UNIVERSIDAD
DE GRANADA

Facultad de Ciencias y Escuela Técnica Superior de Ingeniería Informática y de Telecomunicación

Doble Grado en Ingeniería Informática y Matemáticas

TRABAJO DE FIN DE GRADO

Códigos convolucionales cíclicos sesgados. Algoritmo de Sugiyama

Presentado por:
Alejandro Cárdenas Barranco

Tutor:
Gabriel Navarro Garulo
Departamento de Álgebra

Curso académico 2023-2024

Códigos convolucionales cíclicos sesgados. Algoritmo de Sugiyama

Alejandro Cárdenas Barranco

Alejandro Cárdenas Barranco *Códigos convolucionales cíclicos sesgados. Algoritmo de Sugiyama*.

Trabajo de fin de Grado. Curso académico 2023-2024.

Responsable de tutorización

Gabriel Navarro Garulo
Departamento de Álgebra

Doble Grado en Ingeniería Informática y Matemáticas

Facultad de Ciencias y Escuela Técnica Superior de Ingeniería Informática y de Telecomunicación

Universidad de Granada

Declaración de originalidad

D. Alejandro Cárdenas Barranco

Declara explícitamente que el trabajo presentado como Trabajo de Fin de Grado (TFG), correspondiente al curso académico 2023-2024, es original, en el sentido de que no ha utilizado para la elaboración del trabajo fuentes sin citarlas debidamente.

En Granada a 14 de junio de 2024

Fdo: Alejandro Cárdenas Barranco

Agradecimientos

Me gustaría agradecer a mi tutor, Gabriel Navarro, por su tiempo dedicado y por sus consejos y explicaciones a lo largo de este trabajo.

También agradecer a mis padres todo el esfuerzo realizado, tanto personal como económico, para que yo pudiera estudiar en Granada. También por su apoyo y cariño durante estos años.

Quisiera mencionar también a mis compañeros Álvaro, Jesús, Juanma, Mónica y Manu, por todas esas horas juntos, tanto en clase como fuera, así como por toda la ayuda brindada.

Asimismo, me gustaría agradecer a Jose su apoyo incondicional desde que llegué a Granada y por haberme acompañado durante todo este tiempo.

Por último, agradecer a la universidad, en especial a todos mis profesores, por haberme brindado una de las mejores formaciones que se pueden tener, tanto en matemáticas como en ingeniería informática.

Resumen

Los códigos son estructuras algebraicas que permiten transmitir información de manera que el receptor tenga la capacidad de corregir los errores que se produzcan durante la transmisión de un mensaje. En este trabajo, exploraremos los códigos convolucionales cíclicos sesgados, una clase de códigos muy interesante por su estructura algebraica, que facilita el diseño de algoritmos de decodificación eficientes. Además, presentaremos e implementaremos el algoritmo de Sugiyama para códigos convolucionales cíclicos sesgados utilizando el software SageMath. Para alcanzar estos objetivos, abordaremos varios conceptos fundamentales: comenzaremos con nociones básicas de álgebra, incluyendo teoría de anillos, cuerpos y módulos. Luego, estudiaremos los códigos de bloque, enfocándonos en los códigos lineales y cíclicos. Posteriormente, examinaremos los códigos convolucionales y los anillos de polinomios sesgados, que constituyen la base de los códigos convolucionales cíclicos sesgados. Una vez explicados los códigos convolucionales sesgados, estaremos preparados para estudiar e implementar el algoritmo de Sugiyama para estos códigos.

Palabras clave teoría de códigos, códigos de bloque, códigos cíclicos, códigos convolucionales, SageMath, algoritmo de Sugiyama.

Summary

The primary objective of this project is to explore skew cyclic convolutional codes, a subclass of convolutional codes notable for their algebraic structure, which facilitates the design of efficient decoding algorithms. Additionally, we will present and implement a Sugiyama-like algorithm for decoding these codes using SageMath software.

CHAPTER 1

In this chapter, we will introduce the mathematical tools necessary for developing the coding theory addressed in this work. We will start by presenting the theory of rings and ideals, discussing some of their most important properties. Next, we will study finite fields, which are essential for the construction of codes. We will also cover the field of rational functions, which will be fundamental for constructing convolutional codes. Finally, we will provide a brief introduction to module theory, including basic definitions and properties of modules, as well as some important results about submodules and free modules.

CHAPTER 2

In the second chapter, we will cover some of the fundamentals of block coding theory, focusing primarily on linear codes. We will begin with the most basic definition of a code, i.e., a subset of a finite field. Next, we will define the concept of a linear code, adding a vector space structure to codes that will allow us to use more powerful tools from linear algebra. We will delve into the properties of these codes, including their generator and parity-check matrices, which allow us to uniquely define linear codes. Additionally, we will explore Hamming weight and Hamming distance, essential metrics for measuring the error-correcting performance of a code. Then, we will study a subclass of linear codes known as cyclic codes, which possesses a cyclic structure that allows more efficient encoding and decoding algorithms. Finally, we will discuss BCH and Reed-Solomon codes, known for their strong error-correcting abilities, and the Sugiyama algorithm used for decoding BCH codes.

CHAPTER 3

In this chapter, we delve into convolutional codes, a class of codes different from block codes, characterized by their use of previous message information in encoding, giving them a "memory". We will discuss the mathematical foundations needed, such as the theory of generator matrices, and explain how convolutional codes are defined using the field of rational functions. We also present the encoding process using polynomial matrices. The chapter further examines the concept of canonical generator matrices, their properties, and the criteria for a matrix to be considered canonical. Then, we link convolutional codes to module theory, showing how they can be viewed as submodules of free modules. Finally, we will examine the free distance of convolutional codes, analyzing its importance in determining the error-detecting and correcting capabilities of these codes, while also exploring the generalized Singleton bound, which offers an upper limit for the free distance.

CHAPTER 4

The fourth chapter focuses on the study of skew cyclic convolutional codes (SCCC). We will begin by introducing Ore polynomial rings, which are crucial for defining these codes, particularly focusing on the case where the coefficients of the skew polynomial ring are in a field. We will introduce the main concepts, as well as algorithms to calculate division from the left or right, and an extended Euclid algorithm, which will be fundamental for the construction of a Sugiyama-like algorithm. Then, we will define skew cyclic convolutional codes and provide a method for their construction. Finally, we will present skew Reed-Solomon convolutional codes, detailing their construction and proving their maximum distance separable property with respect to the Hamming distance, which ensures their optimal error-correcting capabilities.

CHAPTER 5

Finally, we will introduce the algorithm that serves as the main objective of this project: the Sugiyama algorithm for skew Reed-Solomon convolutional codes. We will provide a detailed explanation of the decoding process, showing how this algorithm effectively corrects errors in transmitted data, ensuring the accuracy and reliability of communication systems.

To carry out this project, the following classes have been implemented in SageMath:

- `BCHSugiyamaDecoder`: A decoder class implementing the Sugiyama algorithm for BCH codes, calculating syndromes and error correction.
- `SkewRSConvolutionalCode`: A class defining skew Reed-Solomon convolutional codes, including generator polynomial validation and properties.
- `SkewRSConvolutionalEncoder`: An encoder class for Skew Reed-Solomon convolutional odes, handling the encoding and unencoding processes.
- `SkewRSSugiyamaDecoder`: A decoder class using a Sugiyama-like algorithm to decode skew Reed-Solomon convolutional codes, correcting errors through syndrome computation and solving the key equation.

To ensure the correct functioning of these classes, a series of tests have been developed using the Pytest tool. For the examples in this project, both the pre-existing SageMath classes and the newly implemented classes have been used.

The documentation for these classes can be found in Appendix A.

KEYWORDS coding theory, block codes, cyclic codes, convolutional codes, SageMath, Sugiyama algorithm.

Índice general

Introducción

CONTEXTO

> *"El problema principal de la comunicación es reproducir en un punto, ya sea de forma exacta o aproximada, un mensaje en otro punto."*

Con estas palabras comenzada Claude Shannon en 1948 el artículo "A mathematical theory of comunication" [34], un trabajo que supuso el inicio de dos disciplinas imprescindibles en la actualidad: la teoría de códigos y la teoría de la información. En este artículo Shannon demostró que, mediante una *codificación apropiada de la información*, los errores inducidos por un canal o medio ruidoso pueden *reducirse a cualquier nivel deseado* sin empeorar la tasa de transmisión. El trabajo de Shannon proporcionó un marco teórico sobre la transmisión eficiente y fiable de la información.

Aunque el primer trabajo publicado de teoría de códigos fue el de Shannon, el verdadero padre de esta teoría es Richard W. Hamming, a quien Shannon nombró en su artículo. En 1950[1], motivado por la necesidad de corregir los errores producidos durante las operaciones con grandes números, Hamming publicó el artículo "Error detecting and error correcting codes" [14]. Hamming introdujo un tipo de códigos llamados *códigos de Hamming*, los cuales permiten detectar hasta errores de dos bits y corregir errores de un bit.

La teoría de códigos se encarga del diseño de códigos para la transmisión y el almacenamiento de los datos. Su principal objetivo es garantizar que la información pueda ser transmitida o almacenada de manera eficiente y con la menor cantidad posible de errores, aun cuando esta se envía a través de canales con ruido.

Supongamos que deseamos transmitir un *mensaje*. En este caso, habrá un *emisor* y un *receptor* que se comunican, generalmente, de manera unidireccional. El mensaje consiste en una secuencia finita de símbolos de un alfabeto específico y se envía a través de un *canal de comunicación*. Durante la transmisión, la información puede distorsionarse debido a interferencias o ruido presentes en el canal. Para mitigar este problema, es importante representar el mensaje en un formato adecuado para el canal, a la representación de este mensaje se le conoce como *palabra código* y a este proceso de transformación de un mensaje en su palabra código correspondiente se le conoce como *codificación*. Este proceso implica proteger el mensaje original mediante técnicas como la adición de redundancia, la repetición del mensaje o la incorporación de bits de paridad. Una vez se ha codificado el mensaje, se envía a través de un *canal*, donde puede sufrir alteraciones debido a la presencia de una *fuente de ruido*. Al llegar el mensaje al *receptor*, este recibe una versión posiblemente alterada de la palabra código original. Es entonces cuando se da el proceso conocido como *decodificación*, que consiste en detectar y corregir los errores y transformar la palabra código corregida en el formato original.

Todo este proceso de transmisión está recogido en la Figura 1.

[1] Aunque el artículo de Hamming fue publicado en 1950, realmente es un trabajo anterior al de Shannon.

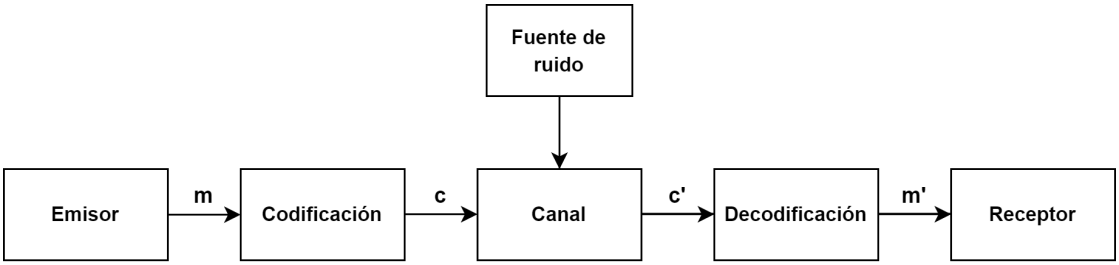

Figura 1.: Esquema del modelo de comunicación [30].

En general, el mensaje enviado y el mensaje recibido serán distintos. De esta forma, la teoría de códigos se encarga principalmente de los pasos de codificación y decodificación del diagrama anterior.

Algunas de las aplicaciones de la teoría de códigos son:

- *Comunicaciones digitales*: Se emplea en la telefonía móvil y redes de datos para asegurar la calidad y fiabilidad de las transmisiones.
- *Almacenamiento de datos*: También es esencial el uso de códigos para garantizar la integridad de la información en discos duros, SSDs y medios ópticos como CDs y DVDs.
- *Transmisiones por satélite*: Son imprescindibles también en todo tipo de transmisiones por satélite (televisión, GPS...), ya que protege de posibles interferencias atmosféricas.
- *Criptografía*: Otro de sus usos más populares (y con el que a veces se confunde), es la criptografía. Por ejemplo, se utilizan en sistemas de cifrado asimétrico, como puede ser el criptosistema McEliece, que usa códigos de Goopa.

ENFOQUE

El objetivo principal de este trabajo es estudiar los códigos convolucionales cíclicos sesgados (SCCC) e implementar el algoritmo de Sugiyama para la decodificación de estos códigos. En consecuencia, el trabajo se enfocará en introducir las herramientas necesarias para poder abordar este objetivo. En cuanto a los fundamentos matemáticos necesarios, estudiaremos los anillos, los cuerpos finitos, los módulos y los anillos de polinomios sesgados. Para la teoría de códigos, veremos nociones básicas de los códigos de bloque, centrándonos en los códigos lineales y cíclicos. También hablaremos de códigos convolucionales, otra clase de códigos distinta a los de bloque, los cuales serán esenciales para la definición del objeto principal de este trabajo, los códigos convolucionales cíclicos sesgados.

ESTRUCTURA DEL TRABAJO

Este trabajo consta de cinco capítulos y dos apéndices. En el primer capítulo, introduciremos los fundamentos matemáticos necesarios para desarrollar la teoría de códigos que se pretende abordar en este trabajo. En el segundo capítulo estudiaremos algunos de los fundamentos de

los códigos de bloque, enfocándonos principalmente en los códigos lineales y cíclicos, presentando también el algoritmo de Sugiyama para la decodificación de códigos BCH, una subclase de los códigos cíclicos. En el tercer capítulo profundizaremos en los códigos convolucionales, explorando sus bases matemáticas y algunas de sus propiedades fundamentales. En el cuarto capítulo, veremos en primer lugar los polinomios de Ore, una estructura matemática que constituye la base de los códigos convolucionales cíclicos sesgados, que se introducen a continuación. Finalmente, en el quinto capítulo describimos el algoritmo de Sugiyama para la decodificación de códigos convolucionales sesgados Reed-Solomon.

En el primer apéndice, se detallará el código implementado en el trabajo, así como la estructura de este y los test realizados para probar su funcionamiento. En el segundo apéndice, se expondrá la planificación temporal del proyecto y una estimación de los costes del mismo.

OBJETIVOS PRINCIPALES DEL TRABAJO

En cuanto al ámbito de las *Matemáticas*, los objetivos de este trabajo son:

- Estudiar las nociones básicas de la teoría de códigos lineales.
- Explorar los códigos convolucionales y su estructura algebraica, así como algunas de sus propiedades fundamentales.
- Investigar la noción de ciclicidad para los códigos convolucionales mediante los polinomios de Ore.
- Estudiar los códigos convolucionales sesgados Reed-Solomon.
- Exponer el algoritmo de Sugiyama para códigos convolucionales sesgados Reed-Solomon.

En cuanto a la *Ingeniería Informática*, observemos que los objetivos anteriores se enmarcan en el campo de la Computación teórica. Además, otros objetivos más específicos son:

- Implementar el algoritmo de Sugiyama para la decodificación de códigos BCH.
- Implementar un sistema de construcción de códigos convolucionales sesgados Reed-Solomon.
- Implementar un sistema de codificación de códigos convolucionales sesgados Reed-Solomon.
- Implementar el algoritmo de Sugiyama para la decodificación de códigos convolucionales sesgados Reed-Solomon.

