 g_2 \\ &= b_1 v s_1 g_1 - b_2 v s_2 g_2 \\ &= \frac{b_1}{a_2} v \cdot (a_2 s_1 g_1 - a_1 s_2 g_2) \\ &= \frac{b_1}{a_2} v \cdot S(g_1, g_2) \end{aligned}$$

Como por hipótesis sabemos que $S(g_1, g_2) \xrightarrow[G]{*} 0$, utilizando el lema 2.8 tenemos que

$$m_1 g_1 - m_2 g_2 \xrightarrow[G]{*} 0$$

■

El siguiente corolario es una consecuencia directa de este resultado y del hecho de que el S-polinomio de dos monomios es 0.

Corolario 2.24. Todo subconjunto finito de monomios es una base de Gröbner.

Sin embargo, la consecuencia más importante de este teorema es el hecho de que nos proporciona una forma de comprobar si un subconjunto de polinomios es una base de Gröbner con un número finito de comprobaciones. Para cada par de polinomios del conjunto hay que calcular la forma normal del S-polinomio y comprobar si es 0 o no. Es decir, para un conjunto de n polinomios hay que hacer $\frac{n\cdot(n-1)}{2}$ comprobaciones (como máximo).

Además, basándose en esta misma idea, Bruno Buchberger [2] desarrolló un algoritmo por el que se puede transformar cualquier conjunto de polinomios en otro que genere el mismo ideal pero que sea una base de Gröbner. La idea es añadir al conjunto de polinomios la forma normal de los S-polinomios que no se reduzcan a 0. De esta forma el conjunto genera el mismo ideal –ya que los S-polinomios son elementos del ideal– y se solventa el problema de que los S-polinomios no se reduzcan a 0, ya que todos los elementos del conjunto se reducen a 0 de forma trivial.

La base obtenida por este algoritmo está lejos de ser óptima ya que seguramente tendrá polinomios redundantes, pero al transformarla en un conjunto reducido, se demuestra que sigue siendo base de Gröbner, con lo que se obtiene la **base de Gröbner reducida**, que sabemos que es única para cada ideal.

Como algunos conceptos anteriores, calcular una base de Gröbner con papel y lápiz es un proceso largo y tedioso, pero estos resultados nos permiten programar herramientas para calcularlas a ordenador. Estas son las funciones que hemos implementado:

`SPol[f,g]` Dados dos polinomios f y g, devuelve el S-polinomio $S(f,g)$.

```
In[8]:= SPol[f_, g_] := Module[{mcm, tl1, tl2, s1, s2},
                        módulo
        tl1 = MonomialList[f][[1]];
              lista de monomios
        tl2 = MonomialList[g][[1]];
              lista de monomios
        mcm = PolynomialLCM[tl1, tl2];
              mínimo común múltiplo polinómico
        (*Tanto el mcm como los s ya incluyen los coeficientes,
        por lo que no hay que calcularlos aparte *)
        s1 = mcm / tl1;
        s2 = mcm / tl2;
        Return[Expand[s1 * f - s2 * g]]
        retorna   expande factores
       ]
```

Figura 2.6: Código fuente de la función `SPol[f,g]`.

`GroebnerQ[G]` Dado un conjunto de polinomios G, devuelve `True` o `False` según si G es una base de Gröbner o no.

`BaseGroebner[P]` Dados un conjunto de polinomios P, devuelve una base de Gröbner que genera el ideal $\mathrm{Id}(P)$.

```
In[10]:= BaseGroebner[F_] := Module[{G, pares, i, j, actual, h},
                                módulo
  G = F;
  pares = {};
  For[i = 1, i ≤ Length[G], i++,
  para cada          longitud
   For[j = 1, j < i, j++,
   para cada
     AppendTo[pares, {G[[i]], G[[j]]}];
     añade al final
    ];
  ]; (* Bucle que genera todas las posibles combinaciones de
        dos elementos del conjunto*)
  While[Length[pares] ≥ 1, (* para cada par... *)
  mient··· longitud
   actual = pares[[1]];
   pares = Drop[pares, 1];
           elimina
   h = RedPol[Apply[SPol, actual], G][[1]];
               aplica
   (*Se calcula una forma normal del S-polinomio *)
   If[TrueQ[h == 0], Continue[],
   si  ¿verdadero?    continúa iteración
    (*Si h se reduce a 0, se pasa al siguiente par *)
    For[i = 1, i ≤ Length[G], i++,
    para cada          longitud
     AppendTo[pares, {G[[i]], h}];
     añade al final
    ]; (* Si no, se añade h a la base de Gröbner y habrá que evaluar
          todos los pares entre h y los elementos de G*)
    AppendTo[G, h];
    añade al final
   ];
  ];
  (*Cuando todos los pares han sido comprobados
     tenemos la base de Gröbner *)
  Return[G]
  retorna
 ]
```

Figura 2.7: Código fuente de la función `BaseGroebner[P]`.

2.4 Ideales de eliminación

Una de las aplicaciones por las que destacan las bases de Gröbner es la ayuda a la resolución de sistemas de ecuaciones no lineales.

Se sabe que las soluciones de un sistema de ecuaciones donde $f_1,\dots,f_k \in \mathbb{K}[x_1,\dots,x_n]$ están igualados a 0, son las mismas que en otro sistema cuyos polinomios generen el mismo ideal. En particular, el espacio de soluciones es el mismo si igualamos a 0 los polinomios de una base de Gröbner del ideal.

$$\left.\begin{array}{l} f_1 = 0 \\ f_2 = 0 \\ \vdots \\ f_k = 0 \end{array}\right\} = \left\{\begin{array}{l} g_1 = 0 \\ g_2 = 0 \\ \vdots \\ g_l = 0 \end{array}\right. \iff \mathrm{Id}(f_1,\dots,f_k) = \mathrm{Id}(g_1,\dots,g_l)$$

Pensemos en los sistemas de ecuaciones lineales: Una forma sencilla de resolverlos es utilizar el *método de Gauss* para transformarlo en un sistema escalonado como podemos ver en el ejemplo 2.25. Por desgracia este método solo funciona para ecuaciones lineales, pero en sistemas que no lo son también se puede utilizar una estrategia similar: buscar ecuaciones que no contengan todas las variables para simplificar el problema. Con esta idea en la cabeza, vamos a ver lo que son los *ideales de eliminación*.

Ejemplo 2.25. *Sistema de ecuaciones lineales transformado en un sistema escalonado por el método de Gauss.*

$$\left.\begin{array}{rcrcrl} x & - & 2y & + & z & = 4 \\ 2x & - & y & + & 4z & = 9 \\ -x & + & y & - & 2z & = -3 \end{array}\right\} \Rightarrow \left.\begin{array}{rcrcrl} x & - & 2y & + & z & = 4 \\ & & 3y & + & 2z & = 1 \\ & & & & z & = 2 \end{array}\right\}$$

Definición 2.26. Sea I un ideal I del anillo $\mathbb{K}[x_1,\dots,x_n]$ y sean $U = \{u_1,u_2,\dots,u_r\} \subseteq \{x_1,x_2,\dots,x_n\}$ algunas de las variables, de forma que $\mathbb{K}[U] = \mathbb{K}[u_1,\dots,u_r] \subseteq \mathbb{K}[x_1,\dots,x_n]$. Entonces llamaremos **ideal de eliminación de** I **respecto a** U al ideal $I \cap \mathbb{K}[U]$ y lo notaremos por I_U.

Es decir, el ideal de eliminación está formado por todos los elementos del ideal que solo están formados por las variables de U. En un sistema de ecuaciones no lineales sería muy útil encontrar una ecuación que solo utilizase una de las variables, al fin y al cabo es lo que permite el método de Gauss para el caso lineal. Pero, ¿se puede encontrar una base del ideal en la que aparezcan dichas ecuaciones?

La respuesta es que sí, calculando una base de Gröbner del ideal para el orden monomial adecuado. Este orden debe verificar que todos los polinomios de $\mathbb{K}[U]$ sean más pequeños que cualquier polinomio de $\mathbb{K}[x_1,\dots,x_n]\backslash\mathbb{K}[U]$. Esto es, cualquier polinomio que contenga una variable no incluida en U será mayor que todos los polinomios de $\mathbb{K}[U]$. Esto se puede conseguir fácilmente utilizando un orden lexicográfico en el que ordenemos las variables de forma que las u_i sean las más pequeñas.

Bajo estas condiciones se verifica la siguiente proposición:

Proposición 2.27. Sea I un ideal del anillo $\mathbb{K}[x_1,\dots,x_n]$ y sean algunas de las variables $U = \{u_1,u_2,\dots,u_r\} \subseteq \{x_1,x_2,\dots,x_n\}$ de forma que $\mathbb{K}[U] = \mathbb{K}[u_1,\dots,u_r] \subseteq \mathbb{K}[x_1,\dots,x_n]$. Además, consideramos un orden monomial en el que $s < t$ para todo $s \in \mathbb{K}[U]$ y $t \in \mathbb{K}[x_1,\dots,x_n]\backslash\mathbb{K}[U]$.
Entonces, dada una base de Gröbner G de I, $G \cap \mathbb{K}[U]$ es una base de Gröbner de I_U.

Esencialmente, lo que significa esta proposición es que, si calculamos una base de Gröbner con un orden lexicográfico, las ecuaciones que contengan solo a las variables *pequeñas* generan ese ideal de eliminación.

Volviendo a los sistemas de ecuaciones, una base del ideal de eliminación I_{u_i} nos permitiría saber los posibles valores de esta variable en el espacio de soluciones. Conocidas las posibles soluciones de u_i, el ideal de eliminación $I_{\{u_i,u_j\}}$ nos permitiría conocer las soluciones de u_j en función de las de u_i y así sucesivamente.

Si aplicásemos este resultado al sistema del ejemplo 2.25, vemos en la figura que el ideal de eliminación será $I_z =< z-2 >$.

```
G = BaseGroebner[{x - 2*y + z - 4, 2*x - y + 4*z - 9, -x + y - 2*z + 5}]
```

$$\left\{-4 + x - 2\,y + z,\ -9 + 2\,x - y + 4\,z,\ 5 - x + y - 2\,z,\ -1 + 3\,y + 2\,z,\ \frac{2}{3} - \frac{z}{3}\right\}$$

```
Reduccion[G] // Expand
                expande factores
```

$$\{-2 + z,\ x,\ 1 + y\}$$

Figura 2.8: Resolución del sistema del ejemplo 2.25 con bases de Gröbner.

Cabe señalar que todos los ejemplos y los programas implementados hasta ahora han sido respecto al orden que utiliza el software *Mathematica* por defecto, que es el orden lexicográfico con el orden natural de las variables $x > y > z$. Sin embargo, en algún momento puede ser útil calcular el ideal de eliminación I_x, por lo que es necesario cambiar el orden sobre el que trabajamos. Para solventar este problema, se ha programado una función con la que cambiar el orden utilizado en cualquier momento y así poder calcular cualquier ideal de eliminación.

`CambiarOrden[var, ord]` Redefine las funciones necesarias para que todas las herramientas utilicen el orden *ord* con las variables *var*.

`IdealEliminacion[F,var]` Dados un conjunto de polinomios F y un conjunto devuelve una base de Gröbner que genera el ideal Id(P).

```
In[14]:= IdealEliminacion[F_, var_] := Module[{G, Id = {}, i},
                                      módulo
    (* Es MUY IMPORTANTE que el orden utilizado haga que las variables
     var sean más pequeñas que las demás   *)
    G = BaseGroebner[F];
    For[i = 1, i ≤ Length[G], i++,
    para cada        longitud
     If[SubsetQ[var, Variables[G[[i]]]], AppendTo[Id, G[[i]]];];
      si  ¿subconjunto?  variables          añade al final
    ];
    Return[Id];
     retorna
   ];
```

Figura 2.9: Código fuente de la función `IdealEliminacion[F,var]`.

3 Teoría de códigos

En cualquier forma de comunicación existe un ruido que modifica el mensaje transmitido en mayor o menor medida, dificultando la comunicación. Los distintos acentos de cada región pueden afectar a la comunicación oral; una mala caligrafía es ruido en la escrita; el paso del tiempo puede alterar el color de un cuadro y desvirtuar el mensaje del artista; o una mala iluminación puede hacer que no salga bien una fotografía o una conversación en lenguaje de signos. Sin embargo, el cerebro humano es capaz en muchas ocasiones de separar el ruido y entender el mensaje original. El problema viene cuando queremos comunicarnos a través de un medio digital.

Supongamos que queremos enviar un mensaje de texto de un dispositivo a otro. El terminal emisor transformará el texto escrito a una cadena binaria de unos y ceros; la transmitirá en forma de impulsos eléctricos que el receptor interpretará como unos y ceros; y éste último la transformará de nuevo al texto original. Estas transformaciones son lo que se conoce como **codificación** y **descodificación**.

Ahora bien, este canal de comunicación también tiene ruido. Una mala conexión entre cables, interferencias o un receptor estropeado pueden hacer que alguno (o algunos) de los bits se reciba de forma diferente a como se emitió, lo que podría suponer un error muy significativo. Para evitar este problema surgen los **códigos correctores de errores**.

El ejemplo más sencillo es el *código de la repetición*:

Ejemplo 3.1. *Supongamos que queremos transmitir el mensaje* 1101. *En vez de enviar el mensaje tal cual, enviaremos cada bit repetido tres veces:* 111 111 000 111. *De esta forma, aunque el ruido modificase el mensaje y se recibiese la cadena* 110 111 010 111, *el receptor sabe que cada grupo de tres bits representa un único dato, por lo que leerá la cadena como* 1101, *eligiendo en cada grupo el valor que se repita dos o tres veces.*

Este código es muy poco eficaz, ya que obliga a enviar el triple de información en cada transmisión, además de que si hubiera dos fallos en el mismo grupo se recibiría el mensaje incorrecto.

La **teoría de códigos** se encarga de buscar e implementar distintos mecanismos para enviar la información de forma que se puedan corregir el mayor número de errores posible con la menor cantidad de información necesaria en cada transmisión.

3.1 Conceptos generales

El primer paso a la hora de construir un **código corrector de errores** es fijar el *alfabeto* con el que se va a llevar a cabo la comunicación. Esto es, un conjunto finito de símbolos $\mathcal{A}$. Lo más habitual es utilizar un cuerpo finito $\mathbb{F}_q$, donde q tiene que ser una potencia de un primo. Además, como el objetivo son las transmisiones digitales, lo más común es que sean **códigos binarios**, es decir, utilizaremos como alfabeto $\mathbb{F}_2$.

Visto esto, veamos qué es un código y en qué consiste la codificación.

Definición 3.2. Dado un alfabeto $\mathcal{A}$ y un natural n, un **código** es un subconjunto $\mathcal{C} \subset \mathcal{A}^n$. Los elementos de $\mathcal{C}$ serán **palabras** de longitud n.

La codificación es el proceso por el que se transforma el mensaje que queremos enviar en la cadena que contiene la información del mensaje y la necesaria para recuperarlo en caso de que llegue con errores:

1. El mensaje original m será una cadena finita de caracteres de $\mathcal{A}$ tan larga como queramos, pongamos de longitud l.

$$m = (x_1, x_2, \ldots, x_l) \in \mathcal{A}^l$$

2. Este mensaje habrá dividirlo en submensajes más pequeños, de longitud k, de forma que l sea un múltiplo de k[1]. Cada uno de estos submensajes se enviará de forma independiente.

$$m = (\ (x_1, \ldots, x_k)\ ,\ (x_{k+1}, \ldots, x_{2k})\ ,\ \ldots\ ,\ (x_{l-k+1}, \ldots, x_l)\ ) \in \mathcal{A}^k \times \mathcal{A}^k \times \cdots \times \mathcal{A}^k$$

3. Cada submensaje se codificará mediante una aplicación c que envía cada mensaje de $\mathcal{A}^k$ a una palabra del código $\mathcal{C}$.

$$\begin{array}{rccl} c: & \mathcal{A}^k & \to & \mathcal{A}^n \\ & (x_1, \ldots, x_k) & \to & c(x_1, \ldots, x_k) \in \mathcal{C} \subset \mathcal{A}^n \end{array}$$

4. Al final se enviarán l/k palabras del código $\mathcal{C}$, que durante la transmisión pueden sufrir alguna transformación, llegando al receptor como $c(x_1, \ldots, x_k) + e$, donde e es el error provocado por el ruido.

5. Los mensajes recibidos se descodificarán mediante una aplicación $d : \mathcal{A}^n \to \mathcal{A}^k$ que verifica

$$d(c(x_1, \ldots, x_k) + e) = (x_1, \ldots, x_k)$$

siempre y cuando el error e sea lo suficientemente pequeño. Al unir todos los submensajes descodificados se recupera el mensaje original.

A la hora de elegir entre un código u otro, son varios los aspectos que hay que tener en cuenta: En primer lugar, según el canal de transmisión, necesitaremos un código que sea capaz de corregir más o menos errores. Además, el objetivo es tener que enviar la menor cantidad de información posible para el mismo mensaje. Nótese que las palabras del código son de longitud n, con k símbolos de información y $n - k$ símbolos redundantes para corregir errores. Se denomina **tasa de transmisión** al cociente k/n. Por último, la complejidad de las aplicaciones c y d puede ser importante cuando el objetivo es automatizarlo por ordenador.

Cuando hablamos de los errores que puede corregir un código es importante distinguir entre *detectar* y *corregir*. Si utilizásemos como código el idioma español y recibiésemos en una transmisión la palabra `caxa`, seríamos capaces de *detectar* un error, pero

[1] Generalmente el parámetro que tenemos fijado es k y es poco probable que el mensaje que queramos enviar tenga una longitud múltiplo de k, pero basta con completar el mensaje original con caracteres que no tengan significado, como espacios o ceros, hasta el siguiente múltiplo de k más cercano.

no podríamos *corregirlo* por la cantidad de palabras del código que surgirían de cambiar una letra (`cana, casa, cama, caza, cara,...`). El código de repetición (ejemplo 3.1) es capaz de detectar dos errores y corregir uno, y si repitiéramos cada bit cuatro veces podría detectar tres errores pero igualmente solo corregir uno.[2]

Con esta idea, es claro que las palabras de un código tienen que ser suficientemente distintas entre sí para evitar confusiones. Las siguientes definiciones ayudan a cuantificar esta diferencia para aquellos códigos cuyo alfabeto es un cuerpo $\mathbb{F}_q$, en los que nos centraremos de aquí en adelante.

Definición 3.3. Dada una palabra $x \in \mathbb{F}_q^n$, se define el **peso de Hamming** de x, $w(x)$, como el número de coordenadas no nulas de x.

Definición 3.4. Dadas dos palabras $x, y \in \mathbb{F}_q^n$, se define la **distancia de Hamming** de x e y como

$$d(x,y) = w(x-y)$$

Se puede demostrar que es una distancia ya que verifica las propiedades de no negatividad, simetría, la desigualdad triangular y verifica que la distancia entre dos palabras es 0 si y solo si las palabras son iguales.

La idea es que la distancia entre dos palabras es el número de coordenadas distintas entre ellas y el número de errores que podrá corregir un código está íntimamente relacionado con la distancia que haya entre las palabras de dicho código.

Definición 3.5. Se llama **distancia mínima** de $\mathcal{C}$ a la distancia más pequeña que hay entre dos palabras del código.

$$d(\mathcal{C}) = min\{d(x,y) : x, y \in \mathcal{C}\}$$

Este concepto es importante por el hecho de que la mayoría de códigos descodifican por el método del *vecino más cercano*, es decir, recibida una palabra que no está en el código, asumiremos que el emisor quería enviar la palabra del código cuya distancia a la recibida es menor. De esta forma, con un código cuya distancia mínima sea 3, siempre se podrá corregir con éxito un fallo único en la transmisión, ya que sólo puede haber una palabra del código a distancia 1. El siguiente lema formaliza esta idea.

Lema 3.6. Dado un código $\mathcal{C}$ se verifica que:

I. $\mathcal{C}$ puede detectar hasta s errores si $d(\mathcal{C}) \geq s+1$.

II. $\mathcal{C}$ puede corregir hasta t errores si $d(\mathcal{C}) \geq 2t+1$.

Por último, antes de pasar a ver algunas familias de códigos, un apunte para simplificar la notación. A menudo, las n-uplas que forman las distintas palabras las notaremos simplemente yuxtaponiendo sus elementos:

$$x = (x_1, x_2, \ldots, x_n) \equiv x_1 x_2 \ldots x_n \in \mathcal{A}^n$$

[2]Existen algunos códigos cuyo único objetivo es detectar errores, sin pretensión de corregirlos, como es el caso de los DNI españoles. Esto es útil siempre y cuando se pueda volver a enviar la información en caso de ser incorrecta, como puede ocurrir si alguien comete una errata en su DNI.

3.2 Códigos lineales

Los códigos lineales son los más conocidos y utilizados en Teoría de Códigos. Son aquellos en los que el conjunto $\mathcal{C}$ es un subespacio vectorial.

Definición 3.7. Un $[n,k]$–**código lineal** q–**ario** es un subespacio vectorial $\mathcal{C} \leq \mathbb{F}_q^n$ de dimensión k.
Un $[n,k,d]$–código lineal es un $[n,k]$–código con distancia mínima d.

Una de las ventajas de los códigos lineales es que, al ser subespacios vectoriales, la suma de dos palabras del código es una nueva palabra del código, lo que permite el siguiente resultado que facilita el cálculo de la distancia mínima:

Teorema 3.8. Sea $\mathcal{C}$ un código lineal, la distancia mínima de $\mathcal{C}$ coincide con el peso más pequeño de una palabra no nula de $\mathcal{C}$.

$$d(\mathcal{C}) = w(\mathcal{C}) = min\{\, w(c) : 0 \neq c \in \mathcal{C} \,\}$$

Demostración:
Sean x e y dos palabras de $\mathcal{C}$ tales que $d(x,y) = d(\mathcal{C})$, tenemos que

$$d(\mathcal{C}) = d(x,y) = w(x-y) \geq w(\mathcal{C})$$

ya que $x-y$ es una palabra del código.

Por otro lado, sea z una palabra del código tal que $w(z) = w(\mathcal{C})$, se verifica que

$$w(\mathcal{C}) = w(z) = d(z,0) \geq d(\mathcal{C})$$

Es decir, tenemos que $d(\mathcal{C}) \geq w(\mathcal{C}) \geq d(\mathcal{C})$ y por tanto

$$d(\mathcal{C}) = w(\mathcal{C})$$

■

Así, para calcular la distancia mínima del código no es necesario calcular la distancia entre todos los posibles pares de palabras (para un código con M palabras, $\frac{M(M-1)}{2}$ comprobaciones), es suficiente con ver el peso de cada palabra no nula ($M-1$ comprobaciones).

Por otro lado, hemos dicho que un $[n,k]$–código lineal es un subespacio vectorial de dimensión k, lo que quiere decir que podemos encontrar una base de dicho espacio formada por k vectores. Esto es, todas las palabras del código se pueden conseguir con una combinación lineal de los k elementos de la base, eligiendo convenientemente los k coeficientes que acompañan a los vectores de la base.

Y viceversa. Cualquier k–upla de $\mathbb{F}_q$ podemos utilizarla como los coeficientes de los vectores de la base, obteniendo una palabra del código. Así es precisamente como vamos a codificar. Utilizando el alfabeto $\mathbb{F}_q$ dividiremos nuestro mensaje en submensajes de longitud k, que utilizaremos como coeficientes en una combinación lineal con los vectores de la base de $\mathcal{C}$. La palabra del código resultante será la que trasmitiremos.

Como conclusión tenemos una forma más intuitiva de entender el nombre de los códigos lineales. Un $[n,k]$–código lineal está compuesto por palabras de longitud n que contienen k símbolos de información y $n-k$ símbolos redundantes.

Además, a la hora de trabajar con los códigos lineales, es sencillo tanto codificar como comprobar si se han producido errores, gracias a los siguientes conceptos:

Definición 3.9. Una matriz $G \in \mathcal{M}_{k\times n}$ cuyas filas forman una base del código lineal $\mathcal{C}$ se denomina **matriz generatriz** de $\mathcal{C}$.

Definición 3.10. Llamaremos **matriz de paridad** o **matriz de control** de $\mathcal{C}$ a una matriz $H \in \mathcal{M}_{(n-k)\times n}$ que verifica

$$x \in \mathcal{C} \iff x \cdot H^T = 0$$

Así, dado un mensaje m de longitud k, se transmitirá la palabra $x = m \cdot G \in \mathcal{C}$. Al otro lado de la comunicación, se recibirá la palabra y que puede, o no, contener errores y para saber si es correcta, basta con comprobar si el producto $y \cdot H^T$ es 0. En caso de tener errores, veremos más adelante como corregirlos.

Directamente desde estas definiciones se deduce el siguiente resultado.