PRINCIPALES FUENTES CONSULTADAS

Entre todas las fuentes bibliográficas consultadas, sobresalen las siguientes:

Para el estudio de las herramientas matemáticas del primer capítulo, [20] y [15] han resultado de utilidad.

En el segundo capítulo, [15] y [1] han sido fundamentales para el desarrollo de la teoría de códigos de bloque.

Para el capítulo tres, dedicado a los códigos convolucionales, las fuentes principales han sido [15], [7], [19], [9] y [25].

En el capítulo cuatro, centrado en los códigos convolucionales cíclicos sesgados (SCCC), se han utilizado como referencias principales [8] y [18] para los anillos de polinomios sesgados, además de [9], [13] y [11] para los códigos convolucionales cíclicos sesgados.

Finalmente, el capítulo cinco, que aborda el algoritmo de decodificación Sugiyama para los códigos convolucionales sesgados Reed-Solomon, se basa en los resultados descritos en [11].

Para el desarrollo de la implementación y pruebas, ha sido relevante la documentación de SageMath [36] y Pytest [21].

1. Preliminares

En este primer capítulo se introducirán las herramientas matemáticas necesarias para el desarrollo de la teoría de códigos que se pretende abordar en este trabajo. Empezaremos con una introducción a la teoría de anillos e ideales, comentando algunas de sus propiedades más importantes. Después, estudiaremos los cuerpos finitos, los cuales son esenciales para la construcción de códigos. También estudiaremos los cuerpos de fracciones, que serán fundamentales para la construcción de códigos convolucionales. Por último, se dará una breve introducción a la teoría de módulos.

Muchos de estos resultados son muy conocidos y en la mayoría de los casos se omitirán las demostraciones de los mismos. Las principales fuentes consultadas para el desarrollo de este capítulo han sido [20]y [15].

1.1. Anillos

Definición 1.1. Un *anillo* $(\mathcal{A}, +, \cdot)$ es un conjunto $\mathcal{A}$ con dos operaciones binarias $\mathcal{A} \times \mathcal{A} \to \mathcal{A}$ llamadas suma $(+)$ y producto $(\cdot)$, que verifican las siguientes propiedades:

1. Propiedad asociativa de la suma:

$$a + (b + c) = (a + b) + c \quad \forall a, b, c \in \mathcal{A}.$$

2. Existencia de un elemento neutro de la suma:

$$\exists 0 \in A : 0 + a = a + 0 = a \quad \forall a \in \mathcal{A}.$$

3. Existencia de un elemento inverso para la suma:

$$\forall a \in \mathcal{A} \; \exists - a \in \mathcal{A} : a + (-a) = (-a) + a = 0.$$

4. Propiedad conmutativa de la suma:

$$a + b = b + a \quad \forall a, b \in \mathcal{A}.$$

5. Propiedad asociativa del producto:

$$a \cdot (b \cdot c) = (a \cdot b) \cdot c \quad \forall a, b, c \in \mathcal{A}.$$

6. Propiedad distributiva del producto:

$$a \cdot (b + c) = a \cdot b + a \cdot c, \quad (a + b) \cdot c = a \cdot c + b \cdot c \quad \forall a, b, c \in \mathcal{A}.$$

Diremos que un anillo es *conmutativo* si se verifica:

$$a \cdot b = b \cdot a \quad \forall a, b \in \mathcal{A}.$$

Un tipo de anillo destacado es el siguiente.

Definición 1.2. Un anillo $\mathcal{A}$ se dice *unitario* si existe un elemento $1 \in \mathcal{A}$ tal que $a \cdot 1 = 1 \cdot a = a \quad \forall a \in \mathcal{A}$. Lo denotaremos *elemento unidad*.

El conjunto de los elementos unidad de un anillo forman un *grupo* con respecto a la operación producto. A este grupo lo llamaremos *grupo de unidades* de $\mathcal{A}$ y se denotará por $\mathcal{U}(\mathcal{A})$.

Ejemplo 1.3. Algunos ejemplos de anillos son los siguientes.

- El conjunto de los números enteros $\mathbb{Z}$ es un anillo conmutativo y unitario.
- Los enteros de Gauss $\mathbb{Z}[i] = \{a + bi : a, b \in \mathbb{Z}\}$ forman también un anillo conmutativo y unitario.
- El conjunto de las matrices cuadradas con coeficientes reales $\mathcal{M}_n(\mathbb{R})$ forman un anillo con las operaciones de la suma y el producto de matrices. Tiene elemento unidad (la matriz identidad) y es no conmutativo.

Dado un anillo $\mathcal{A}$ unitario podría darse que el elemento neutro y el elemento unidad coincidan. En ese caso, el anillo $\mathcal{A}$ tendrá un único elemento y diremos que el anillo es *trivial*.

Definición 1.4. Dado un elemento $a \in \mathcal{A}$ de un anillo unitario, diremos que es *unidad* si existe otro elemento $a^{-1} \in \mathcal{A}$ de forma que $a \cdot a^{-1} = a^{-1} \cdot a = 1$. En tal caso, a^{-1} es único y lo llamaremos *elemento inverso* de a.

Claramente, el 0 no puede tener elemento inverso, pues cualquier otro elemento que multipliquemos por él, nos dará como resultado 0.

Definición 1.5. Diremos que $\mathcal{A}$ es un *anillo de división* si todo elemento $a \in \mathcal{A}$ distinto de 0 tiene inverso.

Definición 1.6. Los *divisores de cero* en un anillo $\mathcal{A}$ son los elementos $a, b \in \mathcal{A}$ tales que $a \cdot b = 0$.

Los anillos conmutativos con $1 \neq 0$ que no contengan divisores de cero se denominarán *dominios de integridad* (DI).

En un dominio de integridad se da la propiedad de *cancelación*.

Proposición 1.7. *Sea $\mathcal{A}$ un dominio de integridad, entonces si $a \neq 0$ y $ax = ay$, se verifica $x = y$.*

Demostración. $ax = ay \Rightarrow a(x-y) = 0$, por ser $\mathcal{A}$ dominio de integridad y $a \neq 0$ se tiene que $x - y = 0 \Rightarrow x = y$. □

Definición 1.8. Sea $\mathcal{A}$ un anillo unitario. Se define su *característica* como el entero positivo más pequeño n tal que $1 + \overset{n)}{\ldots} + 1 = 0$. Si no existe tal n se dice que la característica de $\mathcal{A}$ es 0.

Definición 1.9. Un *subanillo* $\mathcal{B}$ de $\mathcal{A}$ es un subconjunto $\mathcal{B} \subset \mathcal{A}$ que es anillo con las operaciones heredadas de $\mathcal{A}$.

Definición 1.10. Dados $\mathcal{A}, \mathcal{B}$ dos anillos unitarios, una aplicación $f : \mathcal{A} \to B$ se dice *homomorfismo* de anillos si $\forall a, b \in \mathcal{A}$ se tiene:

$$f(a+b) = f(a) + f(b), f(ab) = f(a)f(b) \text{ y } f(1_{\mathcal{A}}) = 1_{\mathcal{B}}.$$

Definición 1.11. Un subconjunto I no vacío de un anillo $\mathcal{A}$ es un *ideal por la izquierda* si:

1. I es un subgrupo aditivo de $\mathcal{A}$.
2. El producto por la izquierda de un elemento de I por otro elemento de $\mathcal{A}$ está en I, es decir, $a \cdot x \in I, \forall (a, x) \in \mathcal{A} \times I$.

Y es un *ideal por la derecha si*:

1. I es un subgrupo aditivo de $\mathcal{A}$.
2. El producto por la derecha de un elemento de I por otro elemento de $\mathcal{A}$ está en I, es decir, $x \cdot a \in I, \forall (a, x) \in \mathcal{A} \times I$.

Un *ideal bilátero* es un ideal por la derecha y por la izquierda. En un anillo conmutativo, las nociones de ideal por la derecha, de ideal por la izquierda y de ideal bilátero coinciden y simplemente hablaremos de ideal.

Definición 1.12. Diremos que un anillo $\mathcal{A}$ es simple si no contiene ideales biláteros distintos de $\mathcal{A}$ y $\{0\}$.

Ejemplo 1.13. Algunos ejemplos de ideales son los siguientes.

- $\{0\}$ y $\mathcal{A}$ son ideales de $\mathcal{A}$. De hecho, si $1 \in I$, entonces $\mathcal{A} = I$.
- Sea $n\mathbb{Z} = \{n \cdot a : a \in \mathbb{Z}\}, n \in \mathbb{N}$, el conjunto de los múltiplos de n. Entonces $n\mathbb{Z}$ es un ideal, pues todo número entero multiplicado por n es un múltiplo suyo.

Dado un ideal bilátero I de un anillo $\mathcal{A}$, es posible definir una relación de equivalencia. Sean $a, b \in \mathcal{A}$, la relación $a\mathcal{R}b \Leftrightarrow a - b \in I$ es de equivalencia. Esta relación nos permite definir el conjunto cociente A/I, de las clases de equivalencia obtenidas a través de ella. Dado un elemento $a \in I$, su clase de equivalencia viene dada por:

$$[a] = a + I = \{a + x : x \in I\}.$$

Las propiedades de ideal hacen que el conjunto cociente A/I sea un anillo, al que llamaremos *anillo cociente*, con las siguientes operaciones (que están bien definidas):

- Suma: $(a+I)+(b+I)=(a+b)+I$.
- Producto $(a+I)\cdot(b+I)=(ab)+I$.

Un ideal puede ser generado por un *subconjunto*. Sea $\emptyset \neq S \subset \mathcal{A}$ un subconjunto, entonces el ideal generado por S en $\mathcal{A}$ es:

$$I(S)=\{x_1h_1+\cdots x_rh_r : h_i \in I, x_i \in \mathcal{A}\} \Leftrightarrow I(S)=\bigcap_{I_i \supset S} I_i.\ I_i \text{ ideal de } \mathcal{A}.$$

Es decir, $I(S)$ es el menor ideal de $\mathcal{A}$ que contiene a S.

- *Ideal principal:* El ideal es generado por un elemento, $bA=(b)=\{ab : a \in \mathcal{A}\}$.
- *Ideal finitamente generado:* El ideal es generado por un número finito de elementos, $S=\{b_1,\ldots,b_n\}$:
$$I(S)=(b_1,\ldots,b_n)\mathcal{A}=\{x_1b_1+\cdots+x_nb_n : x_i \in \mathcal{A}\}.$$

Definición 1.14. Llamaremos *dominio de ideales principales* (DIP) a un dominio de integridad en el que todos sus ideales son principales.

Teorema 1.15 (Forma Normal de Smith)**.** *Sea R un dominio de ideales principales y sea $A \in \mathcal{M}_{k\times n}(R)$ una matriz no nula con coeficientes en R. Entonces $A=CDB$, donde $B \in GL_n(R), C \in GL_k(R)$ y $D \in \mathcal{M}_{k\times n}(R)$, con*

$$D=\begin{pmatrix} \alpha_1 & 0 & \cdots & 0 & \cdots & 0 \\ 0 & \alpha_2 & \cdots & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & & \vdots \\ 0 & 0 & \cdots & \alpha_m & \cdots & 0 \\ \vdots & \vdots & & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & \cdots & 0 \end{pmatrix}$$

para algún $m \leq \min\{n,k\}$ y elementos $\alpha_1,\ldots,\alpha_m \in R$ no nulos tales que $\alpha_i \mid \alpha_{i+1}$, $\forall i \in \{1,\ldots,m-1\}$.

La matriz D del teorema anterior recibe el nombre de la *Forma Normal de Smith* de la matriz A.

1.2. Cuerpos

Definición 1.16. Un cuerpo $(K,+,\cdot)$ es un *anillo conmutativo* K cuyo grupo de unidades $\mathcal{U}(K)$ es $K\setminus\{0\}$. Diremos que un cuerpo es *finito* si tiene un número finito de elementos.

Observemos que estamos suponiendo que forzosamente un cuerpo nunca es el anillo trivial $\{0\}$. En general, los cuerpos finitos con q elementos los denotaremos por $\mathbb{F}_q$.

Definición 1.17. Sea F un subanillo de un cuerpo K, que es, a su vez, un cuerpo. Diremos que F es un *subcuerpo* de K. Se dice también que $F \leq K$ es una *extensión de cuerpos*.

El siguiente teorema recoge algunas de las propiedades más básicas e importantes de los cuerpos finitos.

Teorema 1.18. *Sea $\mathbb{F}_q$ un cuerpo finito con q elementos. Entonces se verifican las siguientes afirmaciones:*

(i) $|\mathbb{F}_q| = p^n$, *para algún número primo p y algún entero positivo n.*

(ii) $\mathbb{F}_q$ *contiene al subcuerpo* $\mathbb{F}_p$.

(iii) $\mathbb{F}_q$ *es un espacio vectorial sobre* $\mathbb{F}_p$ *de dimensión n y hay q^n vectores en el espacio vectorial de dimensión n sobre* $\mathbb{F}_p$.

(iv) $p\alpha = 0 \ \forall \alpha \in \mathbb{F}_q$.

(v) $\mathbb{F}_q$ *es único salvo isomorfismo.*

Proposición 1.19. *Sea $\mathbb{F}_q$ un cuerpo finito de característica p y $n \in \mathbb{N}$. Entonces:*

$$(a+b)^{p^n} = a^{p^n} + b^{p^n}, \quad \forall a, b \ \in \mathbb{F}_q.$$

$$(a-b)^{p^n} = a^{p^n} - b^{p^n}, \quad \forall a, b \ \in \mathbb{F}_q.$$

Demostración. Por el Teorema del binomio, se puede desarrollar la n-ésima potencia de un binomio. De esta forma tenemos que

$$(a+b)^{p^n} = \sum_{k=0}^{p^n} \binom{p^n}{k} a^{p^n-k} b^k = \binom{p^n}{0} x^{p^n} + \binom{p^n}{1} x^{p^n-1} y + \cdots + \binom{p^n}{p^n-1} x y^{p^n-1} + \binom{p^n}{p^n} y^{p^n}.$$

Por otro lado, sabemos que

$$\binom{p^n}{k} = \frac{p^n!}{k!(p^n-k)!}.$$

En consecuencia, $\binom{p^n}{k}$ es divisible por p para $k \in \{1, \ldots, p^n - 1\}$.

Entonces, todos los coeficientes de la ecuación anterior, salvo el primero y el último, son múltiplos de p, y, por tanto, en virtud del Teorema 1.18, su valor es 0.

Así,

$$(a+b)^{p^n} = a^{p^n} + b^{p^n}.$$

La segunda igualdad se prueba de forma análoga teniendo en cuenta que

$$(a-b)^{p^n} = \sum_{k=0}^{p^n} \binom{p^n}{k} (-1)^k a^{p^n-k} b^k.$$

□

1.2.1. Anillos de polinomios sobre cuerpos finitos

Definición 1.20. Sea $\mathcal{A}$ un anillo conmutativo y x un elemento que no pertenece a $\mathcal{A}$. Un polinomio con coeficientes en $\mathcal{A}$ es una expresión de la forma

$$a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0, \quad n \in \mathbb{N}, a_0, \ldots, a_n \in \mathcal{A}.$$

Dado un anillo $\mathcal{A}$ denotaremos por $\mathcal{A}[x]$ al conjunto de todos los polinomios con coeficientes en $\mathcal{A}$.

Definición 1.21. Sea $\mathcal{A}$ un anillo y sean $p(x) = a_m x^m + \cdots + a_1 x + a_0$ y $q(x) = b_n x^n + \cdots + b_1 x + b_0$ dos elementos de $\mathcal{A}[x]$. Supongamos que $m \leq n$.

1. Se define la suma de los polinomios $p(x)$ y $q(x)$ como el polinomio

$$p(x) + q(x) = b_n x^n + \cdots + b_{m+1} x^{m+1} + (a_m + b_m) x^m + \cdots + (a_1 + b_1) x + (a_0 + b_0).$$

2. Se define el producto de los polinomios $p(x)$ y $q(x)$ como el polinomio

$$p(x)q(x) = a_m b_n x^{m+n} + \cdots + (b_0 a_2 + b_1 a_1 + b_2 a_0) x^2 + (b_0 a_1 + b_1 a_0) x + a_0 b_0.$$

Se comprueba fácilmente que si $\mathcal{A}[x]$ es un anillo conmutativo, entonces $\mathcal{A}[x]$ *es también un anillo conmutativo* con las operaciones que acabamos de definir. Además, podemos identificar al anillo $\mathcal{A}$ como los elementos de $\mathcal{A}[x]$ de la forma $p(x) = a$, $a \in \mathcal{A}$. Por tanto, $\mathcal{A}$ es un subanillo de $\mathcal{A}[x]$.

Veamos ahora algunos conceptos referentes a polinomios.

Definición 1.22. Sea $\mathcal{A}$ un anillo conmutativo y $p(x) = a_n x^n + \cdots + a_1 x + a_0 \in \mathcal{A}$.

1. Si $a_n \neq 0$ se dice que el polinomio $p(x)$ tiene grado n y se representa como $\mathrm{gr}(p(x)) = n$. Cuando $p(x) = 0$, consideraremos que su grado es $-\infty$.
2. Al elemento $a_k \in \mathcal{A}$ se le llama *coeficiente de grado k*.
3. El coeficiente de grado n de un polinomio de grado n se llama *coeficiente líder*.
4. El coeficiente de grado 0 de un polinomio se llama *término independiente*.
5. Un polinomio cuyo coeficiente líder valga 1 se dice que es un *polinomio mónico*.

Proposición 1.23. *Sean $p(x), q(x) \in \mathcal{A}[x]$. Entonces:*

$$\begin{aligned} \mathrm{gr}(p(x) + q(x)) &\leq \max\{\mathrm{gr}(p(x)), \mathrm{gr}(q(x))\}, \\ \mathrm{gr}(p(x)q(x)) &\leq \mathrm{gr}(p(x)) + \mathrm{gr}(q(x)). \end{aligned}$$

Una vez dada una pequeña introducción a los anillos de polinomios en anillos arbitrarios, vamos a enfocarnos en el caso que nos interesa para el desarrollo de este trabajo, es decir, cuando el anillo es un *cuerpo finito*. El conjunto de polinomios con coeficientes en $\mathbb{F}_q$ se denota por $\mathbb{F}_q[x]$. Como ya se ha comentado en el caso general, este conjunto es un *anillo conmutativo* y, de hecho, es un *dominio de integridad*.

El anillo $\mathbb{F}_q[x]$ no solo juega un papel fundamental en la construcción de cuerpos finitos, sino también en la construcción de ciertas familias de códigos, como veremos a lo largo de este trabajo.

Sean $f(x)$ y $g(x)$ polinomios en $\mathbb{F}_q[x]$, diremos que $f(x)$ divide a $g(x)$ si existe un polinomio $h(x) \in \mathbb{F}_q[x]$ tal que $g(x) = f(x)h(x)$ y este hecho lo denotamos por $f(x) \mid g(x)$. El polinomio $f(x)$ se llama *divisor o factor* de $g(x)$.

El *máximo común divisor* de $f(x)$ y $g(x)$, suponiendo que al menos uno de ellos no es nulo, es el polinomio mónico en $\mathbb{F}_q[x]$ de mayor grado que divide a ambos. El máximo común divisor de dos polinomios es *único* y se denota como $\mathrm{mcd}(f(x), g(x))$. Diremos que $f(x)$ y $g(x)$ son *coprimos* si $\mathrm{mcd}(f(x), g(x)) = 1$.

El siguiente resultado nos dará las bases para definir posteriormente el algoritmo de Euclides.

Proposición 1.24. *Sean $f(x), g(x) \in \mathbb{F}_q[x]$ con $g(x) \neq 0$. Se verifican:*

(i) (Algoritmo de la división) Existen polinomios $h(x), r(x) \in \mathbb{F}_q[x]$ únicos tales que

$$f(x) = g(x)h(x) + r(x), \quad \mathrm{gr}(r(x)) < \mathrm{gr}(g(x)).$$

(ii) Si $f(x) = g(x)h(x) + r(x)$, entonces $\mathrm{mcd}(f(x), g(x)) = \mathrm{mcd}(g(x), r(x))$.

Utilizando este resultado de manera recursiva podemos hallar el máximo común divisor de dos polinomios. Este procedimiento se conoce como *algoritmo de Euclides* y es análogo a su versión para números enteros.

Teorema 1.25 (Algoritmo de Euclides)**.** *Sean $f(x), g(x) \in \mathbb{F}_q[x]$ con $g(x) \neq 0$.*

1. *Realizar los siguientes pasos hasta que $r_n(x) = 0$ para algún n:*

$$\begin{aligned}
f(x) &= g(x)h_1(x) + r_1(x), && \text{donde } \mathrm{gr}(r_1(x)) < \mathrm{gr}(g(x)),\\
g(x) &= r_1(x)h_2(x) + r_2(x), && \text{donde } \mathrm{gr}(r_2(x)) < \mathrm{gr}(r_1(x)),\\
r_1(x) &= r_2(x)h_3(x) + r_3(x), && \text{donde } \mathrm{gr}(r_3(x)) < \mathrm{gr}(r_2(x)),\\
&\vdots\\
r_{n-3}(x) &= r_{n-2}(x)h_{n-1}(x) + r_{n-1}(x), && \text{donde } \mathrm{gr}(r_{n-1}(x)) < \mathrm{gr}(r_{n-2}(x)),\\
r_{n-2}(x) &= r_{n-1}(x)h_n(x) + r_n(x), && \text{donde } r_n(x) = 0.
\end{aligned}$$

Entonces, $\mathrm{mcd}(f(x), g(x)) = cr_{n-1}(x)$, donde $c \in \mathbb{F}_q$ es una constante para que $cr_{n-1}(x)$ sea mónico.

2. *Existen polinomios* $a(x), b(x) \in \mathbb{F}_q[x]$ *tales que*

$$a(x)f(x) + b(x)g(x) = \text{mcd}(f(x), g(x)).$$

Puesto que en el paso 1 el grado se reduce en al menos una unidad, podemos asegurar que la secuencia de pasos anterior acaba en algún momento.

Proposición 1.26. *Sea* $\mathbb{F}_q$ *un cuerpo, entonces* $\mathbb{F}_q[x]$ *es un dominio de ideales principales.*

Demostración. Sea I un ideal de $\mathbb{F}_q[x]$. Si I es trivial entonces es principal ya que $\{0\} = 0\mathbb{F}[x]$ y $\mathbb{F}_q[x] = 1\mathbb{F}_q[x]$.

Supongamos que I es un ideal no trivial de $\mathbb{F}_q[x]$. Sea $p(x)$ el polinomio de menor grado en $I \setminus \{0\}$. Como I es un ideal, $p(x)\mathbb{F}_q[x] \subseteq I \subseteq \mathbb{F}_q[x]$.

Si $p(x)$ es un elemento p de $\mathbb{F}_q \setminus \{0\}$, entonces $p(x)\mathbb{F}_q[x] = \mathbb{F}_q[x] = I$. Por tanto, I es un ideal principal, pues está generado por un único elemento.

Si, por el contrario, $p(x) \notin \mathbb{F}_q$, sea $a(x) \in I$ un polinomio al que $p(x)$ no divide. Esto es, $p(x) \nmid a(x)$. Por el algoritmo de la División 1.24, existen dos polinomios $q(x), r(x) \in \mathbb{F}_q[x]$ tales que $a(x) = p(x)q(x) + r(x)$ y $\text{gr}(r(x)) < \text{gr}(p(x)), r(x) \neq 0$.

Sin embargo,

$$a(x), p(x) \in I \Rightarrow a(x), p(x)q(x) \in I \Rightarrow r = a(x) - p(x)q(x) \in I.$$

Pero esto no es posible, puesto que habíamos dicho que $p(x)$ era el polinomio de menor grado en $I - \{0\}$. Hemos llegado a una contradicción. Así, $p(x) \mid a(x)$, $\forall a(x) \in I$. Por tanto, $I = p(x)\mathbb{F}_q[x]$ e I es un ideal principal.

□

1.2.2. Cuerpo de fracciones

Los cuerpos de fracciones son un tipo particular de cuerpo, que cobrará importancia cuando estudiemos los códigos convolucionales.

Sea D un dominio de integridad y consideremos en $D \times (D\setminus\{0\})$ la relación $(a, b)\mathcal{R}(c, d) \Leftrightarrow ad - bc = 0$, que se comprueba trivialmente que es de equivalencia. Consideramos el conjunto cociente,

$$\frac{D \times (D\setminus\{0\})}{\mathcal{R}},$$

denotamos la clase del par (a, b) por a/b y definimos las operaciones

$$\frac{a}{b} + \frac{c}{d} = \frac{ad + bc}{bd}, \quad \frac{a}{b} \cdot \frac{c}{d} = \frac{ac}{bd}$$

($bd \neq 0$ por ser D un dominio de integridad), que están bien definidas.

El cero es $0/b$, la unidad es a/a y el inverso de $a/b (a \neq 0)$ es b/a. Con estas operaciones,

$$\frac{D \times (D \backslash \{0\})}{\mathcal{R}}$$

es un cuerpo, al que llamaremos *el cuerpo de fracciones* de D.

El cuerpo de fracciones de D es el mínimo cuerpo que contiene a D.

1.2.3. Elementos primitivos

Cuando trabajamos con cuerpos finitos, necesitamos poder sumar y multiplicar de la manera más simple posible. Recordemos que por el Teorema 1.18, sabemos que $\mathbb{F}_q$, donde $q = p^n$ con p primo, es un espacio vectorial sobre $\mathbb{F}_p$ de dimensión n. De esta manera, la suma en $\mathbb{F}_q$ consistirá en la suma usual de n-tuplas sobre el cuerpo $\mathbb{F}_p$. Sin embargo, la multiplicación en $\mathbb{F}_q$ no es tan sencilla y por eso buscaremos una forma de expresar los elementos del cuerpo que simplifique esta operación.

El conjunto $\mathbb{F}_q^*$ de los elementos no nulos de $\mathbb{F}_q$ es un *grupo*. El siguiente teorema nos será de utilidad.

Teorema 1.27. *Se verifican las siguientes afirmaciones:*

(i) El grupo $\mathbb{F}_q^*$ es cíclico con respecto a la multiplicación y tiene orden $q-1$.

(ii) Si γ es un generador de este grupo cíclico, entonces

$$\mathbb{F}_q = \{0, 1 = \gamma^0, \gamma, \gamma^2, \ldots, \gamma^{q-2}\},$$

y $\gamma^i = 1$ si y solo si $(q-1) \mid i$.

Cada generador γ de $\mathbb{F}_q^*$ se llama *elemento primitivo* de $\mathbb{F}_q$. Cuando los elementos no nulos de un cuerpo finito se expresan como potencias de γ, la multiplicación en el cuerpo se realiza de forma sencilla teniendo en cuenta que

$$\gamma^i \gamma^j = \gamma^{i+j} = \gamma^s, \text{ donde } 0 \leq s \leq q-2 \text{ y } i + j \equiv s \pmod{q-1}.$$

Sea γ un elemento primitivo de $\mathbb{F}_q$, entonces $\gamma^{q-1} = 1$ por definición. De esta manera $(\gamma^i)^{q-1} = 1$ para $0 \leq i \leq q-2$ probando que los elementos de $\mathbb{F}_q^*$ son las raíces de $x^{q-1} - 1 \in \mathbb{F}_p[x]$ y, por tanto, de $x^q - x$. Como 0 es raíz de $x^q - x$, se tiene el siguiente teorema.

Teorema 1.28. *Las raíces del polinomio $x^q - x \in \mathbb{F}_p[x]$ son los elementos de $\mathbb{F}_q$.*

Definición 1.29. Un elemento $\xi \in \mathbb{F}_q$ es una *n-ésima raíz de la unidad* si $\xi^n = 1$. Además, si $\xi^s \neq 1$ para $s \in \{1, \ldots, n-1\}$, diremos que es una *n-ésima raíz primitiva de la unidad*.

Un elemento primitivo γ de $\mathbb{F}_q$ es, por tanto, una $(q-1)$-ésima raíz primitiva de la unidad. Se deduce del Teorema 1.27 que el cuerpo $\mathbb{F}_q$ contiene una n-ésima raíz primitiva de la unidad si y solo si $n \mid (q-1)$, en cuyo caso $\gamma^{(q-1)/n}$ es una n-ésima raíz primitiva de la unidad.

1.2.4. Automorfismos de cuerpos finitos

La siguiente definición nos será de utilidad a la hora de trabajar con anillos de polinomios de Ore.

Definición 1.30. Un *automorfismo* σ de $\mathbb{F}_q$ es una biyección $\sigma : \mathbb{F}_q \to \mathbb{F}_q$ que verifica:

- $\sigma(\alpha + \beta) = \sigma(\alpha) + \sigma(\beta) \quad \forall \alpha, \beta \in \mathbb{F}_q$.
- $\sigma(\alpha\beta) = \sigma(\alpha)\sigma(\beta) \quad \forall \alpha, \beta \in \mathbb{F}_q$.

La aplicación $\sigma_p : \mathbb{F}_q \to \mathbb{F}_q$, donde $q = p^n$ con p primo, definida como $\sigma_p(\alpha) = \alpha^p$, $\forall \alpha \in \mathbb{F}_q$, es un *automorfismo*. Obviamente, $\sigma_p(\alpha\beta) = (\alpha\beta)^p = (\alpha)^p(\beta)^p = \sigma_p(\alpha)\sigma_p(\beta)$, y $\sigma_p(\alpha + \beta) = (\alpha + \beta)^p = (\alpha^p + \beta^p) = \sigma_p(\alpha) + \sigma_p(\beta)$, donde se ha usado la Proposición 1.19. Además, el núcleo de la aplicación es $\{0\}$ y, por tanto, es un automorfismo, al que se le denomina *automorfismo de Frobenius*. De manera análoga, definimos $\sigma_{p^r}(\alpha) = \alpha^{p^r}$.

El conjunto de automorfismos de $\mathbb{F}_q$ forma un grupo con respecto a la operación de composición de funciones. Este grupo se denota como $\mathrm{Gal}(\mathbb{F}_q)$ y se llama el *grupo de Galois* de $\mathbb{F}_q$. Así, definiremos el *orden* de un automorfismo como el menor entero n tal que $\sigma^n(\alpha) = \alpha$, $\forall \alpha \in \mathbb{F}_q$.