Teorema 3.11. Dado un código lineal $\mathcal{C}$ con matriz generatriz G y matriz de paridad H, se verifica

$$GH^T = 0$$

Demostración:
La demostración es trivial a partir de la definición de H. Como las filas de G son palabras del código, al multiplicarlas por H siempre darán 0.

■

Veamos ahora una forma de sencilla para las matrices generatriz y de paridad. Hemos visto que la matriz generatriz está formada por los elementos de una base del espacio vectorial $\mathcal{C}$, pero esta base no es única, por lo que para un mismo código $\mathcal{C}$ se pueden encontrar diferentes generatrices. Además, dada una matriz generatriz G se pueden obtener otras mediante algunas transformaciones que, o bien generan el mismo código, o uno equivalente:

- Permutar filas (o columnas).
- Multiplicar una fila (o columna) por un escalar no nulo.
- Sumar una fila o un múltiplo de una fila a otra.

Mediante estas transformaciones, dada una matriz generatriz cualquiera se puede transformar a *forma estándar*.

Definición 3.12. Dado un $[n,k]$–código lineal $\mathcal{C}$, diremos que una matriz generatriz G está en **forma estándar** si se puede ver como

$$G = [I_k|A]$$

donde I_k es la matriz identidad de orden k y A es una matriz $A \in \mathcal{M}_{k,(n-k)}$.

De forma similar podemos definir la forma estándar de la matriz de paridad, que además verifica la siguiente propiedad.

Lema 3.13. Dado un $[n,k]$–código lineal con matriz generatriz $G = [I_k|A]$, entonces la matriz $H = [-A^T|I_{n-k}]$ es una matriz de paridad de $\mathcal{C}$ y la llamaremos **matriz de paridad estándar**.

Estas matrices en su forma estándar simplifican mucho el proceso de codificación y descodificación ya que la palabra del código contendrá explícitamente el mensaje que se quiere enviar.

Dado un mensaje que queremos enviar por un canal no fiable $m = m_1 m_2 \dots m_k$, si a la hora de codificarlo utilizamos una matriz generatriz estándar G, obtendremos la palabra del código

$$x = m \cdot G = m \cdot [I_k | A] = m_1 m_2 \dots m_k u_1 \dots u_{n-k}$$

donde los símbolos u_i serán la información redundante que permitirá comprobar si durante la transmisión hay errores.

Es más, cuando la palabra enviada llegue al receptor y compruebe si tiene errores con la matriz de paridad, si no ha sufrido ninguna modificación, para recuperar el mensaje original bastará con quedarse con los primeros k símbolos.

Sin embargo, cuando con la matriz de paridad vemos que el vector recibido no es una palabra del código, es evidente que ha sufrido modificaciones durante el trayecto y hay que intentar descubrir qué palabra del código fue la que se envió. Lo más habitual y lo que haremos en este texto, como ya hemos mencionado, es utilizar el método del *vecino más próximo*, es decir, suponer que ha habido el menor número de errores posible y por tanto la palabra que se envió es la más cercana a la recibida.

Este método maximizará la probabilidad de acertar la decodificación siempre y cuando estemos utilizando un **canal simétrico**, lo que quiere decir que:

- Cada símbolo tiene la misma probabilidad de ser modificado durante la transmisión y dicha probabilidad es menor que 1/2.
- Si un símbolo se recibe con error, el resto de errores son igualmente probables.

Una vez que ya hemos decidido que vamos a decodificar según el vecino más próximo, el siguiente problema que surge es saber cuál es ese *vecino* sin necesidad de comprobar la distancia a todas las palabras del código. Una de las formas de encontrar esa palabra más cercana la propuso David Slepian [10] aprovechando la estructura de subespacio aditivo del código. Veámosla.

Definición 3.14. Sea $\mathcal{C}$ un $[n,k]$–código lineal, visto como subespacio vectorial de $\mathbb{F}_q^n$, y sea a un vector cualquiera de $\mathbb{F}_q^n$. Entonces llamamos **clase** al conjunto $a+\mathcal{C}$ definido por

$$a + \mathcal{C} = \{a + x : x \in \mathcal{C}\}$$

Veamos ahora algunas propiedades de estas clases con un teorema y un lema previo.

Lema 3.15. Sea $a+\mathcal{C}$ una clase de $\mathcal{C}$ y sea $b \in a+\mathcal{C}$, entonces

$$b + \mathcal{C} = a + \mathcal{C}$$

Teorema 3.16. Sea $\mathcal{C}$ un $[n,k]$–código lineal q–ario, entonces se verifican

i. Todo vector de $\mathbb{F}_q^n$ está en alguna clase.

ii. Toda clase tiene exactamente q^k vectores.

iii. Dos clases son o iguales o disjuntas.

Demostración:

I. | *Para cualquier vector* $a = a + 0 \in a + \mathcal{C}$.

II. | *Dada cualquier clase* $a + \mathcal{C}$ *se puede establecer una biyección*

$$\mathcal{C} \to a + \mathcal{C}$$

$$x \to a + x$$

por lo que

$$|a + \mathcal{C}| = |\mathcal{C}| = q^k$$

III. | *Supongamos que dos clases* $a+\mathcal{C}$ *y* $b+\mathcal{C}$ *tienen algún elemento en común, digamos* v. *Entonces tenemos que existen* $x, y \in \mathcal{C}$ *tales que*

$$v = a + x = b + y$$

$$a = b + (y - x) \in b + \mathcal{C}$$

Por el lema 3.15 tenemos que

$$a + \mathcal{C} = b + \mathcal{C}$$

■

Definición 3.17. Llamaremos **líder de la clase** al vector de menor peso de cada clase.

Corolario 3.18. Dado $\mathcal{C}$ un $[n,k]$–código lineal q–ario, por el teorema 3.16 tenemos que $\mathbb{F}_q^n$ tiene una partición en $s = q^{n-k}$ clases disjuntas

$$\mathbb{F}_q^n = \mathcal{C} \cup (a_1 + \mathcal{C}) \cup \cdots \cup (a_{s-1} + \mathcal{C})$$

donde los elementos $\{0, a_1, a_2, \ldots, a_{s-1}\}$ son los líderes de cada clase.

Esta partición nos servirá para saber qué error es el que se ha cometido durante la transmisión. El líder de cada clase será el error cometido siempre que se reciba un vector de esa clase.

Supongamos que tenemos un código $\mathcal{C}$ con distancia mínima 3. Es evidente que ningún de peso 1 estarán en $\mathcal{C}$. Es más, sabemos que estarán en clases distintas, ya que la distancia entre dos palabras de peso 1, por la desigualdad triangular es como máximo 2:

$$d(x,y) \leq d(x,0) + d(y,0) = w(x) + w(y) = 2$$

Por esto, cualquier vector que tenga un solo error, será miembro de la clase $a_i + \mathcal{C}$, donde a_i tiene peso 1 y por tanto es el líder de la clase. Al ver en qué clase está la palabra recibida, sabremos exactamente cuál ha sido el error cometido y por tanto cuál es la palabra del código que se envió.

Ahora viene el siguiente problema, ¿cómo sabemos a que clase pertenece cada vector? Es demasiado costoso tener que conocer expresamente a qué clase pertenecen todos los vectores de $\mathbb{F}_q^n$. Para esto nos servirá el siguiente concepto.

Definición 3.19. Dado un $[n,k]$–código $\mathcal{C}$ con matriz de paridad H, definimos el **síndrome** de un vector $y \in \mathbb{F}_q^n$ como

$$S(y) = yH^T$$

Es fácil de ver, por las dimensiones de la matriz de paridad, que el síndrome de un vector tiene $n-k$ coordenadas. Además, por la definición de la matriz de paridad, cuando el síndrome de un vector es 0 es porque el vector es una palabra del código.

$$S(y) = yH^T = 0 \Rightarrow y \in \mathcal{C}$$

Ahora vamos a ver una propiedad del síndrome especialmente importante para nuestro objetivo.

Teorema 3.20. Dos vectores $u, v \in \mathbb{F}_q^n$ están en la misma clase si y solo si tienen el mismo síndrome.

Demostración:
Partimos de dos vectores $u, v \in \mathbb{F}_q^n$ que estén en la misma clase y veremos mediante equivalencias que tienen el mismo síndrome:

$$\begin{aligned} u + \mathcal{C} = v + \mathcal{C} &\iff u - v \in \mathcal{C} \\ &\iff (u-v)H^T = 0 \\ &\iff uH^T = vH^T \\ &\iff S(u) = S(v) \end{aligned}$$

■

Este resultado nos permite saber a qué clase pertenece cada palabra sin tener que conocer explícitamente todas. Basta con saber qué síndrome tiene cada clase (o cada líder de clase) y calcular el síndrome de la palabra en cuestión.

De esta forma, cuando se recibe un mensaje codificado con un código lineal, basta con multiplicar el vector recibido por la matriz de paridad: Si el producto (síndrome) es 0 la palabra recibida está en el código y por tanto no ha habido errores en la transmisión; si por el contrario el síndrome no es nulo, basta con saber qué clase es la que se asocia a ese síndrome y sabremos que el error que se ha producido es el líder de la clase. Para esto, lo habitual cuando se acuerda un código lineal entre emisor y receptor es construir una tabla con dos columnas: en una estarán todos los líderes de clases, que son los posibles errores, y en la otra columna veremos el síndrome asociado a cada líder. Esta tabla se conoce como la **tabla Síndrome-Líder**.

Hemos visto, tras definir los líderes de clase, que todos los vectores de peso mínimo[3] son los líderes de su propia clase y, por tanto, cada vez que se recibe una palabra de esa clase, se sabe que ese es el error cometido. Sin embargo, puede ocurrir que haya clases que no tengan ninguna palabra de peso mínimo. Cuando se recibe un vector de una de estas clases, sabremos que se han cometido más errores de los que es capaz de corregir el código, ya que habrá varias palabras del código a la misma distancia. En estos casos en los que sabemos que ha habido errores pero no somos capaces de

[3] Llamamos así a aquellos vectores cuyo peso es menor o igual que los errores que puede corregir el código. Si la distancia mínima es d, se pueden corregir hasta $t = \lfloor \frac{d-1}{2} \rfloor$ errores, y los vectores de peso mínimo serán aquellos de peso menor o igual que t.

corregirlos, lo habitual es solicitar la retransmisión del mensaje. Esto es conocido como **decodificación incompleta**.

Por otro lado, es necesaria una aclaración sobre el lenguaje en esta sección. Durante todo el texto hemos hablado, y hablaremos, de que un código puede corregir cierto número de errores en la transmisión. Cuando se estudia la estrategia de decodificación es habitual hablar en términos absolutos y decir que se encuentra la palabra que se envió. Evidentemente, esto ocurre siempre y cuando en la transmisión haya habido tantos errores como es capaz de corregir el código o menos. Es posible utilizar un código con capacidad para corregir dos errores y que en la transmisión haya cuatro símbolos modificados, por lo que se descodificará de forma errónea. Esto puede ocurrir en cualquier sistema y con cualquier canal, pero el objetivo es minimizar la probabilidad de que esto ocurra. Conociendo el canal se puede saber qué probabilidad existe de error en cada símbolo y, en consecuencia, elegir un código que con una probabilidad suficientemente alta sea capaz de corregir estos errores. El cálculo de esta probabilidad queda fuera de los objetivos de este texto, pero se puede consultar en libros de la bibliografía como el de Raymond Hill [4, Capítulo 6] o el de F. J. MacWilliams y N. J. A. Sloane [7, Capítulo 1.5].

Por último, vamos a ver un ejemplo que ilustre los conceptos vistos en esta sección con una de las familias más conocidas: los códigos de Hamming.[4]

Los códigos de Hamming son una familia de códigos capaces de corregir un error; se pueden definir sobre cualquier cuerpo, aunque nosotros veremos un caso binario; son sencillos de implementar para codificar y decodificar y se definen a partir de su matriz de paridad de la siguiente forma:

Definición 3.21. Sea r un entero positivo y H una matriz $r \times 2^r - 1$ cuyas columnas son todos los vectores diferentes de $\mathbb{F}_2^r$ en cualquier orden. Al código que tenga a H como matriz de paridad se le conoce como **código de Hamming binario** $Ham(r, 2)$.

Se puede demostrar que el código $Ham(r, 2)$ es un $[2^r - 1, 2^r - r - 1, 3]$–código lineal.