Teorema 1.31. *El grupo de Galois $\mathrm{Gal}(\mathbb{F}_q)$, donde $p = q^n$ y p es primo, es cíclico de orden n y está generado por el automorfismo de Frobenius σ_p.*

Dado un elemento $\alpha \in \mathbb{F}_q$, diremos que queda *fijo* por un automorfismo σ si $\sigma(\alpha) = \alpha$. El conjunto de los elementos de $\mathbb{F}_q$ que quedan fijos por un automorfismo σ forma un subcuerpo de $\mathbb{F}_q$ y se denomina *subcuerpo fijo de* $\mathbb{F}_q$ por σ, se denota por $\mathbb{F}_q^\sigma$.

1.2.5. Clases ciclotómicas y polinomios minimales

Sea $\mathbb{E}$ una extensión de cuerpos finita de $\mathbb{F}_q$. Entonces $\mathbb{E}$ es un espacio vectorial sobre $\mathbb{F}_q$ y entonces $\mathbb{E} = \mathbb{F}_{q^t}$ para algún entero positivo t. Por el Teorema 1.28, cada elemento α de $\mathbb{F}_{q^t}$ es una raíz del polinomio $x^{q^t} - x$, luego, existe un polinomio mónico $M_\alpha(x) \in \mathbb{F}_q[x]$ cuyo grado es mínimo y contiene a α como raíz. Este polinomio se denomina *polinomio minimal de* α sobre $\mathbb{F}_q$. En el siguiente teorema veremos algunos hechos elementales acerca de los polinomios minimales.

Teorema 1.32. *Sea $\mathbb{F}_q \leq \mathbb{F}_{q^t}$ una extensión de cuerpos y sea α un elemento de $\mathbb{F}_{q^t}$ cuyo polinomio minimal es $M_\alpha(x) \in \mathbb{F}_q[x]$. Se verifican las siguientes afirmaciones:*

(i) *$M_\alpha(x)$ es irreducible en $\mathbb{F}_q$.*

(ii) *Si $g(x)$ es cualquier polinomio en $\mathbb{F}_q[x]$ tal que $g(\alpha) = 0$, entonces $M_\alpha(x) \mid g(x)$.*

(iii) *$M_\alpha(x)$ es único, es decir, existe un único polinomio mónico en $\mathbb{F}_q[x]$ de grado mínimo que tiene a α como raíz.*

Sea $f(x)$ un polinomio irreducible en $\mathbb{F}_q$ de grado r. Podemos considerar la extensión generada por una de las raíces de $f(x)$ y obtenemos el cuerpo $\mathbb{F}_{q^r}$. El siguiente teorema nos afirma que en dicho caso, todas las raíces de $f(x)$ están en $\mathbb{F}_{q^r}$.

Teorema 1.33. *Sea $f(x)$ un polinomio mónico irreducible en $\mathbb{F}_q$ de grado r. Entonces:*

(i) Todas las raíces de $f(x)$ están en $\mathbb{F}_{q^r}$ y en cualquier extensión de cuerpos de $\mathbb{F}_q$ generada por una de sus raíces.

(ii) $f(x) = \prod_{i=1}^{r}(x - \alpha_i)$, donde $\alpha_i \in \mathbb{F}_{q^r}$, para $i \in \{1, \ldots, r\}$.

(iii) El polinomio $f(x)$ divide a $x^{q^r} - x$.

En particular, este teorema se verifica para los polinomios minimales $M_\alpha(x)$ en $\mathbb{F}_q$ dado que estos polinomios son mónicos e irreducibles.

Teorema 1.34. *Sea $\mathbb{F}_q \leq \mathbb{F}_{q^t}$ una extensión de cuerpos, y sea α un elemento de $\mathbb{F}_{q^t}$ con polinomio minimal $M_\alpha(x)$ en $\mathbb{F}_q[x]$. Se verifican las siguientes afirmaciones:*

(i) El polinomio $M_\alpha(x)$ divide a $x^{q^t} - x$.

(ii) El polinomio $M_\alpha(x)$ tiene raíces distintas que pertenecen a $\mathbb{F}_{q^t}$.

(iii) El grado de $M_\alpha(x)$ divide a t.

(iv) $x^{q^t} - x = \prod_\alpha M_\alpha(x)$, donde α varía entre un subconjunto de $\mathbb{F}_{q^t}$ de forma que enumera los polinomios minimales de todos los elementos de $\mathbb{F}_{q^t}$ una única vez.

(v) $x^{q^t} - x = \prod_f f(x)$, donde f varía entre todos los polinomios mónicos irreducibles en $\mathbb{F}_q$ cuyo grado divide a t.

Definición 1.35. Dos elementos de $\mathbb{F}_{q^t}$ son *conjugados* en $\mathbb{F}_q$ si tienen el mismo polinomio minimal en $\mathbb{F}_q[x]$.

Será de importancia encontrar todos los conjugados de $\alpha \in \mathbb{F}_q$, es decir, las raíces de $M_\alpha(x)$. Sabemos por el Teorema 1.34 que todas sus raíces son distintas y están en $\mathbb{F}_{q^t}$. Podremos encontrar estas raíces con ayuda del siguiente teorema.

Teorema 1.36. *Sea $f(x)$ un polinomio en $\mathbb{F}_q[x]$ y sea α una raíz de $f(x)$ en alguna extensión $\mathbb{F}_{q^t}$. Entonces:*

(i) $f(x^q) = f(x)^q$

(ii) α^q es también una raíz de $f(x)$ en $\mathbb{F}_q$.

Demostración. (i). Sea $f(x) = \sum_{i=0}^{m} a_i x^i$. Puesto que $q = p^n$, donde p es la característica de $\mathbb{F}_q$, se tiene que aplicando la Proposición 1.19, $f(x)^q = \sum_{i=0}^{m} a_i^q x^{iq}$. Sin embargo, $a_i^q = a_i$, ya que por el Teorema 1.28, los elementos de $\mathbb{F}_q$ son las raíces de $x^q - x$, lo que prueba *(i)*. En particular, $f(\alpha^p) = f(\alpha)^q = 0$, probando *(ii)*. □

Aplicando iteradamente el teorema anterior obtenemos que $\alpha, \alpha^q, \alpha^{q^2}, \ldots$ son raíces de $M_\alpha(x)$. La secuencia anterior terminará tras r términos, siendo r el menos entero tal que $\alpha^{q^r} = \alpha$.

Sea γ un elemento primitivo de $\mathbb{F}_{q^t}$, entonces $\alpha = \gamma^s$ para algún s. Así, $\alpha^{q^r} = \alpha$ si y solo si $\gamma^{sq^r - s} = 1$. Por el Teorema 1.27, $sq^r \equiv s \pmod{q^t - 1}$. En base a esto, definimos la *clase q-ciclotómica de s módulo* $q^t - 1$ como el conjunto

$$C_s = \{s, sq, \ldots, sq^{r-1}\} \pmod{q^t - 1},$$

donde r es el entero positivo más pequeño tal que $sq^r \equiv s \pmod{q^t - 1}$. Los conjuntos C_s dividen el conjunto de enteros $\{0, 1, \ldots, q^t - 2\}$ en conjuntos disjuntos.

Ejemplo 1.37. Las clases 2-ciclotómicas módulo 15 son $C_0 = \{0\}, C_1 = \{1, 2, 4, 8\}, C_3 = \{3, 6, 12, 9\}, C_5 = \{5, 10\}$ y $C_7 = \{7, 14, 13, 11\}$.

Ahora sabemos que las raíces de $M_\alpha(x) = M_{\gamma^s}(x)$ contienen a $\{\gamma^i \mid i \in C_s\}$. De hecho, estas son todas las raíces. Por tanto, si conocemos el tamaño de C_s, sabremos el grado de $M_\alpha(x)$.

Teorema 1.38. *Si γ es un elemento primitivo de $\mathbb{F}_{q^t}$, entonces el polinomio minimal de γ^s en $\mathbb{F}_q$ es*

$$M_{\gamma^s}(x) = \prod_{i \in C_s} (x - \gamma^i).$$

1.3. Módulos

1.3.1. Conceptos básicos de módulos

En esta sección veremos una estructura algebraica muy importante para el estudio de los códigos convolucionales: los módulos. Las principales fuentes para la redacción de esta sección han sido [39] y [9].

Informalmente, un módulo es un "espacio vectorial" sobre un anillo y no sobre un cuerpo. Este anillo puede ser (o no) conmutativo.

Definición 1.39. Sea $\mathcal{R}$ un anillo, entonces un $\mathcal{R}$-módulo por la izquierda M es un grupo abeliano $(M, +)$ y una operación $\cdot : \mathcal{R} \times M \to M$ tales que $\forall r, s \in \mathcal{R}$, $\forall x, y \in M$ se tiene:

1. $(rs)x = r(sx)$.
2. $(r + s)x = rx + sx$.
3. $r(x + y) = rx + ry$.
4. $1x = x$.

La definición de los $\mathcal{R}$-módulos por la derecha es análoga, sólo que el anillo actúa *por la derecha*, es decir, se define una multiplicación escalar de la forma $\cdot : M \times \mathcal{R} \to M$.

Cuando $\mathcal{R}$ es conmutativo, hablaremos simplemente de $\mathcal{R}$-módulos.

Ejemplo 1.40. Veamos algunos ejemplos de módulos.

1. Sea V un espacio vectorial sobre un cuerpo K, entonces V es un K-módulo.
2. Sea $(\mathcal{A}, +, \cdot)$ un anillo. Entonces, $\mathcal{A}$ es un módulo por la izquierda y por la derecha sobre sí mismo utilizando su producto.

Definición 1.41. Sea M un $\mathcal{R}$-módulo por la izquierda. Un *submódulo* N de M es un subgrupo de $(M, +)$ cerrado para la multiplicación con elementos en $\mathcal{R}$, es decir, $rn \in N$ para todo $r \in \mathcal{R}, n \in N$.

De esta forma, N también es un $\mathcal{R}$-módulo con la operación $\cdot : \mathcal{R} \times N \to N$.

Definición 1.42. Se dice que un $\mathcal{R}$-módulo M es *simple* si sus únicos submódulos son $\{0\}$ y M.

Definición 1.43. Sea M un $\mathcal{R}$-módulo, sean N, N_1, N_2 dos subconjuntos no vacíos de M y sea $A \subset \mathcal{R}$. Entonces definimos:

$$N_1 + N_2 = \{n_1 + n_2 \mid n_1 \in N_1, n_2 \in N_2\} \subset M,$$

$$AN = \left\{ \sum_{i=1}^{k} a_i n_i \mid a_i \in A, n_i \in N, k \in \mathbb{N} \right\} \subset M.$$

Si N_1 y N_2 son submódulos, entonces $N_1 + N_2$ es también un submódulo de M. Además, para un ideal por la izquierda $A \subset \mathcal{R}$, el producto AN es siempre un submódulo de M.

Definición 1.44. Sean M_1, M_2 submódulos de un $\mathcal{R}$-módulo M. Si $M = M_1 + M_2$ y $M_1 \cap M_2 = \{0\}$, entonces se dice que M es una *suma directa* de M_1 y M_2. Se denota como $M = M_1 \oplus M_2$.

En este caso, cada $m \in M$ se calcula de forma única como $m = m_1 + m_2$, con $m_1 \in M_1, m_2 \in M_2$. Los submódulos M_1 y M_2 se llaman *sumandos directos* de M.

1.3.2. Módulos libres

Definición 1.45. Sea M un $\mathcal{R}$-módulo por la izquierda y sea I un conjunto arbitrario de índices. Diremos que un conjunto $\{e_i : i \in I\}$ *genera* M si cualquier elemento $m \in M$ puede escribirse como una combinación lineal $m = \sum_{i \in S} \lambda_i e_i$, siendo $S \subset I$ finito. Si además este conjunto es linealmente independiente, diremos que es una *base* de M.

Teorema 1.46. *Sea M un $\mathcal{R}$-módulo. Equivalen las siguientes afirmaciones.*

(i) M tiene una base $\{e_i : i \in I\}$.

(ii) $M \cong \bigoplus_{i \in I} \mathcal{R}$.

Definición 1.47. Un *módulo libre* es un $\mathcal{R}$-módulo por la izquierda M que posee una base. El *rango* de M es la cardinalidad de la base.

1.3.3. El módulo $\mathbb{F}[t]^n$

En el ejemplo 1.40 vimos que un anillo es un módulo sobre sí mismo.

Sea $\mathbb{F}[t]$ un anillo de polinomios con coeficientes en un cuerpo finito $\mathbb{F}$. Entonces,

$$\mathbb{F}[t]^n := \{(v_1, \ldots, v_n) \mid v_i \in \mathbb{F}[t], i = 1, \ldots, n\}.$$

En esta sección veremos dos resultados referentes a los submódulos y sumandos directos de $\mathbb{F}[t]^n$. Estos resultados pueden encontrarse en [9] y se omitirán las demostraciones.

Proposición 1.48. *Sea V un submódulo de $\mathbb{F}[t]^n$. Se verifican las siguientes afirmaciones*

(i) V es un módulo libre y su rango es finito.

(ii) Si $v_1, \ldots, v_r \in \mathbb{F}[t]^n$ forman un conjunto generador de V, entonces $V = Im(M) = \{uM \mid u \in \mathbb{F}[t]^r\}$ donde

$$M := \begin{bmatrix} v_1 \\ \vdots \\ v_r \end{bmatrix} \in \mathbb{F}[t]^{r \times n}. \tag{1.1}$$

Llamaremos a M matriz generadora de V.

(iii) Sean $P \in \mathbb{F}[t]^{r \times r}$ y M como en (ii). Entonces $V = Im(PM) \Leftrightarrow P$ es invertible sobre $\mathbb{F}[t]$.

Proposición 1.49. *Sea $V \subseteq \mathbb{F}[t]^n$ un submódulo y sea $v_1, v_2, \ldots, v_r \in \mathbb{F}[t]^n$ un conjunto generador de V. Sea M la matriz (1.1). Equivalen las siguientes afirmaciones.*

(i) V es un sumando directo de $\mathbb{F}[t]^n$.

(ii) Cualquier base de V puede ser completada a una base de $\mathbb{F}[t]^n$.

(iii) La forma normal de Smith de M está dada por la matriz $\begin{pmatrix} I_k & 0 \\ 0 & 0 \end{pmatrix}$, donde k es el rango de V.

(iv) Si $\{v_1, \ldots, v_r\}$ es una base de V (equivalentemente, si r es el rango de V), entonces M es invertible por la derecha.

(v) Para todo $v \in \mathbb{F}[t]^n$ y todo $\lambda \in \mathbb{F}[t] \setminus \{0\}$ se tiene que

$$\lambda v \in V \Rightarrow v \in V. \tag{1.2}$$

(vi) Existe una matriz $N \in \mathbb{F}[t]^{n \times 1}$ tal que $V = \ker N := \{v \in \mathbb{F}[t]^n \mid vN = 0\}$.

(vii) Para todos los submódulos $W \in \mathbb{F}[t]^n$ con el mismo rango de V se tiene que

$$V \subseteq W \Rightarrow V = W.$$

Una matriz con la propiedad (iii) se dirá que es básica.

2. Códigos de bloque

En este capítulo trataremos algunos de los fundamentos de la teoría de códigos de bloque, centrándonos principalmente en los códigos lineales. Partiremos de la definición más básica de código, es decir, un subconjunto de un cuerpo finito. Después se definirá el concepto de código lineal, añadiendo a los códigos una estructura de espacio vectorial que nos permitirá utilizar herramientas más potentes propias del álgebra lineal. Por último, estudiaremos una subclase de los códigos lineales, los códigos cíclicos. Estos nos permitirán aplicar algoritmos de decodificación muy eficientes. Las principales fuentes consultadas para este capítulo han sido [15] y [1].

2.1. Códigos lineales

Consideremos el espacio vectorial de todas las n-tuplas sobre el cuerpo finito $\mathbb{F}_q$, que denotaremos $\mathbb{F}_q^n$.

Definición 2.1. Un (n, M)-código $\mathcal{C}$ sobre el cuerpo $\mathbb{F}_q$ es un subconjunto de $\mathbb{F}_q^n$ de tamaño M. A los elementos de $\mathcal{C}$ los llamaremos *palabras código*. A n se le llama la *longitud* del código. Normalmente, cada vector $(a_1, a_2, ..., a_n) \in \mathbb{F}_q^n$ lo escribiremos de forma más compacta como $a_1 \cdots a_n$.

Ejemplo 2.2. Un elemplo de código podría ser un (4,6)-código formado por los siguientes elementos:

$$1100, \quad 1011, \quad 1101, \quad 1110, \quad 1111, \quad 0001.$$

Sin ninguna estructura adicional, las propiedades de un código son limitadas. Por tanto, dotaremos al código de *linealidad*. Esto nos permitirá aplicar herramientas matemáticas más potentes que nos hagan sacarle más provecho.

Definición 2.3. $\mathcal{C}$ es un $[n, k]$-código lineal sobre $\mathbb{F}_q$ si es un subespacio k-dimensional de $\mathbb{F}_q^n$.

De esta forma, un $[n, k]$-código lineal $\mathcal{C}$ tendrá q^k elementos. Como veremos, la linealidad tiene varias ventajas a la hora de codificar y decodificar de manera más eficiente.

Definición 2.4. Una *matriz generadora* para un $[n, k]$-código lineal $\mathcal{C}$ es cualquier matriz G de dimensión $k \times n$ cuyas filas forman una base de $\mathcal{C}$.

En general, hay varias matrices generadoras para un código. Un conjunto de información de $\mathcal{C}$ es cualquier conjunto de k columnas linealmente independientes de G. Las demás $r = n - k$ columnas se denominan conjunto redundante y r es la redundancia de $\mathcal{C}$.

Si las primeras k columnas forman un *conjunto de información*, el código tiene una única matriz generadora de la forma $[I_k \mid A]$, donde I_k es la matriz identidad de dimensión k. Se dice que dicha matriz está en *forma estándar*.

Debido a que un código lineal es un subespacio vectorial, sabemos que es el núcleo de una transformación lineal. Esta propiedad da pie a la siguiente definición.

Definición 2.5. Una matriz H de dimensión $(n-k) \times n$ es una matriz de paridad para un $[n,k]$-código lineal $\mathcal{C}$ si cumple

$$\mathcal{C} = \left\{ \mathbf{x} \in \mathbb{F}_q^n \mid H\mathbf{x}^{\mathbf{T}} = 0 \right\}.$$

Nótese que las filas de H también serán linealmente independientes. En general, al igual que ocurría con las matrices generadoras, pueden existir varias matrices de paridad para $\mathcal{C}$.
El siguiente teorema nos da una de ellas cuando $\mathcal{C}$ tiene una matriz generadora en forma estándar.

Teorema 2.6. *Si* $G = [I_k \mid A]$ *es una matriz generadora para un* $[n,k]$*-código lineal* $\mathcal{C}$ *en forma estándar, entonces* $H = \left[-A^T \mid I_{n-k}\right]$ *es una* matriz de paridad *para* $\mathcal{C}$.

Demostración. Puesto que $HG^T = -A^T + A^T = 0$, se tiene que $\mathcal{C}$ está contenido en el núcleo de la transformación $\mathbf{x} \to H\mathbf{x}^T$. Como H tiene rango $n-k$, esta transformación lineal tiene un núcleo de dimensión k, que coincide con la dimensión de $\mathcal{C}$. □

Ejemplo 2.7. Consideramos la matriz $G = [I_4 \mid A]$, donde

$$G = \left[\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array}\right]$$

es la matriz generadora en forma estándar de un $[7,4]$-código binario $\mathcal{H}_3$. Utilizando el Teorema 2.6, una matriz de paridad para $\mathcal{H}_3$ es

$$H = [A^T \mid I_3] = \left[\begin{array}{cccc|ccc} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{array}\right].$$

Este código se llama $[7,4]$-*código de Hamming*.

2.1.1. Códigos duales

Recordemos que la matriz generadora G de un $[n,k]$-código $\mathcal{C}$ es simplemente una matriz cuyas filas forman una base del código visto como subespacio vectorial. Las filas de una matriz de paridad H también son linealmente independientes y, por tanto, forman una base de algún espacio vectorial. Así, H es la matriz generadora de un código, que llamaremos *dual u ortogonal* de $\mathcal{C}$ y lo denotaremos como $\mathcal{C}^\perp$. Nótese que $\mathcal{C}^\perp$ es un $[n, n-k]$-código.
Una forma alternativa de definir el código dual es usando el producto escalar. Recordemos que el producto escalar de dos vectores $\mathbf{x} = x_1 \dots x_n, \mathbf{y} = y_1 \dots y_n$ de $\mathbb{F}_q^n$ se define como:

$$\mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^{n} x_i y_i.$$

Por tanto, otra definición alternativa es la siguiente.

Definición 2.8. Dado un $[n,k]$-código $\mathcal{C}$, su *código dual* $\mathcal{C}^\perp$ es

$$\mathcal{C}^\perp = \left\{ \mathbf{x} \in \mathbb{F}_q^n \mid \mathbf{x} \cdot \mathbf{c} = 0,\ \forall c \in \mathcal{C} \right\}.$$

La siguiente proposición caracteriza a las matrices generadoras y de paridad de $\mathcal{C}^\perp$ en términos de las de $\mathcal{C}$.

Proposición 2.9. *Si G y H son las matrices generadoras y de paridad de $\mathcal{C}$, respectivamente, entonces H y G son las matrices generadoras y de paridad para $\mathcal{C}^\perp$ respectivamente.*

Demostración. Se tiene que

$$\mathbf{y} \in \mathcal{C}^\perp \Leftrightarrow \mathbf{c} \cdot \mathbf{y} = 0, \forall c \in \mathcal{C} \Leftrightarrow (\mathbf{x}G) \cdot \mathbf{y} = 0, \forall \mathbf{x} \in \mathbb{F}_q^k \Leftrightarrow \mathbf{x}G\mathbf{y}^T = 0, \forall \mathbf{x} \in \mathbb{F}_q^k \Leftrightarrow G\mathbf{y}^T = \mathbf{0},$$

donde hemos utilizado que $rg(G) = k$.

Por tanto,

$$\mathcal{C}^\perp = \left\{ \mathbf{y} \in \mathbb{F}_q^n \mid G\mathbf{y}^T = \mathbf{0} \right\},$$

luego, G es una matriz de paridad de $\mathcal{C}^\perp$.

Veamos ahora que H es una matriz generadora para $\mathcal{C}^\perp$. Puesto que $H\mathbf{c}^T = 0, \forall c \in \mathcal{C}$ (por ser una matriz de paridad para $\mathcal{C}$), cada fila de H está en $\mathcal{C}^\perp$. Utilizando que H tiene $n-k$ filas linealmente independientes y $dim(\mathcal{C}^\perp) = n-k$, obtenemos que H es una matriz generadora de $\mathcal{C}^\perp$. □

Definición 2.10. Un código $\mathcal{C}$ es auto-ortogonal si $\mathcal{C} \subseteq \mathcal{C}^\perp$ y diremos que es auto-dual si $\mathcal{C} = \mathcal{C}^\perp$.

Proposición 2.11. *Un código auto-dual tiene longitud par y su dimensión es $n/2$.*

Demostración. Si un $[n,k]$-código $\mathcal{C}$ tiene dimensión k, sabemos que su dual $\mathcal{C}^\perp$ tiene dimensión $n-k$. Si suponemos que es auto-dual, entonces $\mathcal{C} = \mathcal{C}^\perp$ y, por tanto, $n-k=k$. Así $n=2k$, n es par y la dimensión de $\mathcal{C}$ es $n/2$. □

Ejemplo 2.12. El $[4,2]$-código ternario $\mathcal{H}_{3,2}$, también llamado tetracódigo, tiene una matriz generadora G en forma estándar dada por

$$G = \left[\begin{array}{cc|cc} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & -1 \end{array}\right].$$

Veamos que este código es auto-dual.

En primer lugar, comprobemos que es auto-ortogonal, es decir, si $\mathcal{C} \subseteq \mathcal{C}^\perp$. Dado $\mathbf{c} \in \mathcal{C}$, sabemos que $\mathbf{c}$ pertenece a $\mathcal{C}^\perp$ si

$$G\mathbf{c}^T = \mathbf{0} \Leftrightarrow \mathbf{c}G^T = \mathbf{0}.$$

Así, como $c = xG, x \in \mathbb{F}_q^k$, debemos comprobar que $GG^T = O$.

$$GG^T = \left[\begin{array}{cccc} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & -1 \end{array}\right] \left[\begin{array}{cc} 1 & 0 \\ 0 & 1 \\ 1 & 1 \\ 1 & -1 \end{array}\right] = O.$$

Por tanto, $\mathcal{H}_{3,2}$ es *auto-ortogonal*. Además, dado que $dim(\mathcal{C}^\perp) = 8-4 = 4 = dim(\mathcal{C})$, $\mathcal{C}$ es también auto-dual.

2.1.2. Pesos y distancias

Al tratar de corregir errores, es esencial establecer algún criterio para medir cuán diferentes son las palabras *transmitidas y las recibidas*. Además, un *invariante* importante de un código $\mathcal{C}$ será la distancia mínima entre palabras código distintas.

Definición 2.13. La *distancia de Hamming* $d(\mathbf{x},\mathbf{y})$ entre dos vectores $\mathbf{x},\mathbf{y} \in \mathbb{F}_q^n$ se define como el número de coordenadas en las que $\mathbf{x}$ e $\mathbf{y}$ difieren.

Veamos que efectivamente es una distancia.

Teorema 2.14. *La función distancia $d(\mathbf{x},\mathbf{y})$ de Hamming satisface las siguientes propiedades:*

(i) $d(\mathbf{x},\mathbf{y}) \geq 0\ \forall \mathbf{x},\mathbf{y} \in \mathbb{F}_q^n$ *(no negatividad).*

(ii) $d(\mathbf{x},\mathbf{y}) = 0$ si y solo si $\mathbf{x} = \mathbf{y}$.

(iii) $d(\mathbf{x},\mathbf{y}) = d(\mathbf{y},\mathbf{x})\ \forall \mathbf{x},\mathbf{y} \in \mathbb{F}_q^n$ *(simetría).*

(iv) $d(\mathbf{x},\mathbf{z}) \leq d(\mathbf{x},\mathbf{y}) + d(\mathbf{y},\mathbf{z})\ \forall \mathbf{x},\mathbf{y},\mathbf{z} \in \mathbb{F}_q^n$ *(desigualdad triangular).*

Demostración. Las tres primeras propiedades son evidentes. Veamos que se da la desigualdad triangular. Sean $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{F}_q^n$. Supongamos que $d(\mathbf{x}, \mathbf{y}) = r$ y que $d(\mathbf{y}, \mathbf{z}) = s$. Por tanto, $\mathbf{x}$ e $\mathbf{y}$ difieren en r posiciones, digamos $i_1, i_2, \ldots, i_r$. Análogamente, $\mathbf{z}$ e $\mathbf{y}$ difieren en las posiciones $j_1, j_2, \ldots, j_s$. Entonces, todas las posiciones en las que $\mathbf{x}$ y $\mathbf{z}$ difieran tienen que estar en el conjunto $\{i_1, i_2, \ldots, i_r, j_1, j_2, \ldots, j_s\}$, que tiene como máximo $r + s$ elementos, lo que nos dice que

$$d(\mathbf{x}, \mathbf{z}) \leq r + s = d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z})$$

□

Ejemplo 2.15. Sean $\mathbf{x} = 11001001$ e $\mathbf{y} = 11010111$ elementos de $\mathbb{F}_2^8$, entonces su *distancia de Hamming* es $d(\mathbf{x}, \mathbf{y}) = 4$.

La *distancia mínima* de un código $\mathcal{C}$ es la distancia más pequeña entre palabras código distintas. Es esencial a la hora de determinar la capacidad de corregir errores de $\mathcal{C}$ como veremos en el siguiente teorema. Si la distancia mínima d de un $[n, k]$-código se conoce, nos referiremos al código como un $[n, k, d]$-código.

Teorema 2.16 (Decodificación de máxima verosimilitud)**.** *Sea $\mathcal{C}$ un $[n, k, d]$-código. Entonces, $\mathcal{C}$ puede detectar hasta $d - 1$ errores y puede corregir hasta*

$$t = \left\lfloor \frac{d-1}{2} \right\rfloor$$

Demostración. Si hay menos de d errores, debido a que la distancia mínima es d, no se obtendrá una palabra código, y, por tanto, podemos detectar hasta $d - 1$ errores. Para la segunda parte, supongamos que se envía una palabra $\mathbf{u}$ y se recibe una palabra $\mathbf{v}$ con, a lo sumo, t errores. Entonces $d(\mathbf{u}, \mathbf{v}) \leq t$. Supongamos que existe otra palabra $\mathbf{w} \in \mathcal{C}$ tal que $d(\mathbf{u}, \mathbf{w}) \leq t$, entonces utilizando la desigualdad triangular de la distancia de Hamming:

$$d(\mathbf{v}, \mathbf{w}) = d(\mathbf{v}, \mathbf{u}) + d(\mathbf{u}, \mathbf{w}) \leq 2t \leq d - 1.$$

Esto es una contradicción, puesto que recordemos que d es la distancia mínima entre dos palabras código, por lo que $\mathcal{C}$ puede corregir hasta t errores. □

De esta forma, nos interesará que la *distancia mínima* sea lo mayor posible, con el fin de detectar el mayor número de errores.

Veamos otro concepto interesante relacionado con la distancia de Hamming.

Definición 2.17. El *peso de Hamming* $wt(x)$ de un vector $\mathbf{x} \in \mathbb{F}_q^n$ es el número de coordenadas no nulas en $\mathbf{x}$.

Ejemplo 2.18. Sea $\mathbf{x} = 1101001$ un elemento de $\mathbb{F}_2^7$, entonces $wt(\mathbf{x}) = 4$. Nótese que $wt(\mathbf{x}) = d(\mathbf{x}, \mathbf{0}) = 4$.

La relación con la *distancia de Hamming* se sigue del siguiente teorema.