Ejemplo 3.22. *En este ejemplo vamos a estudiar el* $[7, 4, 3]$*–código lineal binario* $Ham(3, 2)$*:*

Solo por ser un $[7, 4, 3]$*–código lineal binario, ya sabemos que los mensajes serán palabras de longitud 4 de unos y ceros, que al codificarlos serán 7–uplas. Además, las palabras del código se diferencian entre sí en al menos 3 símbolos, por lo que podremos corregir con éxito un error.*

Ahora bien, por la definición de los códigos de Hamming, su matriz de paridad es cualquiera que tenga como columnas todos los vectores no nulos posibles. Por ejemplo, esta con los vectores en orden:

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

Sin embargo, para simplificar el proceso es recomendable escoger la matriz de paridad estándar, H, reordenando los vectores. Y asociada a esta matriz de paridad, la matriz generatriz estándar, G, como podemos ver a continuación.

[4] No se van a estudiar con detalle en este texto, pero el lector que tenga curiosidad puede consultar los libros de la bibliografía, [4, Capítulo 8] entre otros.

$$H = \left(\begin{array}{cccc|ccc} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{array}\right) \qquad G = \left(\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array}\right)$$

En rojo se han marcado las matrices A y $-A^T$, aunque al ser un código binario el signo negativo no influye. Además, se puede comprobar en la figura 3.1 que verifican que $G\dot{H}^T = 0$. También hay que tener en cuenta que estamos trabajando en el cuerpo $\mathbb{F}_2$, por esto la orden `Mod2`[5] *tras hacer el producto de matrices en Mathematica.*

```
G = {{1,0,0,0,0,1,1},{0,1,0,0,1,0,1},{0,0,1,0,1,1,0},{0,0,0,1,1,1,1}};

H = {{0,1,1,1,1,0,0},{1,0,1,1,0,1,0},{1,1,0,1,0,0,1}};

G.Transpose[H] // Mod2 // MatrixForm
   transposición        forma de matriz

{{0,0,0},{0,0,0},{0,0,0},{0,0,0}}
```

Figura 3.1: Comprobación $G \cdot H^T = 0$.

Una vez que ya sabemos con qué código estamos trabajando, supongamos que queremos enviar el carácter Z *a través de un canal no fiable. En el estándar ASCII, la* Z *se corresponde con el código numérico* 90*, que en binario es el número* 1011010*, que es el mensaje que enviaremos. Además, como estamos trabajando con un código de dimensión* 4*, hay que dividirlo en submensajes de esta longitud, por lo que añadimos un cero a la izquierda (que no influye en la representación binaria de* 90*) y lo dividimos en los dos mensajes* 0101 *y* 1010*. Es evidente que el orden en el que se envían los mensajes es importante.*

Por último, desde el punto de vista del emisor, queda calcular las palabras del código asociadas a estos dos submensajes, simplemente multiplicándolos por la matriz generatriz. Y enviarlas.

```
{0, 1, 0, 1}.G // Mod2
{0, 1, 0, 1, 0, 1, 0}

{1, 0, 1, 0}.G // Mod2
{1, 0, 1, 0, 1, 0, 1}
```

Figura 3.2: Cálculo de las palabras del código que se van a transmitir.

Se puede comprobar que en ambos casos las palabras que se van a transmitir contienen el mensaje explícitamente (los cuatro primeros símbolos), siendo el resto de la palabra información redundante.

[5] La función `Mod2` se ha creado únicamente para simplificar los cálculos, pero no aporta nada diferente a la función habitual `Mod`. `Mod2[a] = Mod[a,2]`.

Por otro lado, el emisor que sabe que va a recibir un mensaje codificado con este código, tiene que construir la tabla Síndrome-Líder, en la que se asocia cada líder de clase con su síndrome. Como en este ejemplo estamos trabajando con un $[7,4]$*–código binario* ($q = 2$)*, por el corolario 3.18 sabemos que existen* $q^{n-k} = 2^{7-4} = 8$ *clases diferentes, siendo una de ellas el propio código, por lo que nuestra tabla tendrá 7 entradas. Además, sabemos que los vectores que sólo tienen una coordenada no nula tienen que ser líderes de clase, por lo que ya tenemos los siete líderes de clase. Para calcular sus síndromes basta con multiplicarlos por la traspuesta de la matriz de paridad (3.19), como se ha hecho en la figura 3.3 con ayuda del software Mathematica.*

```
TSL = {};
For[i = 1, i ≤ 7, i++,
 para cada
  v = ConstantArray[0, 7];
       arreglo constante
  v[[i]] = 1;
  s = v.Transpose[H] // Mod2;
          transposición
  AppendTo[TSL, {v, s}];
   añade al final
 ];
TSL // MatrixForm
          forma de matriz
```

Líder	Síndrome
1 000 000	011
0 100 000	101
0 010 000	110
0 001 000	111
0 000 100	100
0 000 010	010
0 000 001	001

Figura 3.3: Cálculo de la tabla Síndrome-Líder.

Una vez construida el receptor ya puede decodificar los mensajes. Supongamos que se reciben los vectores 0111010 *y* 1010101*, en ese orden. El receptor multiplicará ambos vectores por la matriz de paridad para comprobar si son palabras del código o, en caso de que no lo sean, saber cual es su síndrome. Como vemos en la figura 3.4, en la primera palabra ha habido algún error, pero la segunda ha llegado tal cual se envió, ya que su síndrome es nulo y por tanto es una palabra del código.*

```
{0, 1, 1, 1, 0, 1, 0}.Transpose[H] // Mod2
                       transposición
{1, 0, 1, 0, 1, 0, 1}.Transpose[H] // Mod2
                       transposición

{1, 1, 0}

{0, 0, 0}
```

Figura 3.4: Cálculo del síndrome de las palabras recibidas.

En cuanto al primer vector, hemos visto que no es una palabra del código, por lo que ha habido algún error durante la transmisión que tenemos que corregir. Para esto nos fijamos

en el síndrome, 110*, que en la tabla 3.3 vemos que se corresponde con el vector que tiene un uno en la tercera coordenada. Esto quiere decir que la palabra que se transmitió es aquella tal que* $x + 0010000 = 0111010$*, y por tanto la palabra que el emisor envió es* 0101010*.*

Ahora que ya sabemos cuales son las palabras del código que el emisor envió, hay que decodificarlas para obtener la información que quería enviar sin los símbolos redundantes. En este caso es muy sencillo porque hemos utilizado la matriz generatriz estándar, por lo que la información son las 4 primeras coordenadas de cada palabra, esto es 0101 *y* 1010*, que uniéndolas obtiene el número* 01011010*. Haciendo el proceso inverso, esta es la representación binaria del número 90, que se corresponde con el carácter* Z *en el estándar ASCII. Comunicación exitosa.*

Cabe destacar que este código solo funciona cuando en la transmisión hay un único error. Supongamos que el emisor quiere enviar la información 1110*. La codificaría a la palabra* 1110000*, que es la que enviará. Sin embargo, durante la transmisión ocurren dos errores, en las coordenadas* 3 *y* 5*, y llega la palabra* 1100100*.*

Al multiplicar este vector por la matriz de paridad, el receptor obtendrá el síndrome 010*, que le llevará a pensar que ha habido un error en la sexta coordenada, corregirá a la palabra del código* 1100110 *y pensará que la información que quería transmitir el emisor es* 1100*. De aquí la importancia de elegir bien el código que se utiliza para según qué canales.*

3.3 Códigos cíclicos

El método Síndrome-Líder que hemos visto en la sección anterior es sencillo de implementar, sin más que guardar en memoria la relación de síndromes y errores, pero se complica cuando trabajamos con códigos lineales grandes. Es por esto que se buscan formas más sencillas de decodificar, manteniendo las buenas propiedades de los códigos lineales. Así surgen los códigos cíclicos, que son un tipo especial de códigos lineales, con la propiedad adicional de que al desplazar los símbolos de una palabra, el resultado es una nueva palabra del código. Veamos la definición formal:

Definición 3.23. Sea $\mathcal{C}$ un $[n,k]$–código lineal q–ario, diremos que es un **código cíclico** si dada cualquier palabra $c = (c_0, c_1, \ldots, c_{n-1}) \in \mathcal{C}$, la palabra $c^R = (c_{n-1}, c_0, c_1, \ldots, c_{n-2})$ también pertenece a $\mathcal{C}$.

Los códigos cíclicos son un tipo especial, no muy numeroso, dentro de los códigos lineales, pero dado un cuerpo $\mathbb{F}_q$ y una longitud de código $n \geq 3$ siempre existen al menos los siguientes códigos cíclicos en $\mathbb{F}_q^n$:

- El $[n,0]$–código trivial que contiene únicamente la palabra nula.
- El $[n,1]$–código que contiene todas las palabras $(a,a,\ldots,a)$, con $a \in \mathbb{F}_q$, llamado el *código de repetición*.
- El $[n,n-1]$–código con todas las palabras $c = (c_0, c_1, \ldots, c_{n-1})$ tales que $\sum_i c_i = 0$, llamado el *código de paridad unitaria* (un único dígito de control).
- El $[n,n]$–código que contiene todas las palabras de longitud n, que es el código que no tiene dígitos de control.

Para algunos valores de n y de q, estos son los únicos códigos cíclicos que existen, pero a menudo hay otros más interesantes como veremos más adelante.

El principal atractivo de los códigos cíclicos es el hecho de poder ver las palabras del código como polinomios, y las permutaciones cíclicas como una operación de módulo en el anillo de polinomios. Veamos esto con más claridad, empezando por la relación entre las palabras del código y los polinomios.

Definición 3.24. Dado $\mathcal{C}$ un $[n,k]$–código lineal q–ario y una palabra cualquiera del código $c = (c_0, c_1, \ldots, c_{n-1}) \in \mathcal{C}$, se define la **función generatriz** de c como el polinomio

$$c(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_{n-1} x^{n-1}$$

Con esta definición, la función generatriz de la permutación cíclica de una palabra del código será $c^R(x) = x \cdot c(x) \bmod x^n - 1$. Con la notación que hemos visto en el primer capítulo (2.6), la permutación cíclica de una palabra $c(x)$ es la forma normal de $x \cdot c(x)$ módulo $x^n - 1$.

Para simplificar la notación, escribiremos $[p(x)]_n$ para referirnos a la forma normal de $p(x)$ módulo $x^n - 1$. Así, la permutación cíclica de $c(x)$ será

$$c^R(x) = [x \cdot c(x)]_n$$

Esta caracterización como polinomios es tan útil que hablaremos indistintamente de las palabras como n–uplas o como polinomios. Desde este punto de vista, un código lineal $\mathcal{C}$ será un conjunto de polinomios de forma que las combinaciones lineales de polinomios también están en $\mathcal{C}$; y un código cíclico es aquel en el que para cualquier palabra del código $c(x)$, $[x \cdot c(x)]_n$ también está en el código.

Lo que hace interesante a los códigos cíclicos es el hecho de que, como conjuntos de polinomios, tienen estructura de ideal.

Teorema 3.25. Dado $\mathcal{C}$ un $[n,k]$–código cíclico q–ario, una palabra cualquiera del código $c(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_{n-1} x^{n-1}$ y otro polinomio cualquiera de $F_q[x]$, $p(x)$, entonces $[p(x) \cdot c(x)]_n$ también está en el código.

Demostración:

Supongamos que $p(x) = \sum_i p_i x^i$ *con* $p_i \in \mathbb{F}_q$. *Tenemos que*

$$[p(x) \cdot c(x)]_n = \left[\sum_i p_i x^i \cdot c(x) \right]_n = \sum_i p_i \left[x^i \cdot c(x) \right]_n$$

Sabemos que $\left[x^i \cdot c(x) \right]_n$ *es una palabra del código (el resultado de hacer i permutaciones cíclicas a* $c(x)$*), y por tanto es una combinación lineal de palabras del código, que sabemos que está en* $\mathcal{C}$.

■

Ahora que hemos visto que los códigos cíclicos se pueden ver como ideales de polinomios en una variable, vamos a estudiarlos como tales. Todo ideal de polinomios univariable tiene un generador, que va a ser el polinomio de menor grado.

Definición 3.26. Dado un código cíclico $\mathcal{C}$, llamaremos **polinomio generador** a una palabra de grado mínimo de $\mathcal{C}$. Se suele denotar por $g(x)$.

Lema 3.27. Dado un código cíclico $\mathcal{C}$ con polinomio generador $g(x)$, se verifica que:

i. Si $g'(x)$ es otro polinomio generador de $\mathcal{C}$, entonces $g'(x) = \lambda g(x)$ para algún $0 \neq \lambda \in \mathbb{F}$.

ii. Si $p(x)$ es un polinomio tal que $[p(x)]_n$ es una palabra del código, entonces $g(x)$ divide a $p(x)$.

Como puede haber varios polinomios generadores, es habitual utilizar el polinomio generador mónico por simplificar, aunque son equivalentes todos.

Ahora estamos en condiciones de demostrar uno de los resultados más importantes de esta sección: la correspondencia biunívoca entre los códigos de longitud n y los divisores de $x^n - 1$.

Teorema 3.28. Se verifica que

a) Dado $\mathcal{C}$ un $[n,k]$–código cíclico q–ario, su polinomio generador $g(x)$ es un divisor de $x^n - 1$. Además, un polinomio $c(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_{n-1} x^{n-1}$ está en el código si y solo si es divisible por $g(x)$. Se verifica que $k = n - gr(g)$.

b) Recíprocamente, si $g(x)$ es un divisor de $x^n - 1 \in \mathbb{F}_q[x]$, existe un $[n,k]$–código cíclico q–ario con $g(x)$ como polinomio generador tal que $k = n - gr(g)$, formado por todos los múltiplos de $g(x)$ de grado menor o igual que $n-1$.

Demostración:
En primer lugar vamos a demostrar el apartado a). Partimos del supuesto de que $\mathcal{C}$ es un $[n,k]$–código cíclico q–ario y $g(x)$ es su polinomio generador.

- *Es evidente que $[x^n - 1]_n = 0$ es una palabra de $\mathcal{C}$ y por el lema 3.27 tenemos que $g(x)$ divide a $x^n - 1$.*

$\Rightarrow|$ *Si $c(x)$ es una palabra del código, $[c(x)]_n = c(x)$ y por el lema 3.27 sabemos que es divisible por $g(x)$.*

$\Leftarrow|$ *Dado un múltiplo de $g(x)$ con $gr(g) \leq n-1$, es claro que es una palabra del código por la estructura de ideal de $\mathcal{C}$.*

- *Como todas las palabras del código son múltiplos de $g(x)$, se pueden ver como $c(x) = g(x) \cdot p(x)$ para algún polinomio $p(x)$. Como el grado de $c(x)$ es menor o igual que $n-1$, el grado de $p(x)$ tiene que ser, a lo sumo, $gr(p) \leq n - 1 - gr(g)$. Es decir, la palabra asociada a $p(x)$, que coincide con la dimensión de $\mathcal{C}$, es $k = n - gr(g)$.*

Ahora pasamos a demostrar el recíproco b):
Partimos de un divisor de $x^n - 1$, $g(x)$. Es fácil ver que el conjunto de múltiplos de $g(x)$ de grado menor o igual que $n-1$ es un espacio vectorial (la suma y el producto por escalares se quedan en el conjunto) y por tanto un $[n,k]$–código lineal, siendo k el grado de los polinomios por el que se puede multiplicar $g(x)$ y que resulte en grado menor o igual que $n-1$, esto es, $k = n - gr(g)$.

Queda por demostrar que es un código cíclico, para lo que hay que comprobar que la permutación cíclica de cualquier palabra se queda en el código. Dado una palabra

del código cualquiera $p(x)g(x)$ hemos visto que su permutación cíclica es $[x \cdot p(x)g(x)]_n$ y utilizando que $g(x)$ divide a $x^n - 1$ tenemos:

$$[x \cdot p(x)g(x)]_n \text{ módulo } g(x) = x \cdot p(x)g(x) \text{ módulo } g(x) = 0$$

Y por tanto $[x \cdot p(x)g(x)]_n$ es un múltiplo de $g(x)$ y está en $\mathcal{C}$.

■

Este teorema muestra la importancia del polinomio generador, pero a la hora de utilizar un código cíclico puede ser igual de importante el *polinomio de paridad*:

Definición 3.29. Dado $\mathcal{C}$ un $[n,k]$–código cíclico q–ario con polinomio generador $g(x)$, llamaremos **polinomio de paridad** $h(x)$ al polinomio resultado de:

$$h(x) = \frac{x^n - 1}{g(x)}$$

Por último, vamos a presentar dos corolarios con distintas matrices generatrices y de paridad.

Corolario 3.30. Dado $\mathcal{C}$ un $[n,k]$–código cíclico q–ario cuyos polinomios generador y de paridad son $g(x) = g_0 + g_1 x + \cdots + g_r x^r$ y $h(x) = h_0 + h_1 x + \cdots + h_k x^k$ respectivamente, si definimos $\tilde{h} = h_k + h_{k-1} x + \cdots + h_0 x^k$ como el *recíproco* del polinomio de paridad, entonces las siguientes son sus matrices generatriz y de paridad:

$$G = \begin{pmatrix} g_0 & g_1 & \cdots & \cdots & g_r & 0 & \cdots & \cdots & 0 \\ 0 & g_0 & g_1 & \cdots & \cdots & g_r & 0 & \cdots & 0 \\ \vdots & & & \ddots & & & \ddots & & \vdots \\ \vdots & & & & \ddots & & & \ddots & 0 \\ 0 & \cdots & \cdots & 0 & g_0 & g_1 & \cdots & \cdots & g_r \end{pmatrix} = \begin{pmatrix} g(x) \\ x \cdot g(x) \\ x^2 \cdot g(x) \\ \vdots \\ x^{k-1} \cdot g(x) \end{pmatrix}$$

$$H = \begin{pmatrix} h_k & h_{k-1} & \cdots & \cdots & h_0 & 0 & \cdots & \cdots & 0 \\ 0 & h_k & h_{k-1} & \cdots & \cdots & h_0 & 0 & \cdots & 0 \\ \vdots & & & \ddots & & & \ddots & & \vdots \\ \vdots & & & & \ddots & & & \ddots & 0 \\ 0 & \cdots & \cdots & 0 & h_k & h_{k-1} & \cdots & \cdots & h_0 \end{pmatrix} = \begin{pmatrix} \tilde{h}(x) \\ x \cdot \tilde{h}(x) \\ x^2 \cdot \tilde{h}(x) \\ \vdots \\ x^{r-1} \cdot \tilde{h}(x) \end{pmatrix}$$

Además, para codificar un vector $m = (m_0, m_1, \ldots, m_{k-1})$ como la palabra $c = m \cdot G$, se puede calcular utilizando polinomios ya que se verifica la igualdad

$$c(x) = m(x)g(x)$$

Utilizando las matrices de este corolario es muy sencillo codificar un mensaje, sin más que multiplicar su función generatriz por el polinomio generador del código, pero a la hora de calcular el síndrome de un mensaje recibido no hay otra forma que utilizar el producto de matrices, que puede ser muy costoso.

Es por esto que se suelen utilizar las matrices *sistemáticas* que se describen en el siguiente corolario que, a pesar de ser algo más complicadas de calcular, simplifican la forma de codificar y descodificar.

Corolario 3.31. Dado $\mathcal{C}$ un $[n,k]$–código cíclico q–ario cuyo polinomio generador es $g(x) = g_0 + g_1 x + \cdots + g_r x^r$, para cada $i = 0,\ldots,k-1$ definimos G_i como el vector cuya función generatriz es el siguiente polinomio:

$$G_i(x) = x^{r+i} + (x^{r+i} \bmod g(x))$$

Utilizando estos k vectores como filas definimos la matriz generatriz como:

$$G = \begin{pmatrix} G_0 \\ G_1 \\ \vdots \\ G_{k-1} \end{pmatrix}$$

De forma similar definimos los vectores H_j que serán las columnas de la matriz de paridad H:

$$H_j(x) = x^j \bmod g(x) \qquad\qquad H = \begin{pmatrix} H_0^T & H_1^T & \ldots & H_{n-1}^T \end{pmatrix}$$

Además, para codificar un mensaje m basta calcular el polinomio

$$c(x) = x^r m(x) - (x^r m(x) \bmod g(x))$$

Y el síndrome s de un vector recibido r se puede calcular como

$$s(x) = r(x) \bmod g(x)$$

Estas matrices se conocen como *sistemáticas* porque incluyen la matriz identidad, lo que hace muy sencilla la decodificación. En este caso, la matriz identidad la forman las k columnas de la derecha de G.
Para comprobarlo basta con estudiar la forma de sus vectores fila:

$$G_i(x) = x^{r+i} + (x^{r+i} \bmod g(x))$$

Como $g(x)$ tiene grado r, cualquier polinomio en forma normal respecto a $g(x)$ tendrá, como máximo, grado $r-1$, por lo que el único monomio con grado r o más de G_i es x^{r+i}. Es decir, la única coordenada no nula a partir de la r–ésima es un 1 en la $r+i$–ésima, por lo que las últimas k columnas forman la matriz identidad.

De forma similar, en H la matriz identidad la forman las r primeras columnas, ya que los r polinomios $H_j(x)$ con grado menor o igual que $r-1$ ya están en forma normal respecto a $g(x)$, por lo que las únicas coordenadas no nulas en dichas columnas serán los 1 de la fila j–ésima.

Ejemplo 3.32. *El código de Hamming $Ham(3,2)$ del ejemplo 3.22 se puede ver como un código cíclico con polinomio generador $g(x) = 1 + x + x^3$. Recordamos que este ejemplo es un $[7,4,3]$–código lineal.*

Para calcular el polinomio de paridad, como es un código de longitud 7, dividimos $x^7 - 1$ entre $g(x)$ y obtenemos

$$h(x) = \frac{x^7 - 1}{x^3 + x + 1} = x^4 + x^2 + x + 1$$

Con este polinomio ya podemos calcular unas matrices generatriz y de paridad para este código cíclico segun el corolario 3.30.

$$G = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix} \qquad H = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$$

Sin embargo, ya hemos mencionado que es preferible calcular las matrices sistemáticas del corolario 3.31 para facilitar los cálculos:

$$\begin{aligned} G_0 &= x^3 + (x^3 \ mod\ x^3 + x + 1) = x^3 + (x + 1) \\ G_1 &= x^4 + (x^4 \ mod\ x^3 + x + 1) = x^4 + (x^2 + x) \\ G_2 &= x^5 + (x^5 \ mod\ x^3 + x + 1) = x^5 + (x^2 + x + 1) \\ G_3 &= x^6 + (x^6 \ mod\ x^3 + x + 1) = x^6 + (x^2 + 1) \end{aligned}$$

$$\begin{aligned} H_0 &= 1 \ \ mod\ x^3 + x + 1 = 1 \\ H_1 &= x \ \ mod\ x^3 + x + 1 = x \\ H_2 &= x^2\ mod\ x^3 + x + 1 = x^2 \\ H_3 &= x^3\ mod\ x^3 + x + 1 = x + 1 \\ H_4 &= x^4\ mod\ x^3 + x + 1 = x^2 + x \\ H_5 &= x^5\ mod\ x^3 + x + 1 = x^2 + x + 1 \\ H_6 &= x^6\ mod\ x^3 + x + 1 = x^2 + 1 \end{aligned}$$

Y por tanto las matrices generatriz y de paridad sistemáticas son

$$G' = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \qquad H' = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$$

Es fácil comprobar que estas matrices de paridad se corresponden con el código de Hamming, ya que las columnas de H y H' verifican la definición 3.21. Para comprobar que las matrices generatrices también son correctas bastaría comprobar $G \cdot H = 0$ y $G' \cdot H' = 0$.

Por último, notar que las matrices sistemáticas se pueden ver como

$$G' = [A \mid I_4] \qquad H' = \left[I_3 \mid -A^T\right]$$

3.4 Códigos BCH

Ya hemos visto (3.21) que los códigos de Hamming son una familia de códigos de distintas longitudes y dimensiones, pero que siempre son capaces de corregir un único error. Esto puede no ser suficiente en muchas situaciones, por lo que surge la necesidad de buscar códigos capaces de corregir más errores. Dichos códigos de Hamming pueden corregir un error añadiendo r símbolos de información redundante, que se traducen en tener una matriz de paridad con r filas

$$H = \begin{pmatrix} v_0 & v_1 & \dots & v_{n-1} \end{pmatrix}$$

donde v_i son los n vectores columna de $\mathbb{F}_2^r$.