Teorema 2.19. *Sean $\mathbf{x}, \mathbf{y} \in \mathbb{F}_q^n$, entonces $d(\mathbf{x}, \mathbf{y}) = wt(\mathbf{x} - \mathbf{y})$. Además, si $\mathcal{C}$ es un código lineal, la distancia mínima d coincide con el mínimo peso de Hamming de una palabra código no nula en $\mathcal{C}$.*

Demostración. La primera parte de la prueba es inmediata, ya que si tenemos dos elementos distintos, el número de coordenadas en las que difieren es el mismo que el número de coordenadas no nulas al restarlos, esto es $d(\mathbf{x}, \mathbf{y}) = d(\mathbf{0}, \mathbf{x} - \mathbf{y}) = wt(\mathbf{x} - \mathbf{y})$.
Para la segunda parte, queremos ver que

$$\min_{\mathbf{u},\mathbf{v} \in \mathcal{C}} d(\mathbf{u}, \mathbf{v}) = \min_{\mathbf{w} \in \mathcal{C}^*} wt(\mathbf{w}).$$

Sean $\mathbf{u}, \mathbf{v} \in \mathcal{C}$ palabras códigos cuya distancia es mínima, entonces por ser $\mathcal{C}$ un subespacio vectorial, se tiene que $\mathbf{u} - \mathbf{v} \in \mathcal{C}$. De esta forma, $d(\mathbf{u}, \mathbf{v}) = wt(\mathbf{u} - \mathbf{v})$. Por tanto,

$$\min_{\mathbf{w} \in \mathcal{C}^*} wt(\mathbf{w}) \leq \min_{\mathbf{u},\mathbf{v} \in \mathcal{C}} d(\mathbf{u}, \mathbf{v}).$$

Por otro lado, sea $\mathbf{w} \in \mathcal{C}^*$ la palabra código cuyo peso de Hamming es mínimo. Así, como $wt(\mathbf{w}) = d(\mathbf{0}, \mathbf{w})$ tenemos

$$\min_{\mathbf{u},\mathbf{v} \in \mathcal{C}} d(\mathbf{u}, \mathbf{v}) \leq \min_{\mathbf{w} \in \mathcal{C}^*} wt(\mathbf{w}).$$

Y se consigue la igualdad deseada. □

Como consecuencia del teorema anterior, la *distancia mínima* también se conoce como *peso mínimo del código*.

Otra cota importante para códigos lineales es la *cota de Singleton*.

Teorema 2.20 (Cota de Singleton)**.** *Sea $\mathcal{C}$ un $[n, k, d]$-código lineal sobre $\mathbb{F}_q$. Entonces,*

$$d \leq n - k + 1.$$

Demostración. Recordemos que $\mathcal{C}$ tiene q^k palabras código de longitud n. Para cada palabra código $\mathbf{c} = c_1 \cdots c_{k-1} \cdots c_n$ consideraremos únicamente las primeras $k - 1$ componentes. Existen q^{k-1} palabras de longitud $k - 1$ en $\mathbb{F}_q^{k-1}$. Dado que tenemos un total de q^k palabras código, deben existir dos palabras código que tengan las primeras $k - 1$ componentes iguales. La distancia entre estas dos palabras código es $n - (k - 1)$ y, por tanto, $d \leq n - (k - 1) = n - k + 1$. □

Los códigos que alcancen la cota de Singleton se denominan códigos MDS (máxima distancia separable).

Definición 2.21. Sea $A_i(\mathcal{C})$ el número de palabras código de peso i en $\mathcal{C}$. Para cada $0 \leq i \leq n$, la lista $A_i(\mathcal{C})$ se denomina *distribución de pesos o espectro de pesos* de $\mathcal{C}$.

Una gran parte de la investigación en teoría de códigos ha sido el cálculo de la distribución de pesos de códigos específicos o de familias de códigos conocidas.

Ejemplo 2.22. Vamos a calcular la distribución de pesos del código binario $\mathcal{C}$ con la siguiente matriz generadora:

$$G = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

Sea $\mathbf{x} = (x_1, x_2, x_3) \in \mathbb{F}_2^3$. Entonces,

$$(x_1, x_2, x_3) \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} = (x_1, x_1, x_2, x_2, x_3, x_3).$$

y las palabras código serán las siguientes:

$$000 \to 000000, \quad 001 \to 000011, \quad 010 \to 001100, \quad 011 \to 001111,$$
$$100 \to 110000, \quad 101 \to 110011, \quad 110 \to 111100, \quad 111 \to 11111.$$

Así, $\mathcal{C} = \{000000, 000011, 001100, 001111, 110000, 110011, 111100, 11111\}$.

Se tiene que la distribución de pesos es $A_0(\mathcal{C}) = A_6(\mathcal{C}) = 1$, $A_2(\mathcal{C}) = A_4(\mathcal{C}) = 3$. Normalmente, solo se listan los A_i no nulos.

Algunos hechos elementales acerca de las *distribuciones de pesos* están recogidos en el siguiente teorema.

Teorema 2.23. *Sea $\mathcal{C}$ un $[n, k, d]$-código lineal sobre $\mathbb{F}_q$. Entonces:*

(i) $A_0(\mathcal{C}) + A_1(\mathcal{C}) + \cdots + A_n(\mathcal{C}) = q^k$.

(ii) $A_0(\mathcal{C}) = 1$ *y* $A_1(\mathcal{C}) = A_2(\mathcal{C}) = \cdots = A_{d-1}(\mathcal{C}) = 0$.

(iii) Si $\mathcal{C}$ es un código binario que contiene la palabra código $\mathbf{1} = 11\cdots1$, entonces $A_i(\mathcal{C}) = A_{n-i}(\mathcal{C})$ para $i \in \{0, \ldots, n\}$.

Demostración.

- La propiedad *(i)* es evidente. Recordemos que el número total de palabras de un $[n, k]$-código lineal sobre $\mathbb{F}_q$ es q^k, y el peso de cada una de las palabras va de 0 hasta n.
- Para la propiedad *(ii)* sabemos que $A_0(\mathcal{C}) = 1$, ya que $\mathbf{0} \in \mathcal{C}$ (por ser un subespacio vectorial), como consecuencia, si se diese que $A_i(\mathcal{C}) \neq 0$ para algún $i \in \{1, \ldots, d-1\}$ existiría una palabra cuyo peso es menor que $d-1$, pero entonces su distancia de Hamming respecto a $\mathbf{0}$ sería menor que d lo que contradice la propia definición de d.
- Por último, veamos la propiedad *(iii)* . Puesto que $\mathcal{C}$ es un subespacio vectorial y $\mathbf{1} \in \mathcal{C}$ se tiene que si $\mathbf{c} \in \mathcal{C}$, entonces $\mathbf{1} - \mathbf{c} \in \mathcal{C}$. Obsérvese que $wt(\mathbf{1} - \mathbf{c}) = n - wt(\mathbf{c})$. De esta manera, para cada palabra código $\mathbf{c}$ de peso i existirá otra palabra código $\mathbf{1} - \mathbf{c}$ de peso $n - i$, lo que concluye la demostración.

□

2.1.3. Codificación de códigos lineales

Sea $\mathcal{C}$ un $[n,k]$-código lineal sobre el cuerpo $\mathbb{F}_q$ con matriz generadora G. Sabemos que este código tiene q^k palabras código, que podemos hacer corresponder con q^k mensajes.

La manera más simple de ver un mensaje es como una k-tupla $\mathbf{m} \in \mathbb{F}_q^k$. Este mensaje lo codificaremos como la palabra código $\mathbf{c} = \mathbf{m}G$. Si G está en forma estándar, las primeras k coordenadas de la palabra código $\mathbf{c}$ son los símbolos de información $\mathbf{m}$,mientras que los $n-k$ símbolos restantes serán los símbolos de paridad, que son símbolos redundantes añadidos a $\mathbf{m}$ que nos ayudarán a recuperar el mensaje si ocurre algún error.

Ejemplo 2.24. Sea $\mathcal{C}$ un $[6,3]$-código binario lineal con la siguiente matriz generadora:

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}.$$

Dado un mensaje $\mathbf{m} = (x_1, x_2, x_3) \in \mathbb{F}_2^3$ lo codificamos utilizando la matriz generadora G.

$$(x_1, x_2, x_3) \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} = (x_1, x_2, x_3, x_1 + x_2, x_2 + x_3, x_1 + x_3).$$

Así, este código codifica de la siguiente forma:

$$\begin{aligned} &000 \to 000000, \quad 001 \to 001011, \quad 010 \to 010110, \quad 011 \to 011101, \\ &100 \to 100101, \quad 101 \to 101110, \quad 110 \to 110011, \quad 111 \to 111000. \end{aligned}$$

Por tanto, $\mathcal{C} = \{000000, 001011, 010110, 011101, 100101, 101110, 110011, 111000\}$.

Puede darse que la matriz generadora G no esté en forma estándar, sin embargo, si existiera un conjunto de índices $i_1, i_2, \ldots, i_k$ correspondientes a las columnas de la matriz G, de manera que la matriz $k \times k$ formada por esas k columnas de G es igual a I_k, se dice que la codificación es *sistemática*. Equivalentemente, una codificación es *sistemática* cuando el mensaje se encuentra incrustado íntegramente en el código (aunque sea de forma desordenada). Se observa que la codificación en el Ejemplo 2.24 es sistemática, ya que las tres primeras coordenadas del mensaje codificado corresponden con el mensaje original.
Sin embargo, no todas las codificaciones son sistemáticas.

Ejemplo 2.25. Sea $\mathcal{C}$ un $[2,4]$-código binario lineal cuya matriz generadora es

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}.$$

Dado un mensaje $\mathbf{m} = (x_1, x_2) \in \mathbb{F}_2^2$ lo codificamos utilizando la matriz generadora G.

$$(x_1, x_2, x_3) \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} = (x_1 + x_2, x_2, x_2, x_1 + x_2).$$

Entonces, este código codifica de la siguiente forma:

$$00 \to 0000, \quad 01 \to 1111, \quad 10 \to 1001, \quad 11 \to 0110.$$

Por tanto, $\mathcal{C} = \{0000, 1111, 1001, 0110\}$.

Observamos en el Ejemplo 2.25 que no existe ningún conjunto de índices i_1, i_2 correspondientes a las columnas de la matriz G, de manera que esas columnas formen la matriz identidad I_2. Además, la codificación del mensaje 01 es 1111 que no contiene al mensaje original. Cuando esto ocurre se dice que la codificación es *no sistemática*.

Acabamos de ver un método para codificar un mensaje $\mathbf{m}$ utilizando la matriz generadora de un código $\mathcal{C}$. En ocasiones, también es posible utilizar la matriz de paridad H para este mismo fin. La forma más simple de hacerlo es cuando la matriz G está en forma estándar $[I_k \mid A]$. En este caso, el Teorema 2.6 nos asegura que $H = \left[-A^T \mid I_{n-k}\right]$. Supongamos que un mensaje $\mathbf{m} = x_1 \cdots x_k$ se codifica en la palabra código $c = c_1 \cdots c_n$. Como G está en forma estándar, $c_1 \cdots c_k = x_1 \cdots x_k$. Por tanto, necesitamos determinar los $n - k$ símbolos de paridad (símbolos redundantes) $c_{k+1} \cdots c_n$. Puesto que $\mathbf{0} = H\mathbf{c}^T = \left[-A^T \mid I_{n-k}\right] \mathbf{c}^T, A^T\mathbf{m}^T = [c_{k+1} \cdots c_n]^T$. Este método se puede generalizar cuando G es un codificador sistemático.

Ejemplo 2.26. Sea $\mathcal{C}$ el mismo código binario que en el Ejemplo 2.24. Podemos utilizar el Teorema 2.6 para encontrar la matriz de paridad H (puesto que G está en forma estándar). Así,

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

Recordemos que el ejemplo anterior, habíamos obtenido que dado un mensaje $\mathbf{m} = (x_1, x_2, x_3) \in \mathbb{F}_2^3$, su codificación era la palabra código $\mathbf{c} = c_1 \cdots c_n = \mathbf{m}G = (x_1, x_2, x_3, x_1 + x_2, x_2 + x_3, x_1 + x_3)$.

Seguimos el proceso descrito para codificar $\mathbf{m}$ a partir de H. Puesto que $\mathbf{0} = H\mathbf{c}$, obtenemos el siguiente sistema:

$$\begin{cases} c_1 + c_2 + c_4 = 0, \\ c_2 + c_3 + c_5 = 0, \\ c_1 + c_3 + c_6 = 0. \end{cases}$$

Como G está en forma estándar, $c_1c_2c_3 = x_1x_2x_3$ y resolviendo este sistema, obtenemos la misma palabra código que con la matriz generadora.

Debido a la correspondencia uno a uno entre mensajes y palabras código, a veces nos puede interesar obtener el mensaje $\mathbf{m}$ a partir de la palabra código $\mathbf{c}$ mediante un proceso inverso a la codificación. Supongamos que $\mathbf{c} = \mathbf{m}G$. Si G está en forma estándar, será fácil recuperar el mensaje original, simplemente tendremos que quedarnos con las primeras k componentes de $\mathbf{c}$. Si G no está en forma estándar, al existir una biyección entre las palabras código y los mensajes, existirá una matriz K de dimensión $n \times k$ de manera que $GK = I_k$. Llamaremos a K la *inversa por la derecha de* G, la cual no es necesariamente única. De esta forma, puesto que $\mathbf{c} = \mathbf{m}G$, $\mathbf{c}K = \mathbf{m}GK = \mathbf{m}$ y se podrá recuperar el mensaje original.

Ejemplo 2.27. Sea $\mathcal{C}$ un $[7,3]$-código lineal dado por la matriz generadora

$$G = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}.$$

Codificamos el mensaje $\mathbf{m} = 110$ con el procedimiento visto anteriormente. Así, $\mathbf{c} = \mathbf{m}G = 0110001$.

Si queremos obtener el mensaje $\mathbf{m}$ a partir de $\mathbf{c}$, en primer lugar calculamos la matriz inversa por la derecha de G, que en este caso es

$$K = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

Así, el mensaje original vendrá dado por

$$\mathbf{c}K = (0,1,1,0,0,0,1) \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} = (1,1,0) = \mathbf{m}.$$

2.2. Códigos cíclicos

En esta sección estudiaremos una clase destacada de códigos: los *códigos cíclicos*. Estos códigos fueron introducidos en el año 1957 por Eugene Prange en [31], que se percató de que los códigos cíclicos tenían una rica estructura algebraica. Esto sugirió que el álgebra podía desempeñar un papel crucial en la teoría de códigos. Los códigos cíclicos fueron unos de los primeros códigos

usados en la práctica, ya que era fácil implementarlos utilizando registros de desplazamiento. La principal fuente consultada para esta sección ha sido [15].

Definición 2.28. Un código lineal $\mathcal{C}$ de longitud n sobre $\mathbb{F}_q$ es *cíclico* si para cada palabra código $\mathbf{c} = c_0 \cdots c_{n-2}c_{n-1}$ en $\mathcal{C}$, el vector $c_{n-1}c_0 \cdots c_{n-2}$, obtenido mediante un desplazamiento cíclico de $\mathbf{c}$, llevando $i \to i+1 \bmod n$, está también en $\mathcal{C}$.

Por tanto, dado un elemento $\mathbf{c}$ de un código cíclico $\mathcal{C}$, se tiene que los n desplazamientos cíclicos de $\mathbf{c}$ también están en $\mathcal{C}$.