La intuición nos dice que si añadiésemos otros r símbolos redundantes podríamos corregir un error más, es decir, otras r filas donde w_i son los mismos vectores de $\mathbb{F}_2^r$ aunque no necesariamente en el mismo orden. De hecho, podemos ver los w_i como el resultado de aplicar f, una biyección de $\mathbb{F}_2^r$ en sí mismo, a los vectores v_i.

$$H = \begin{pmatrix} v_0 & v_1 & \dots & v_{n-1} \\ w_0 & w_1 & \dots & w_{n-1} \end{pmatrix} = \begin{pmatrix} v_0 & v_1 & \dots & v_{n-1} \\ f(v_0) & f(v_1) & \dots & f(v_{n-1}) \end{pmatrix}$$

Sin embargo, no cualquier elección de f sirve para construir un código capaz de corregir más de un error, ya que hay que asegurar que todos los vectores de peso menor o igual que 2 tienen síndrome distinto. De otra forma, al obtener un síndrome asociado a dos de estos vectores, no sabríamos decidir cuál de los dos ha sido el error cometido. Se puede comprobar que esto no es posible si f es lineal, por lo que hay que buscar la forma de multiplicar entre sí los vectores de forma que esta multiplicación sea consistente con la estructura de espacio vectorial. Esto es posible si vemos los vectores de $\mathbb{F}_2^r$ como elementos del cuerpo finito (o de Galois) $\mathbb{F}_{2^r}$.

Ahora, las funciones $f : \mathbb{F}_{2^r} \to \mathbb{F}_{2^r}$ se pueden ver como polinomios. Ya hemos dicho que las funciones lineales, polinomios de grado 1, no funcionan; y se puede comprobar que los de grado 2 tampoco, pero $f(v) = v^3$ sí funciona.

Para dejar claro el cambio de punto de vista, vamos a cambiar la notación para identificar como $(a_0, a_1, \dots, a_{n-1})$ a los elementos de $\mathbb{F}_{2^r}$ que denotábamos por $(v_0, v_1, \dots, v_{n-1})$ al considerarlos en $\mathbb{F}_2^r$.

Con todo esto tenemos que la matriz de paridad

$$H = \begin{pmatrix} a_0 & a_1 & \dots & a_{n-1} \\ a_0^3 & a_1^3 & \dots & a_{n-1}^3 \end{pmatrix}$$

define un código lineal de longitud $n = 2^r - 1$ capaz de corregir 2 errores. Además, si volvemos a ver los elementos de $\mathbb{F}_{2^r}$ como vectores de $\mathbb{F}_2^r$, H estará compuesta de $2r$ filas y por tanto estamos hablando de un código binario de dimensión $k = 2^r - 1 - 2r$.

Los **códigos BCH** precisamente generalizan esta construcción para cualquier número de errores, como muestra el siguiente teorema:

Teorema 3.33. Sean $(a_0, a_1, \dots, a_{n-1})$ una lista de n elementos no nulos diferentes de $\mathbb{F}_{2^r}$ y t un natural tal que $t \leq \frac{n-1}{2}$, entonces la matriz

$$H = \begin{pmatrix} a_0 & a_1 & \dots & a_{n-1} \\ a_0^3 & a_1^3 & \dots & a_{n-1}^3 \\ \vdots & \vdots & & \vdots \\ a_0^{2t-1} & a_1^{2t-1} & \dots & a_{n-1}^{2t-1} \end{pmatrix}$$

es la matriz de paridad de un código de longitud n capaz de corregir t errores y de dimensión $k \geq n - rt$.

Este código se puede ver como el conjunto de palabras $c = (c_0, c_1, \dots c_{n-1})$ que verifican este sistema de ecuaciones:

$$\sum_{i=0}^{n-1} c_i a_i^j = 0 \qquad j = 1, 3, 5, \dots, 2t-1 \tag{3.1}$$

O equivalentemente

$$\sum_{i=0}^{n-1} c_i a_i^j = 0 \qquad j = 1, 2, 3, \ldots, 2t$$

Definición 3.34. Los códigos descritos en el teorema 3.33 se conocen como **códigos BCH**.

La importancia de esta familia de códigos no viene de su estructura, hay códigos capaces de corregir más de un error con más dimensión y menos longitud, si no del hecho de que se pueden ver como códigos cíclicos facilitando mucho su codificación y decodificación.

Lo único que hace falta para que el código BCH sea un código cíclico es elegir correctamente el orden de los elementos de $\mathbb{F}_{2^r}$. Este orden es

$$(1, a, a^2, \ldots, a^{n-1})$$

donde n tiene que ser un divisor de $2^r - 1$ y a es un elemento de $\mathbb{F}_{2^r}$ de orden n, es decir, $a^n = 1$.

Con esta elección de los elementos de H, la ecuación 3.1 quedaría como

$$\sum_{i=0}^{n-1} c_i a^{ij} = 0 \qquad j = 1, 3, 5, \ldots, 2t-1$$

o equivalentemente, viendo c como un polinomio

$$c(a^j) = 0 \qquad j = 1, 3, 5, \ldots, 2t-1 \quad (\text{o} \quad j = 1, 2, 3, \ldots, 2t)$$

Para ver que es un código cíclico, vamos a comprobar que la permutación cíclica de una palabra del código también verifica estas ecuaciones. Sea $c(x)$ una palabra del código, su permutación cíclica es $c^R(x) = [x \cdot c(x)]_n$, es decir,

$$c^R(x) = x \cdot c(x) + p(x)(x^n - 1)$$

para algún polinomio $p(x)$. La ecuación j–ésima para ver si c^R está en el código será

$$c^R(a^j) = a^j \cdot c(a^j) + p(a^j)(a^{jn} - 1)$$

pero sabemos que $c(a^j) = 0$ porque c es una palabra del código y que $a^n = 1$ ya que a tiene orden n. Con esto se comprueba que $c^R(a^j) = 0$ y por tanto es un código cíclico.

Ahora bien, por ser un código cíclico, sabemos que debe tener un polinomio generador, que por definición es el polinomio de menor grado que verifique

$$g(a) = g(a^3) = \cdots = g(a^{2t-1}) = 0$$

El problema para calcularlo es que las sucesivas potencias de a son elementos de $\mathbb{F}_{2^r}$, mientras que $g(x)$ queremos verlo como el polinomio generador de un código binario, por lo que sus coeficientes están en $\mathbb{F}_2$.

Los siguientes resultados permiten calcularlo utilizando conceptos de la Teoría de Galois.

Definición 3.35. Dado un elemento $a \in \mathbb{F}_{q^r}$, se definen los **conjugados** de a como los elementos de la sucesión $a, a^q, a^{q^2}, \dots a^{q^{k-1}}$ donde k es el menor número que verifica $a^{q^k} = a$ y que llamaremos **grado** de a.

Definición 3.36. Dado un elemento $a \in \mathbb{F}_{q^r}$, llamaremos **polinomio mínimo** de a, $p_a(x)$, al polinomio de menor grado con coeficientes en $\mathbb{F}_q$ que verifica $p_a(a) = 0$.

Este polinomio es irreducible sobre $\mathbb{F}_q$, pero en $\mathbb{F}_{q^r}$ se puede ver como

$$p_a(x) = (x-a)(x-a^q)(x-a^{q^2})\dots(x-a^{q^{k-1}})$$

por lo que el grado del polinomio mínimo coincide con el grado de a.

El **polinomio mínimo** de un conjunto de elementos $\{a_1, a_2, \dots, a_m\}$ es el polinomio de menor grado con coeficientes en $\mathbb{F}_q$ que verifica $p(a_1) = p(a_2) = \dots = p(a_m) = 0$. Visto así, el polinomio generador de un código BCH será el polinomio mínimo de $A = \{a, a^3, a^5, \dots, a^{2t-1}\}$.

Así, en el caso binario, si definimos A^* como el conjunto de todos los conjugados de los elementos de A, $A^* = \{\, \beta^{2i} : \beta \in A,\ i \geq 0 \,\}$, el polinomio generador del código será

$$g(x) = \prod_{\beta \in A^*} (x - \beta)$$

Vamos a resumir el caso binario de estos resultados en un solo teorema y vemos un ejemplo para clarificar.

Teorema 3.37. Dado un natural r, un elemento a de $\mathbb{F}_{2^r}$ de grado n y un natural t que verifique $t \leq \frac{n-1}{2}$, podemos definir un código BCH cíclico binario de longitud n, capaz de corregir t errores, tal que una palabra $c(x)$ está en el código si verifica las siguientes t ecuaciones:

$$c(a^j) = 0 \qquad j = 1, 3, 5, \dots, 2t-1$$

Además, el polinomio generador de este código es el polinomio mínimo sobre $\mathbb{F}_2$ de

$$A = \{a, a^3, a^5, \dots, a^{2t-1}\}$$

que se calcula como

$$g(x) = \prod_{\beta \in A^*} (x - \beta)$$

donde $A^* = \{\, \beta^{2i} : \beta \in A,\ i \geq 0 \,\}$.

Ejemplo 3.38. *Consideremos C un código BCH con $r = 4$, de longitud $n = 15$ y capaz de corregir $t = 3$ errores. Y sea a un elemento de $\mathbb{F}_{16}$ de orden 15 ($a^{15} = 1$).*
El polinomio generador de C será el polinomio mínimo del conjunto $A = \{a, a^3, a^5\}$. Y los conjugados de estos elementos son:

- *Los conjugados de a son $\{a, a^2, a^4, a^8\}$, ya que $a^{16} = a$.*
- *Los conjugados de a^3 son $\{a^3, a^6, a^{12}, a^{24} = a^9\}$, ya que $a^{18} = a^3$.*
- *Los conjugados de a^5 son $\{a^5, a^{10}\}$.*

Y por tanto, el conjunto de los conjugados es

$$A^* = \{a, a^2, a^3, a^4, a^5, a^6, a^8, a^9, a^{10}, a^{12}\}$$

por lo que sabemos que $g(x)$ tendrá grado $|A^*| = 10$, que será el producto de los polinomios mínimos de a, a^3 y a^5.

Para calcular explícitamente estos polinomios mínimos, y por tanto $g(x)$, hay que utilizar una representación concreta de $\mathbb{F}_{16}$. En este ejemplo vamos a verlo como el conjunto de potencias de un elemento a que verifica $a^4 = a + 1$. Los elementos de este cuerpo los vamos a ver como polinomios en a de grado menor o igual que 3, por ejemplo $a^7 = a^4 a^3 = a^3(a+1) = a^4 + a^3 = a^3 + a + 1$.

Así, el polinomio mínimo de a es, por definición,

$$p_1(x) = x^4 + x + 1$$

El polinomio mínimo de a^3 es un polinomio de grado 4 (porque son 4 conjugados), $p_3(x) = \sigma_4 x^4 + \sigma_3 x^3 + \sigma_2 x^2 + \sigma_1 x + \sigma_0$, con $\sigma_i \in \mathbb{F}_2$, que verifica $p_3(a^3) = 0$. Vamos a calcularlo. En primer lugar, las potencia de a^3:

- $(a^3)^2 = a^6 = a^2(a+1) = a^3 + a^2$
- $(a^3)^3 = a^9 = a^4 a^4 a = (a+1)(a+1)a = (a^2+1)a = a^3 + a$
- $(a^3)^4 = (a+1)(a+1)(a+1) = (a^2+1)(a+1) = a^3 + a^2 + a + 1$

Con esto calculado, tenemos que

$$\begin{aligned} p_3(a^3) =& \sigma_4(a^3 + a^2 + a + 1) + \sigma_3(a^3 + a) + \sigma_2(a^3 + a^2) + \sigma_1 a^3 + \sigma_0 \\ =& a^3(\sigma_4 + \sigma_3 + \sigma_2 + \sigma_1) + a^2(\sigma_4 + \sigma_2) + a(\sigma_4 + \sigma_3) + \sigma_4 + \sigma_0 = 0 \end{aligned}$$

Como los σ_i son elementos de $\mathbb{F}_2$, tenemos que $\sigma_4 = \sigma_3 = \sigma_2 = \sigma_1 = \sigma_0$, por lo que la única solución no trivial es que el polinomio mínimo de a^3 sea

$$p_3(x) = x^4 + x^3 + x^2 + x + 1$$

De forma análoga se puede calcular el polinomio mínimo de a^5, que a priori sabemos que será de grado 2, y que resulta ser

$$p_5(x) = x^2 + x + 1$$

Finalmente, el polinomio generador de este código BCH cíclico es

$$g(x) = p_1(x)p_3(x)p_5(x) = (x^4 + x + 1)(x^4 + x^3 + x^2 + x + 1)(x^2 + x + 1)$$

$$g(x) = x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1$$

Estos códigos BCH, con la construcción que hemos visto como códigos cíclicos binarios en el teorema 3.37, los podríamos utilizar como cualquier otro código binario, codificando y decodificando según el corolario 3.31, pero existen algoritmos más eficientes. Un ejemplo de ello es el que vamos a estudiar en el siguiente capítulo utilizando bases de Gröbner.

A modo de adelanto, con este algoritmo vamos a codificar un mensaje m, visto como un polinomio, simplemente multiplicándolo por el polinomio generador.

$$c(x) = m(x)g(x)$$

A la hora de decodificar, será necesario calcular los síndromes que definimos a continuación.

Definición 3.39. Sea un $[n,k]$–código BCH cíclico binario C construido utilizando el elemento $a \in \mathbb{F}_{2^r}$ de orden n. Dado un mensaje recibido $r(x)$, se definen los síndromes de $r(x)$ como

$$s_i = r(a^i) \qquad i \geq 0$$

Un receptor que se esté comunicando con este código, recibirá un mensaje que es susceptible de tener errores $r(x) = c(x) + e(x)$. El síndrome que acabamos de definir depende únicamente del error de transmisión, ya que por la definición del código

$$s_i = r(a^i) = c(a^i) + e(a^i) = e(a^i) \qquad i \geq 0$$

Si la palabra estuviera en el código, todos los síndromes serían nulos y viceversa.

Por último, vamos a hacer un inciso para ver cómo trabajar con cuerpos finitos con el software Mathematica:

Los también llamados *Cuerpos de Galois* están implementados en Mathematica como un paquete independiente, `FiniteFields`, que hay que instalar con una orden sencilla antes de poder utilizarlo. El manual completo de este paquete se puede encontrar en la web [11].

Los distintos cuerpos se definen con la orden `GF[_]`, pero hay varias elegir los argumentos. La más sencilla consiste en definir el cuerpo $\mathbb{F}_{q^r}$ como `GF[q,r]`. Sin embargo, si se quiere definir a partir del polinomio mínimo hay que utilizar la sintaxis `GF[q,{r0,r1,r2,...}]` para definir un cuerpo q–ario con polinomio mínimo

$$r_0 + r_1 x + r_2 x^2 + \cdots = 0$$

Como muestra, el cuerpo utilizado en el ejemplo 3.38 se define con `GF[2,{1,1,0,0,1}]`.

La forma canónica de definir los elementos de estos cuerpos es como una n–upla, entre llaves y entre corchetes tras el nombre del cuerpo, con los coeficientes del polinomio sobre el generador del cuerpo que lo define. Precisamente, el generador será `GF[q,r][{0,1}]`. Estos elementos se pueden sumar, multiplicar y dividir con normalidad.

Para terminar, un ejemplo sencillo de utilización de este paquete.

```
<< FiniteFields`

F16 = GF[2, {1, 1, 0, 0, 1}]                    a^8

GF[2, {1, 1, 0, 0, 1}]                          {1, 0, 1, 0}_2

a = F16[{0, 1}]                                 a^2 + a^3

{0, 1, 0, 0}_2                                  {0, 0, 1, 1}_2

PowerList[F16]

{{1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 1, 0}, {0, 0, 0, 1}, {1, 1, 0, 0},
 {0, 1, 1, 0}, {0, 0, 1, 1}, {1, 1, 0, 1}, {1, 0, 1, 0}, {0, 1, 0, 1},
 {1, 1, 1, 0}, {0, 1, 1, 1}, {1, 1, 1, 1}, {1, 0, 1, 1}, {1, 0, 0, 1}}
```

Figura 3.5: Ejemplo de uso del paquete `FiniteFields`.

4

Algoritmo para decodificar códigos BCH utilizando Bases de Gröbner

En este capítulo vamos a ver un método para decodificar los códigos BCH cíclicos binarios que hemos visto en la sección anterior. Para esto, vamos a utilizar la potencia de las bases de Gröbner en un método puramente algebraico presentado por varios autores del IEEE[1] [3].

En la sección 3.2 de este texto vimos un método para decodificar cualquier código lineal, la tabla Síndrome-Líder, que consiste en calcular un síndrome y buscar en una tabla a qué error corresponde dicho síndrome. Este método es sencillo de implementar, pero ya mencionamos que es muy costoso porque hay que almacenar mucha información en memoria. Para el código del ejemplo 3.38, de longitud 15 y capaz de corregir 3 errores, habría que almacenar el síndrome-líder de todos los posibles errores de longitud 3 o menos. Esto es, $\binom{15}{3}+\binom{15}{2}+\binom{15}{1} = 2955$ entradas almacenadas, para un código relativamente pequeño. Más de 20 millones de entradas en un código de longitud 31 capaz de corregir 5 errores. Por esto, sería deseable un método por el que no hubiera que *buscar* una coincidencia, sino que el resultado de cierto algoritmo fuera el error directamente. Esto es, un método puramente algebraico como el que vamos a presentar.

Por otro lado, hay algunos métodos que mejoran la eficiencia de los algoritmos de búsqueda, pero a cambio no son capaces de corregir todos los errores que permite el código, por lo que se pierde potencia del código. El método que veremos a continuación, sin embargo, sí es capaz de corregir todos los errores que permite la distancia mínima del código.

La idea general de este método es la siguiente. Con los síndromes de la palabra recibida se generarán una serie de ecuaciones con tantas variables como errores queramos corregir. Con los polinomios asociados a estas ecuaciones calcularemos una Base de Gröbner y un ideal de eliminación con una única variable. Las raíces del polinomio generador de este ideal nos indicarán las posiciones de los errores.

Veámoslo con detalle.

En primer lugar, recordamos que estamos trabajando con un código BCH, de longitud n, diseñado para poder corregir t errores y definido de forma que un polinomio está en C si verifica

$$c(a^i) = 0 \qquad i = 1, 3, 5, \ldots, 2t-1$$

donde a es un elemento de $\mathbb{F}_{2^r}$ de orden n. Además, n tiene que ser un divisor de $2^r - 1$ y se verifica $t \leq \frac{n-1}{2}$. Como cualquier código cíclico, tiene un polinomio generador $g(x)$ que se puede calcular según indica el teorema 3.37.

También sabemos que, dado un mensaje $m(x)$, se codifica multiplicándolo por el polinomio generador: $c(x) = m(x)g(x)$. Y dado un mensaje recibido $r(x)$, se calculan sus síndromes como

$$s_i = r(a^i) \qquad i = 1, 3, 5, \ldots, 2t-1$$

[1] Institute of Electrical and Electronics Engineers

Estos síndromes se pueden ver como una suma de potencias con tantos términos como errores hayan ocurrido,

$$s_i = z_1^i + z_2^i + \cdots + z_v^i$$

donde v es el número de errores de transmisión, $z_j = a^{e_j}$ y e_j indica la posición del error.

El objetivo de la decodificación es encontrar las v posiciones de error a partir de los síndromes s_i. Esto es equivalente a resolver el sistema de ecuaciones

$$s_i = x_1^i + x_2^i + \cdots + x_v^i \qquad i = 1, 3, 5, \ldots, 2t-1$$

donde $x_j \in \mathbb{F}_{2^r}$ y $x_j^{n+1} = x_j$. Las soluciones de este sistema también anulan el denominado **polinomio localizador de errores**, definido para cualquier error corregible como

$$L(x) = \prod_{j=1}^{v} (x - z_j)$$

La existencia y unicidad[2] de una solución para este sistema de ecuaciones viene dada por el hecho de que cada error de longitud t o menos tendrá un conjunto de síndromes distinto.

Sin embargo, como a priori no sabemos cuántos errores ha habido en la transmisión, definiremos el siguiente sistema de ecuaciones, siendo w el número de errores que creemos que ha podido haber.

$$\begin{aligned} s_i &= x_1^i + x_2^i + \cdots + x_w^i & i &= 1, 3, 5, \ldots, 2t-1 \\ x_j^{n+1} &= x_j & j &= 1, 2, \ldots, w \end{aligned} \tag{4.1}$$

Según el resultado que obtengamos, sabremos si hemos acertado el número de errores, hay más o menos.

Al conjunto de polinomios que generan estas ecuaciones lo llamaremos F, que sabemos que es un subconjunto de $\mathbb{F}_2[x_1, x_2, \ldots, x_w]$, y al ideal generado por estos polinomios, $I(F)$. Recordando la definición 2.26, vamos a considerar el ideal de eliminación de $I(F)$ respecto de x_1, es decir, $I(F) \cap \mathbb{F}_2[x_1]$. Este es un ideal en una sola variable y como tal, está generado por un único polinomio al que llamaremos g_1.

Ahora pensemos en las soluciones de este sistema. Estas son w–uplas cuyas coordenadas denotaran las posiciones en las que ha habido errores. Definimos el conjunto de todas estas coordenadas como

$$E = \{\beta : (z_1, z_2, \ldots, \beta, \ldots, z_w) \text{ es una solución del sistema}\}$$

Con todo esto, estamos en condiciones de presentar el resultado más importante para el algoritmo de decodificación que vamos a presentar. La idea es que el polinomio $g_1(x_1)$ se anula en todos los elementos de E.

[2] Salvo el orden, ya que por la simetría del sistema se pueden permutar las posiciones de una solución y seguirá verificando las ecuaciones.

Proposición 4.1. Sea $g_1 \in \mathbb{F}_2[x_1]$ el generador mónico del ideal $I(F) \cap \mathbb{F}_2[x_1]$ y E el conjunto de las coordenadas de todas las soluciones de (4.1), entonces

$$g_1(x_1) = \prod_{\beta \in E} (x_1 + \beta)$$

La idea de la demostración es sencilla. Para cualquier elemento de E, existe un vector solución del sistema $z = (z_1, z_2, \ldots, \beta, \ldots, z_w)$, pero por la simetría de los polinomios de F, el vector $(\beta, z_1, z_2, \ldots, z_w)$ también será solución del sistema, por lo que todos los elementos de E son la primera coordenada de algún vector solución, y por tanto son solución de los polinomios de $\mathbb{F}_2[x_1]$ del ideal.

Con esto llegamos al corolario en el que se basa el algoritmo que vamos a presentar.

Corolario 4.2. Si ha habido v errores, para un número de errores supuesto w menor que el número de errores que puede corregir el código t, el polinomio generador mónico de $I(F) \cap \mathbb{F}_2[x_1]$, $g_1(x_1)$ verifica que

i. $g_1(x_1) = 1$ para $v > w$.

ii. $g_1(x_1) = L(x_1)$ para $v = w$.

iii. $g_1(x_1) = x_1 L(x)$ para $v = w - 1$.

iv. $g_1(x_1) = x_1^{n+1} + x_1$ para $v \leq w - 2$.

Demostración:

Sabemos que todo polinomio de error de longitud menor o igual que t tiene asociado un conjunto de síndromes distinto. Teniendo esto en cuenta, llamaremos $z_1, z_2, \ldots, z_v$ a los elementos que indican la posición de los v errores.

i. | *Como el sistema de ecuaciones (4.1) tiene menos variables que errores ha habido, no existe ninguna combinación de w elementos que den lugar a estos síndromes, por lo que el sistema no tiene solución y por tanto $I(F) = I(F) \cap \mathbb{F}_2[x_1] =< 1 >$.*

ii. | *Como $v = w$, $E = \{z_1, z_2, \ldots, z_v\}$ y por la proposición 4.1 tenemos que $g_1(x_1) = L(x_1)$.*

iii. | *Como $v = w - 1 < t$, la única forma de obtener los síndromes del sistema con w variables es con los elementos $E = \{z_1, z_2, \ldots, z_v, 0\}$, y por tanto*

$$g_1(x_1) = \prod_{\beta \in E} (x_1 + \beta) = x_1 \prod_{i=1}^{v} (x_1 + z_i) = x_1 L(x_1)$$

iv. | *Para $v \leq w - 2$, sabemos que*

$$s_i = z_1 + z_2 + \cdots + z_v + y + y = z_1 + z_2 + \cdots + z_v$$

por lo que cualquier elemento y de $\mathbb{F}_{2^r}$ puede ser solución. Por tanto, $E = \mathbb{F}_{2^r}$ y

$$g_1(x_1) = \prod_{\beta \in E} (x_1 + \beta) = \prod_{b \in \mathbb{F}_{2^r}} (x_1 + b) = x_1^{n+1} + x_1$$

■

Lo que implica este corolario es de suma importancia. Con una suposición correcta sobre el número de errores, se definen una serie de ecuaciones a partir de unos síndromes fáciles de calcular; se calcula una Base de Gröbner de estos polinomios con un orden lexicográfico y el ideal de eliminación asociado a la variable más pequeña nos lleva directamente al polinomio localizador de errores. Es más, si la suposición no es correcta, sabremos si es por exceso o por defecto y podremos repetir el proceso con una mejor suposición.

Veamos el algoritmo con detalle.

Algoritmo de decodificación

Partimos de la suposición de que nos llega un mensaje $r(x)$ codificado con un código BCH cíclico binario de longitud n capaz de corregir t errores, construido a partir de un elemento $a \in \mathbb{F}_{2^r}$ de orden $n|2^r - 1$.