Normalmente, al estudiar los códigos cíclicos sobre $\mathbb{F}_q$ representaremos las palabras código en forma polinomial. Esto se debe a que existe una correspondencia biyectiva entre los vectores $\mathbf{c} = c_1 \cdots c_{n-1} \in \mathbb{F}_q^n$ y los polinomios $c(x) = c_0 + c_1x + \cdots c_{n-1}x^{n-1} \in \mathbb{F}_q[x]$ de grado máximo $n-1$. Nótese que si $c(x) = c_0 + c_1x + \cdots c_{n-1}x^{n-1}$, entonces $xc(x) = c_{n-1}x^n + c_0x + c_1x^2 + \cdots c_{n-2}x^{n-1}$, que representaría a la palabra código $\mathbf{c}$ desplazada cíclicamente una posición a la derecha si x^n fuese igual a 1.

Formalmente, el hecho de que un código $\mathcal{C}$ sea invariante bajo un desplazamiento cíclico, implica que si $c(x)$ está en $\mathcal{C}$, entonces $xc(x)$ también lo está, siempre que multipliquemos módulo $x^n - 1$. Esto nos sugiere que el contexto adecuado para el estudio de los códigos cíclicos es el anillo cociente

$$\mathcal{R}_n = \mathbb{F}_q[x]/\langle x^n - 1\rangle.$$

De esta manera, bajo la correspondencia de vectores con polinomios descrita anteriormente, los códigos cíclicos son *ideales* de $\mathcal{R}_n$ y los ideales de $\mathcal{R}_n$ son códigos cíclicos. Así, el estudio de los códigos cíclicos en $\mathbb{F}_q^n$ es equivalente al estudio de los ideales en $\mathcal{R}_n$, que depende de la factorización de $x^n - 1$, que estudiaremos a continuación.

2.2.1. Factorización de $x^n - 1$

Nuestro primer objetivo será encontrar los factores irreducibles de $x^n - 1$ sobre $\mathbb{F}_q$. Existen dos posibilidades: que $x^n - 1$ tenga factores irreducibles repetidos o que no los tenga. Para el estudio de los códigos cíclicos nos centraremos en la segunda posibilidad. La siguiente proposición nos da una condición para que $x^n - 1$ sobre $\mathbb{F}_q$ no tenga factores irreducibles repetidos.

Proposición 2.29. *Equivalen las siguientes afirmaciones:*

(i) $x^n - 1$ *no tiene factores irreducibles repetidos en* $\mathbb{F}_q$.

(ii) $mcd(q, n) = 1$.

Demostración.

(i) ⇒ (ii)

Supongamos que $x^n - 1$ no tiene factores irreducibles repetidos sobre $\mathbb{F}_q$. Por reducción al absurdo, si $mcd(q, n) \neq 1$, entonces $n = q^e m$ para algún $e > 0$, $q \nmid m$. De esta forma,

$$x^n - 1 = x^{q^e m} - 1 = (x^m)^{q^e} - 1 = (x^m - 1)^{q^e},$$

que claramente tiene factores irreducibles repetidos, lo cual supone una contradicción. Se ha utilizado la Proposición 1.19.

(ii) ⇒ (i)

Supongamos ahora que $mcd(q, n) = 1$. Sea I el ideal de $\mathbb{F}_q[x]$ generado por los elementos $x^n - 1$ y nx^{n-1}. Por hipótesis, como $mcd(q, n) = 1$ se tiene que n, visto como un elemento de $\mathbb{F}_q$, es una unidad. Por tanto,

$$nx^{n-1} \in I \Rightarrow x^{n-1} \in I \Rightarrow x^n \in I \Rightarrow x^n - (x^n - 1) \in I \Rightarrow 1 \in I \Rightarrow I = (1) = \mathbb{F}_q[x].$$

Por tanto, se tiene que en $\mathbb{F}_q[x]$, $mcd(x^n - 1, nx^{n-1}) = 1$, ya que el ideal generado por ambos es el trivial. Recordemos que un polinomio $f \in \mathbb{F}_q[x]$ no tiene factores irreducibles repetidos si y solo si $mcd(f, f') = 1$. Utilizando este último resultado, donde $f = x^n - 1$, se concluye la demostración.

□

Para factorizar $x^n - 1$ sobre $\mathbb{F}_q$ necesitamos encontrar una extensión de cuerpos $\mathbb{F}_{q^t}$ que contenga todas sus raíces. Dicho de otra forma, $\mathbb{F}_{q^t}$ debe contener una *raíz n-ésima primitiva de la unidad*, que en virtud del Teorema 1.27, esto ocurrirá si y solo si $n \mid (q^t - 1)$. En tal caso, $\gamma^{(q^t-1)/n}$ es dicha raíz.

Definición 2.30. El *orden* de q módulo n, denotado por $ord_n(q)$, es el menor entero positivo a tal que $q^a \equiv 1 \pmod{n}$.

Nótese que si $t = ord_n(q)$, entonces $\mathbb{F}_{q^t}$ contiene una raíz n-ésima primitiva de la unidad α, puesto que por la definición de orden, $q^t \equiv 1 \pmod{n} \Leftrightarrow n \mid (q^t - 1)$. Además, ninguna extensión de cuerpos más pequeña contendrá una raíz primitiva de la unidad, al ser t el menor entero que cumple dicha condición. Como los α^i son distintos para $i \in \{0, \ldots, n\}$ y $(\alpha^i)^n = 1$, $\mathbb{F}_{q^t}$ contiene todas las raíces de $x^n - 1$. En consecuencia, $\mathbb{F}_{q^t}$ es un *cuerpo de descomposición* [1] de $x^n - 1$ sobre $\mathbb{F}_q$. De esta manera, los factores irreducibles de $x^n - 1$ sobre $\mathbb{F}_q$ deben ser el producto de los distintos polinomios minimales de las n-ésimas raíces de la unidad en $\mathbb{F}_{q^t}$. Supongamos que γ es un elemento primitivo de $\mathbb{F}_{q^t}$. Entonces $\alpha = \gamma^d$ es una raíz n-ésima primitiva de la unidad, donde $d = (q^t - 1)/n$. Las raíces de $M_{\alpha^s}(x)$ son

$$\{\gamma^{ds}, \gamma^{dsq}, \ldots, \gamma^{dsq^{r-1}}\} = \{\alpha^s, \alpha^{sq}, \ldots, \alpha^{sq^{r-1}}\},$$

donde r es el entero positivo más pequeño tal que $dsq^r \equiv ds \pmod{q^t - 1}$ por el Teorema 1.38. Pero $dsq^r \equiv ds \pmod{q^t - 1}$ si y solo si $sq^r \equiv s \pmod{n}$.

[1] Es decir, un cuerpo donde $x^n - 1$ se factoriza mediante polinomios de grado uno.

Esto nos lleva a extender la definición de clases q-ciclotómicas desarrolladas en la Sección 1.2.5.

Sea s un entero, con $s \in \{0, \ldots, n-1\}$. La clase q-ciclotómica de s módulo n es el conjunto

$$C_s = \{s, sq, \ldots, sq^{r-1}\} \pmod{n},$$

donde r es el entero positivo más pequeño tal que $sq^r \equiv s \pmod{n}$. En la Sección 1.2.5 habíamos estudiado un caso más restrictivo, donde $n = q^t - 1$. Sin embargo, este concepto puede generalizarse a un n arbitrario.

Teorema 2.31. *Sea n un entero positivo coprimo con q. Sea $t = ord_n(q)$. Sea α una n-ésima raíz primitiva de la unidad en $\mathbb{F}_{q^t}$.*

(i) Para cada entero $s \in \{0, \ldots, n-1\}$, el polinomio minimal de α^s en $\mathbb{F}_q$ es

$$M_{\alpha^s}(x) = \prod_{i \in C_s} (x - \alpha^i),$$

donde C_s es la clase q-ciclotómica de s módulo n.

(ii) Los conjugados[2] de α^s son los elementos α^i con $i \in C_s$.

(iii) Además,

$$x^n - 1 = \prod_s M_{\alpha^s}(x)$$

es la factorización de $x^n - 1$ en factores irreducibles en $\mathbb{F}_q$, donde s varía en un conjunto de representantes de las clases q-ciclotómicas módulo n.

2.2.2. Teoría básica de códigos cíclicos

Recordemos que los códigos cíclicos sobre $\mathbb{F}_q$ son precisamente los ideales de

$$\mathcal{R}_n = \mathbb{F}_q[x]/\langle x^n - 1\rangle.$$

La Proposición 1.26 nos afirmaba que $\mathbb{F}_q[x]$ era un dominio de ideales principales. Veremos más adelante que los ideales de $\mathcal{R}_n$ también son principales, y, por tanto, los códigos cíclicos son los ideales principales de $\mathcal{R}_n$.

La manera más formal de escribir un código cíclico es como un elemento $c(x) + (x^n - 1)$ del ideal $\mathcal{R}_n$. Sin embargo, para simplificar la notación escribiremos $c(x)$. Así, pensaremos en los elementos de $\mathcal{R}_n$ como polinomios en $\mathbb{F}_q[x]$ en los que los términos de la forma ax^{ni+j}, $j \in \{0, \ldots, n-1\}$, $i \in \mathbb{N}$, se cambiarán por ax^j.

Para distinguir el ideal principal $(g(x))$ de $\mathbb{F}_q[x]$ del mismo ideal en $\mathcal{R}_n$, usaremos la notación $\langle g(x)\rangle$ para el ideal de $\mathcal{R}_n$ generado por $g(x)$.

[2] Recordemos que dos elementos son conjugados en $\mathbb{F}_q$ si tienen el mismo polinomio minimal en $\mathbb{F}_q[x]$.

El siguiente teorema nos permitirá describir los códigos cíclicos, ya que nos afirma que hay una correspondencia biyectiva entre los códigos cíclicos en $\mathcal{R}_n$ y los polinomios mónicos que dividen a $x^n - 1$.

Teorema 2.32. *Sea $\mathcal{C}$ un código cíclico no nulo en $\mathcal{R}_n$. Entonces existe un polinomio $g(x) \in \mathcal{C}$ que verifica las siguientes propiedades:*

(i) $g(x)$ *es el único polinomio mónico de grado mínimo en* $\mathcal{C}$.

(ii) El polinomio $g(x)$ *genera* $\mathcal{C}$. *Es decir,* $\mathcal{C} = \langle g(x) \rangle$.

(iii) El polinomio $g(x)$ *divide a* $x^n - 1$.

Sea $k = n - \operatorname{gr}(g(x))$ *y sea* $g(x) = \sum_{i=0}^{n-k} g_i x^i$ *donde* $g_{n-k} = 1$. *Entonces:*

(iv) La dimensión de $\mathcal{C}$ *es* k *y* $\{g(x), xg(x), \ldots, x^{k-1}g(x)\}$ *es una base de* $\mathcal{C}$.

(v) Todo elemento de $\mathcal{C}$ *se puede escribir de manera única como el producto de* $g(x)f(x)$, *donde* $f(x) = 0$ *o* $\operatorname{gr}(f(x)) < k$.

(vi) Una matriz generadora de $\mathcal{C}$ *es*

$$G = \begin{bmatrix} g_0 & g_1 & \cdots & g_{n-k} & & & \bigcirc \\ & g_0 & g_1 & \cdots & g_{n-k} & & \\ & & \ddots & \ddots & & \ddots & \\ \bigcirc & & & g_0 & g_1 & \cdots & g_{n-k} \end{bmatrix},$$

donde cada fila es un desplazamiento cíclico de la fila anterior.

(vii) Si α *es una raíz n-ésima primitiva de la unidad en alguna extensión de* $\mathbb{F}_q$, *entonces*

$$g(x) = \prod_s M_{\alpha^s}(x),$$

siendo el producto sobre un subconjunto de representaciones de las clases q-ciclotómicas módulo n.

Demostración.

(i) y (ii)

Sea $g(x)$ un polinomio mónico de grado mínimo en $\mathcal{C}$, que existe por ser $\mathcal{C}$ no nulo. Si $c(x) \in \mathcal{C}$, entonces, por el algoritmo de la división en $\mathbb{F}_q[x]$ 1.24, $c(x) = g(x)h(x) + r(x)$, donde o bien $r(x) = 0$ o $\operatorname{gr}(r(x)) < \operatorname{gr}(g(x))$. Como $\mathcal{C}$ es un ideal de $\mathcal{R}_n$, $r(x) \in \mathcal{C}$ y al tener $g(x)$ grado mínimo, se deduce que $r(x) = 0$. Por tanto, $\mathcal{C} = \langle g(x) \rangle$, lo que prueba *(ii)*. Si existiese otro polinomio $q(x)$ mónico de grado mínimo en $\mathcal{C}$ se tiene que $q(x) = g(x)h(x)$. Como $q(x)$ es mónico y del mismo grado que $g(x)$ necesariamente $h(x) = 1$ y, por tanto, $g(x) = q(x)$, lo que prueba *(i)*.

(iii)

Utilizando de nuevo el algoritmo de la división, $x^n - 1 = g(x)h(x) + r(x)$, donde $r(x) = 0$ o $\mathrm{gr}(r(x)) < \mathrm{gr}(g(x))$. Como en $\mathcal{R}_n$ se verifica que $x^n - 1 = 0 \in \mathcal{C}$, se tiene que $r(x) \in \mathcal{C}$, lo que es una contradicción a menos que $r(x) = 0$. En consecuencia, $g(x) \mid x^n - 1$.

(iv) y (v)

Supongamos que $\mathrm{gr}(g(x)) = n - k$. Por los apartados *(ii) y (iii)*, si $c(x) \in \mathcal{C}$ con $c(x) = 0$ o $\mathrm{gr}(c(x)) < n$, entonces $c(x) = g(x)f(x)$ para algún polinomio $f(x)$ en $\mathbb{F}_q[x]$. Si $c(x) = 0$, $f(x) = 0$. Por otro lado, $\mathrm{gr}(c(x)) < n$ implica que $\mathrm{gr}(f(x)) < k$. Entonces,

$$\mathcal{C} = \{g(x)f(x) \mid f(x) = 0 \text{ ó } \mathrm{gr}(f(x)) < k\}.$$

Por tanto, $\mathcal{C}$ tiene dimensión al menos k y $\{g(x), xg(x), \ldots, x^{k-1}g(x)\}$ es un sistema de generadores de $\mathcal{C}$. Como estos k polinomios tienen grados diferentes, son independientes en $\mathbb{F}_q[x]$. Además, tienen un grado estrictamente menor que n y, por tanto, permanecen iguales en $\mathcal{R}_n$. Esto prueba que $\{g(x), xg(x), \ldots, x^{k-1}g(x)\}$ es una base de $\mathcal{C}$ y que su dimensión es exactamente k, lo que nos lleva a *(iv)*. De la unicidad de $g(x)$ se obtiene que la descomposición $c(x) = g(x)f(x)$ es única, lo que prueba *(v)*.

(vi)

La matriz G es matriz generadora de $\mathcal{C}$, ya que $\{g(x), xg(x), \ldots, x^{k-1}g(x)\}$ es una base de $\mathcal{C}$.

(vii)

Se deduce del Teorema 2.31 y de *(iii)*. □

Observemos que *(ii)* prueba que $\mathcal{R}_n$ es un dominio de ideales principales, pues todos sus ideales (es decir, los códigos cíclicos) pueden ser generados por un único elemento.

Corolario 2.33. *Sea $\mathcal{C}$ un código cíclico no nulo en $\mathcal{R}_n$. Equivalen las siguientes afirmaciones;*

(i) *$g(x)$ es el polinomio mónico de menor grado en $\mathcal{C}$.*

(ii) *$\mathcal{C} = \langle g(x) \rangle$, $g(x)$ es mónico y divide a $x^n - 1$.*

Demostración.
(i) ⇒ (ii) Probado en el Teorema 2.32.