1. En primer lugar, calculamos los t síndromes

$$s_i = r(a^i) \qquad i = 1, 3, 5, \ldots, 2t-1$$

   Si todos estos síndromes son 0, la palabra no ha sufrido modificación y $c(x) = r(x)$. Fin del algoritmo. En caso contrario, ha habido errores y seguimos al siguiente paso.

2. Hacemos una suposición sobre el número de errores que puede haber en $r(x)$, digamos w. Esta suposición se puede hacer como el número esperado de errores, si conocemos la probabilidad de error del canal; suponiendo que ha habido el mínimo posible de errores ($w = 2$); o el máximo que puede corregir el código ($w = t$)[3].

3. Definimos el conjunto de $t + w$ polinomios F

$$F = \begin{cases} x_1 + x_2 + \cdots + x_w + s_1 \\ x_1^3 + x_2^3 + \cdots + x_w^3 + s_3 \\ \vdots \\ x_1^{2t-1} + x_2^{2t-1} + \cdots + x_w^{2t-1} + s_{2t-1} \\ \\ x_1^{n+1} + x_1 \\ x_2^{n+1} + x_2 \\ \vdots \\ x_w^{n+1} + x_w \end{cases}$$

4. Calculamos una Base de Gröbner G del ideal generado por F con un orden puramente lexicográfico.

5. Calculamos el ideal de eliminación respecto de x_1 (la variable más pequeña), sin más que elegir el único polinomio de G que solo contenga la variable x_1. A este polinomio lo llamamos $g_1(x)$.

[3] Esta última a priori no es recomendable ya que el paso 4. con w más pequeñas es mucho más sencillo computacionalmente.

6. Según el valor de $g_1(x)$ distinguimos varios casos:
   - Si $g_1(x) = 1$, hemos supuesto menos errores de los reales. Cambiamos $w = w + 2$ y volvemos al paso 3.
   - Si $g_1(x) = x^{n+1} + x$, hemos supuesto más errores de los reales. Cambiamos $w = w - 2$ y volvemos al paso 3.
   - Si no se da el caso anterior, pero $g_1(0) = 0$, calculamos el polinomio localizador de error como $L(x) = \frac{g_1(x)}{x}$.
   - Si no se da ninguno de los casos anteriores, $L(x) = g_1(x)$.
7. Con el polinomio localizador calculado se buscan sus v raíces (probando qué elementos de $\mathbb{F}_{2^r}$ anulan $L(x)$, por ejemplo), que serán de la forma a^{e_i}, donde $0 \leq e_i \leq n-1$ es la posición del i-ésimo error y $1 \leq i \leq v$.
8. El polinomio de error será $e(x) = \sum_{i=1}^{v} x^{e_i}$.
9. La palabra del código que se envió fue $c(x) = r(x) + e(x)$.
10. El mensaje original es el polinomio $m(x) = \frac{c(x)}{g(x)}$.

Para programar este algoritmo, lo recomendable es elegir el código BCH concreto que va a decodificar, puesto que hacer un algoritmo genérico sería ineficiente: habría que especificar en los parámetros las características del código cada vez que se use.

En la siguiente figura podemos el ver un programa para decodificar el código del ejemplo 3.38, pero está escrito de forma que sea fácil de modificar para adaptarlo a otros códigos cíclicos binarios.

```
DecodBCH[r_] := Module[{g, t, n, F16, a, Elem, s, i, w, F1, F2, F, G, RG, IE, L, pe, e, c, m},
                        módulo
  (* --------Aspectos generales del código ---------------------------------------*)
  g = x^10 + x^8 + x^5 + x^4 + x^2 + x + 1; (* Polinomio generador del código *)
  t = 3 ; (*Número de errores que puede corregir el código *)
  n = 15; (* Longitud del código *)
  F16 = GF[2, {1, 1, 0, 0, 1}]; (* Cuerpo finito sobre el que trabajamos*)
  a = F16[{0, 1}]; (* Elemento del cuerpo de orden n *)
  Elem = Table[a^i, {i, 1, 15}];
         tabla

  (* -------Cálculo de síndromes y construcción del sistema ----------------------*)
  s = Table[r /. x → a^i, {i, 1, 2 t - 1, 2}]; (* Calculamos los síndromes *)
      tabla
  If[s == ConstantArray[0, t], Return[{0, r, PolynomialMod[PolynomialQuotient[r, g, x], 2]}]];
  si      arreglo constante      retorna    función mod poli···  cociente de polinomios
  w = 2; (*Suposición inicial*)
  Label[wrongw]; (*Si la suposición es incorrecta, habrá que volver aquí *)
  etiqueta
  F1 = Table[Sum[x_j^(2 * i - 1), {j, 1, w}] + s[[i]], {i, t, 1, -1}];
       tabla  suma
  F2 = Table[x_j^16 + x_j, {j, 1, w}];
       tabla
  F = Expand[Join[F2, F1] * F16[{1}]] ; (* Se multiplican todos los polinomios del sistema
      expan···  junta
   por el 1 del cuerpo finito para que todos los coeficientes se consideren en el cuerpo*)
  (* ------Base de Gröbner e Ideal de Eliminación ---------------------------------*)
  G = BaseGroebner[F];
  RG = Reduccion[G]; (*Base de Gröbner Reducida *)
  IE = IdealEliminacion[RG, Join[{x_w}, Elem]]; (* Añadimos los elementos del
                            junta
   cuerpo finito porque el programa los considera variables *)
  L = IE[[1]]; (*Posible polinomio localizador *)

  If[SubsetQ[Variables[L], {x_w}], , w++; Goto[wrongw]]; (* Si x_w no aparece en el
  si  ¿subcon···  variables                 ve a
   polinomio localizador es porque éste es trivial y hemos supuesto menos errores que
   los reales. Aumentamos w y volvemos a empezar. *)

  (*------ Cálculo de las posiciones de error ------------------------------------*)
  pe = {}; (*Posiciones de error*)
  For[i = 0, i ≤ 14, i++,
  para cada
   La = L /. x_w → a^i; (* L(a^i) *)
   If[La == 0,
    si
     AppendTo[pe, i];
     añade al final
   ];
  ];

  (* ------- Recuperar el error, la palabra del código y el mensaje ---------------*)
  e = Sum[x^pe[[i]], {i, Length[pe]}];
      suma              longitud
  c = PolynomialMod[r + e, 2];
      función mod polinómica
  m = PolynomialMod[PolynomialQuotient[c, g, x], 2];
      función mod poli···  cociente de polinomios

  Return[{e, c, m}]
  retorna
]
```

Figura 4.1: Código fuente de la función `DecodBCH[r]`.

5 Conclusión

En este trabajo hemos estudiado dos ramas de las matemáticas especialmente relacionadas con la informática, las **Bases de Gröbner** y la **Teoría de Códigos**, desarrollando desde los aspectos más básicos de ambas teorías hasta un algoritmo que unifica ambas, decodificando códigos cíclicos binarios por medio de bases de Gröbner.

En el capítulo 2 hemos estudiado las **bases de Gröbner**, desde la definición de la reducción de polinomios hasta el algoritmo que desarrolló Bruno Buchberger [2] para calcular las bases. Aunque supera el ámbito de este trabajo, desde entonces se han desarrollado muchos otros algoritmos que mejoran la eficiencia, como el desarrollado por Makarim y Stevens [6], e incluso algunos para contextos específicos como el de Hinkelmann y Arnold para polinomios con coeficientes binarios [5]. El algoritmo que hemos programado en este trabajo para calcular las bases de Gröbner se puede encontrar en la figura 2.7.

En el capítulo 3 se ha desarrollado la **teoría de Códigos**, desde el mismo concepto de la codificación y la definición de un código, hasta el proceso por el cual podemos definir un código BCH cíclico capaz de corregir tantos errores como queramos. De igual forma, aunque no se estudien en este texto, existen muchas otras familias de códigos, como los códigos Reed-Solomon [8, Sección 9.6], y se sigue investigando de forma activa en este campo buscando nuevos enfoques y algoritmos para mejorar las comunicaciones, como es el caso del artículo escrito por Ouahada y Ferreira [9].

Por último, en el capítulo 4 hemos estudiado y programado en el software *Mathematica* 4.1 un algoritmo de decodificación de códigos BCH cíclicos binarios con bases de Gröbner propuesto por Chen, Reed, Helleseth y Truong [3]. Más concretamente, se ha programado un algoritmo que decodifica el código BCH de longitud 15 y capaz de corregir 3 errores que se ha estudiado en el ejemplo 3.38. Sirva este programa como un ejemplo para diseñar algoritmos para códigos de mayor longitud y dimensión, pero en ningún caso como el algoritmo más eficiente para decodificar este código concreto. El hecho de tener una dimensión tan pequeña hace que este código se pueda decodificar con un algoritmo de fuerza bruta con mejores resultados, como el que se presenta en la figura 5.1.

Igualmente relacionado con la eficiencia, cabe destacar las limitaciones que hemos tenido en este trabajo. El algoritmo de decodificación que hemos programado funciona bien, y en un tiempo razonable, para corregir dos errores. Sin embargo, la complejidad que supone un tercer error, sumado a la forma de trabajar con cuerpos finitos que utiliza *Mathematica*, hace que el tiempo de ejecución para corregir tres errores sea inasumible. En futuros trabajos en esta línea habría que plantearse utilizar otro software de programación, o intentar diseñar una forma más eficiente de trabajar con cuerpos de Galois.

```
BCH153[r_] := Module[{m, c, g, dif, l, min, pos},
                     módulo
  g = 1 + x + x^2 + x^4 + x^5 + x^8 + x^10;
  m = Table[IntegerDigits[i, 2], {i, 0, 15}];
      tabla   dígitos de entero
  c = Table[PolynomialMod[Sum[m[[i]][[j]] * x^(j - 1), {j, 1, Length[m[[i]]]}] * g, 2], {i, 1, 16}];
      tabla   función mod poli···  suma                                              longitud
  dif = Table[PolynomialMod[r - c[[i]], 2], {i, Length[c]}];
        tabla   función mod polinómica                    longitud
  l = Table[Length[dif[[i]]], {i, Length[dif]}];
      tabla   longitud                  longitud
  min = Min[l];
        mínimo
  pos = {};
  For[i = 1, i ≤ Length[l], i++,
  para cada        longitud
   If[min == l[[i]], AppendTo[pos, i]];
   si                añade al final
  ];
  If[Length[pos] ≥ 2, Print["Hay dos palabras en el código a la misma distancia de ", r]];
  si  longitud              escribe
  If[min ≥ 4, Print["Ha habido más de 3 errores."]];
  si          escribe
  Return[c[[pos[[1]]]]]
  retorna

 ]
```

Figura 5.1: Código fuente de la función `BCH153[ r ]`.

Bibliografía

[1] T. Becker, V. Weispfenning, H. Kredel, *Gröbner bases: a computational approach to commutative algebra*, Springer-Verlag, 1993.

[2] B. Buchberger, *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal*, Ph.D. Thesis, Inst. University of Innsbruck, Innsbruck, Austria, 1965.

[3] X. Chen, I. S. Reed, T. Helleseth, T. K. Truong, *Use of Gröbner Bases to decode Binary Cyclic Codes up to the True Minimum Distance*, IEEE Trans. Inform. Theory, **40** (1994), no. 5, 1654-1661.

[4] R. Hill, *A First Course in Coding Theory*, Oxford University Press, 1997.

[5] F. Hinkelmann y E. Arnold, *Fast Gröbner Basis Computation for Boolean Polynomials*, CoRR, 2010.

[6] R. H. Makarim, M. Stevens, *M4GB: an efficient Gröbner-basis algorithm*, ISSAC'17—Proceedings of the 2017 ACM International Symposium on Symbolic and Algebraic Computation (2017), ACM, 293-300.

[7] F. J. Macwilliams, N. J. A. Sloane, *The Theory of Error-Correcting Codes*, North-Holland, 1997.

[8] R. J. McEliece, *The Theory of Information and Coding Second Edition*, Cambridge University Press, 2002.

[9] K. Ouahada, H. C. Ferreira, *New distance concept and graph theory approach for certain coding techniques design and analysis*, Commun. Appl. Ind. Math. **10** (2019), no. 1, 53–70.

[10] D. Slepian, *Some further theory of group codes*, The Bell System Technical Journal, **39** (1960), no. 5, 1219 - 1252.

[11] http://reference.wolfram.com/language/FiniteFields/tutorial/FiniteFields.html