(ii) ⇒ (i)

Sea $g_1(x)$ el polinomio mónico de menor grado en $\mathcal{C}$. Como se ha visto en la demostración del Teorema 2.32, $g_1(x) \mid g(x)$ en $\mathbb{F}_q[x]$ y $\mathcal{C} = \langle g_1(x) \rangle$. Puesto que $g_1(x) \in \mathcal{C} = \langle g(x) \rangle$, se tiene que

$$g_1(x) \equiv g(x)a(x) \pmod{x^n - 1}.$$

También se tiene que $g_1(x) = g(x)a(x) + (x^n - 1)b(x)$ en $\mathbb{F}_q[x]$. Dado que $g(x) \mid (x^n - 1)$, entonces $g(x)$ divide a $g_1(x)$. Teniendo en cuenta que tanto $g_1(x)$ como $g(x)$ son mónicos y se dividen mutuamente en $\mathbb{F}_q[x]$, necesariamente son iguales.

□

El Teorema 2.32 nos dice que existe un polinomio mónico $g(x)$ que divide a $x^n - 1$ y genera $\mathcal{C}$. El Corolario 2.33 asegura que $g(x)$ es único. A este polinomio le llamamos *el polinomio generador* del código cíclico $\mathcal{C}$. Por el Corolario 2.33, este polinomio es tanto el polinomio mónico en $\mathcal{C}$ de menor grado, como el polinomio mónico que divide a $x^n - 1$ y genera $\mathcal{C}$. Por tanto, existe una biyección entre los códigos cíclicos no nulos y los divisores propios[3] de $x^n - 1$. Para obtener la biyección entre el conjunto de los códigos cíclicos en $\mathcal{R}_n$ y el conjunto de divisores mónicos de $x^n - 1$,definimos el polinomio generador del código cíclico nulo $\{\mathbf{0}\}$ como $x^n - 1$, es decir, $\langle x^n - 1\rangle = \{\mathbf{0}\}$ (recordemos que $x^n - 1 = 0$ en $\mathcal{R}_n$). Esta biyección nos conduce al siguiente corolario.

Corolario 2.34. *El número de códigos cíclicos en $\mathcal{R}_n$ es igual a 2^m, siendo m el número de clases q-ciclotómicas módulo n.*

Por tanto, si $g(x)$ es un polinomio de grado $n - k$ que divide a $x^n - 1$ en $\mathbb{F}_q[x]$, generará un $[n, k]$-código cíclico. Veamos un ejemplo de algún código cíclico.

Ejemplo 2.35. Dado el polinomio $x^9 - 1$, podemos factorizarlo en polinomios irreducibles en $\mathbb{F}_2[x]$ como $x^9 - 1 = (1 + x)(1 + x + x^2)(1 + x^3 + x^6)$.

Todos los divisores de $x^9 - 1$ generarán un código cíclico. En particular, veamos los códigos cíclicos $\langle 1 + x\rangle$ y $\langle 1 + x + x^2\rangle$.

- $\langle 1 + x\rangle$: La dimensión del código es $k = 9 - 1 = 8$ y este polinomio genera un $[9, 8]$-código cíclico. Haciendo uso del Teorema 2.32 podemos obtener la matriz generadora del código:

$$G = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

- $\langle 1 + x + x^2\rangle$: La dimensión del código es $k = 9 - 2 = 7$. Por tanto, este polinomio genera un $[9, 8]$-código cíclico. Haciendo uso del Teorema 2.32 podemos obtener la matriz

[3] Entendemos por divisores propios de un polinomio al conjunto de todos sus divisores salvo él mismo.

generadora del código:

$$G = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

Los códigos cíclicos se pueden caracterizar a partir de los ceros del polinomio $x^n - 1$, es decir, a partir de las raíces n-ésimas de la unidad. Esta caracterización nos será de utilidad cuando estudiemos los códigos BCH.

Recordemos que el Teorema 2.31 nos dice que si $t = ord_n(q)$, entonces $\mathbb{F}_{q^t}$ es un cuerpo de descomposición del polinomio $x^n - 1$. Así, $\mathbb{F}_{q^t}$ contiene una raíz n-ésima primitiva de la unidad α y $x^n - 1 = \prod_{i=0}^{n-1}(x - \alpha^i)$ es la factorización de $x^n - 1$ en $\mathbb{F}_{q^t}$ mediante polinomios de grado uno. Además, $x^n - 1 = \prod_s M_{\alpha^s}(x)$ es la factorización de $x^n - 1$ utilizando polinomios irreducibles en $\mathbb{F}_q$, donde s varía en un conjunto de representantes de las clases q-ciclotómicas módulo n.

Sea $\mathcal{C}$ un código cíclico en $\mathcal{R}_n$ generado por el polinomio $g(x)$. Por los Teoremas 2.31 y 2.32, el polinomio $g(x)$ se puede factorizar de la siguiente manera:

$$g(x) = \prod_s M_{\alpha^s}(x) = \prod_s \prod_{i \in C_s} (x - \alpha^i), \tag{2.1}$$

donde s, de nuevo, varía en un conjunto de representantes de las clases q-ciclotómicas módulo n. Sea $T = \bigcup_s C_s$ la unión de estas clases q-ciclotómicas, las raíces de la unidad $\mathcal{Z} = \{\alpha^i \mid i \in T\}$ se llaman *ceros* del código cíclico $\mathcal{C}$. El conjunto T se denomina *conjunto característico* de $\mathcal{C}$.

El siguiente teorema nos proporciona propiedades básicas de los códigos cíclicos a partir de sus conjuntos característicos.

Teorema 2.36. *Sea α una raíz n-ésima primitiva de la unidad en algún cuerpo de descomposición de $\mathbb{F}_q$ con conjunto característico T y polinomio generador $g(x)$. Se verifican las siguientes afirmaciones:*

(i) El polinomio $g(x)$ se factoriza como $g(x) = \prod_{i \in T}(x - \alpha^i)$.

(ii) La palabra código $c(x) \in \mathcal{R}_n$ está en $\mathcal{C}$ si y solo si $c(\alpha^i) = 0$, $\forall i \in T$.

(iii) La dimensión de $\mathcal{C}$ es $n - |T|$.

Demostración.

- La propiedad *(i)* se deduce fácilmente de (2.1) teniendo en cuenta que $T = \bigcup_s C_s$.
- Recordemos que por el Teorema 2.32, $c(x)$ es múltiplo de $g(x)$. Además, este mismo teorema nos asegura que $g(\alpha^i) = 0$, $\forall i \in T$, lo que prueba *(ii)*.

- La prueba de *(iii)* es directa utilizando, de nuevo, el Teorema 2.32, ya que por *(i)*, $|T|$ es igual al grado del polinomio $g(x)$.

□

2.3. Códigos BCH

En esta sección estudiaremos una de las familias más importantes de códigos cíclicos: los códigos BCH. También estudiaremos una subfamilia de estos códigos, los códigos Reed-Solomon. Los códigos BCH fueron diseñados para aprovechar la *cota de Bose-Ray-Chaudhuri-Hocquenghem*, abreviada como *cota BCH*. La cota BCH dependerá de los ceros del código y, en particular, de la posibilidad de encontrar cadenas de ceros "consecutivas".

En lo que sigue, consideraremos que $\mathcal{C}$ es un código cíclico de longitud n en $\mathbb{F}_q$ y que α es una raíz n-ésima primitiva de la unidad en $\mathbb{F}_{q^t}$, donde $t = ord_n(q)$. Recordemos que T es un conjunto característico de $\mathcal{C}$ siempre y cuando los ceros de $\mathcal{C}$ sean $\{\alpha^i \mid i \in T\}$. Por tanto, T debe ser una unión de clases q-ciclotómicas módulo n. Diremos que T contiene un subconjunto consecutivo de s elementos $\mathcal{S}$ si existe un conjunto $\{b, b+1, \ldots, b+s+1\}$ de s elementos consecutivos tal que

$$\{b, b+1, \ldots, b+s+1\} \mod n = \mathcal{S} \subseteq T.$$

Antes de proceder con la cota BCH, enunciaremos un lema conocido que usaremos en la demostración de la cota.

Sean $\alpha_1, \ldots, \alpha_s$ elementos de un cuerpo $\mathbb{F}$. Una matriz de la forma

$$V = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \alpha_1 & \alpha_2 & \cdots & \alpha_s \\ \alpha_1^2 & \alpha_2^2 & \cdots & \alpha_s^2 \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_1^{s-1} & \alpha_2^{s-1} & \cdots & \alpha_s^{s-1} \end{pmatrix}$$

se llama *matriz de Vandermonde* [4].

Lema 2.37. *El determinante de una matriz de Vandermonde viene dado por*

$$\det(V) = \prod_{1 \leq i < j \leq s} (\alpha_j - \alpha_i).$$

En particular, V es regular si los elementos $\alpha_1, \ldots, \alpha_s$ son distintos.

Ya tenemos las herramientas necesarias para demostrar el teorema de la cota BCH.

Teorema 2.38 (Cota BCH)**.** *Sea $\mathcal{C}$ un código cíclico de longitud n sobre $\mathbb{F}_q$ y sea T su conjunto característico. Supongamos que $\mathcal{C}$ tiene peso mínimo d y que T contiene $\delta - 1$ elementos consecutivos para algún entero δ. Entonces $d \geq \delta$.*

[4] A la transpuesta de V también se le conoce como matriz de Vandermonde.

Demostración. Sean $\alpha^b, \alpha^{b+1}, \ldots, \alpha^{b+\delta-2}$ los $\delta - 1$ ceros consecutivos de $\mathcal{C}$. Sea $c(x)$ una palabra código no nula en $\mathcal{C}$ cuyo peso de Hamming es w, de la forma

$$c(x) = \sum_{j=1}^{w} c_{i_j} x^{i_j}.$$

Supongamos que $w < \delta$. Como $c(\alpha^i) = 0$ para $i \in \{b, b+1, \ldots, b+\delta-2\}$, se tiene que $M\mathbf{u}^T = \mathbf{0}$, con

$$M = \begin{bmatrix} \alpha^{i_1 b} & \alpha^{i_2 b} & \cdots & \alpha^{i_w b} \\ \alpha^{i_1 (b+1)} & \alpha^{i_2 (b+1)} & \cdots & \alpha^{i_w (b+1)} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha^{i_1 (b+w-1)} & \alpha^{i_2 (b+w-1)} & \cdots & \alpha^{i_w (b+w-1)} \end{bmatrix}$$

y $\mathbf{u} = (c_{i_1}, c_{i_2}, \ldots, c_{i_w})$. Puesto que $\mathbf{u} \neq \mathbf{0}$, M es una matriz singular, y, por tanto, $\det(M) = 0$. Sin embargo, $\det(M) = \alpha^{(i_1+i_2+\cdots i_w)b} \det(V)$, donde V es la matriz de Vandermonde

$$V = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \alpha^{i_1} & \alpha^{i_2} & \cdots & \alpha^{i_w} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha^{i_1 (w-1)} & \alpha^{i_2 (w-1)} & \cdots & \alpha^{i_w (w-1)} \end{bmatrix}.$$

Como los α^{i_j} son distintos, $\det(V) \neq 0$ por el Lema 2.37, lo que contradice que $\det(M) = 0$. □

Recordemos que la efectividad de un código $\mathcal{C}$ a la hora de corregir errores depende de su dimensión y de su peso mínimo, los cuales nos gustaría maximizar. Tener un peso mínimo grande, basándonos en la cota BCH, puede conseguirse eligiendo un conjunto característico T de $\mathcal{C}$ con un gran número de elementos consecutivos. Puesto que por el Teorema 2.36 la dimensión de $\mathcal{C}$ es $n - |T|$, lo ideal sería que $|T|$ sea lo más pequeño posible. Así, si queremos que $\mathcal{C}$ tenga distancia mínima de al menos δ, podemos elegir un conjunto característico tan pequeño como sea posible que sea unión de clases q-ciclotómicas con $\delta - 1$ elementos consecutivos.

Definición 2.39. Sea $\delta \in \{2, \ldots, n\}$. Un *código BCH* $\mathcal{C}$ sobre $\mathbb{F}_q$ de longitud n y distancia mínima designada δ es un código cíclico con conjunto característico

$$T = C_b \cup C_{b+1} \cup \cdots \cup C_{b+\delta-2}, \tag{2.2}$$

donde C_i es la clase q-ciclotómica módulo n que contiene a i.

Teorema 2.40. *Un código BCH de distancia mínima designada δ tiene peso mínimo de al menos δ.*

Demostración. El conjunto característico (2.2) contiene $\delta - 1$ elementos consecutivos. El resultado se sigue de la cota BCH 2.38. □

2.3.1. Códigos Reed-Solomon

Una subfamilia importante de los códigos BCH son los códigos Reed-Solomon, que abreviaremos como códigos RS.

Definición 2.41. *Un código Reed-Solomon (RS)* sobre $\mathbb{F}_q$ es un código BCH de longitud $n = q - 1$.

De este modo, $ord_n(q) = 1$ implica que todos los factores irreducibles de $x^n - 1$ son de grado uno y todas las clases q-ciclotómicas módulo n tienen tamaño 1. El siguiente teorema nos da algunas propiedades de los códigos RS.

Teorema 2.42. *Sea $\mathcal{C}$ un código RS sobre $\mathbb{F}_q$ de longitud $n = q - 1$ y distancia mínima designada δ. Entonces:*

(i) El código $\mathcal{C}$ tiene un conjunto característico de la forma $T = \{b, b+1, \ldots, b+\delta-2\}$.

(ii) El código $\mathcal{C}$ tiene distancia mínima $d = \delta$ y dimensión $k = n - d + 1$.

Demostración. Veamos la demostración por apartados.

- La propiedad *(i)* se deduce teniendo en cuenta que al ser un código BCH de distancia mínima designada δ, su conjunto característico T tiene que tener tamaño $\delta - 1$ y en consecuencia $T = \{b, b+1, \ldots, b+\delta-2\}$.
- El Teorema 2.36 nos dice que la dimensión de $\mathcal{C}$ es $k = n - |T| = n - \delta + 1$. Además, la cota de Singleton 2.20 y el Teorema 2.40 nos aseguran que $k = n - \delta + 1 \geq n - d + 1 \geq k$, y, por tanto, $k = n - d + 1$, lo que prueba *(ii)*.

□

2.4. Decodificación de códigos BCH con el algoritmo de Sugiyama

En esta sección vamos a ver un algoritmo de decodificación para códigos BCH, el conocido *algoritmo de Sugiyama*. Este algoritmo fue creado por Sugiyama, Kasahara, Hirasawa y Namekawa en 1975 [35]. Se diseñó para una clase de códigos llamada la clase de los códigos de Goopa, los cuales incluyen a los códigos BCH como una subclase suya. Sin embargo, en esta sección solo lo aplicaremos a códigos BCH. Este algoritmo es una aplicación del algoritmo de Euclides para polinomios 1.25.

Sea $\mathcal{C}$ un código BCH sobre $\mathbb{F}_q$ de longitud n, distancia designada δ y conjunto característico T, con $\{b, b+1, \ldots, b+\delta-2\} \subseteq T$. Puesto que la distancia mínima de $\mathcal{C}$ es al menos δ, por el Teorema 2.16, $\mathcal{C}$ puede corregir hasta $t = \left\lfloor \frac{\delta-1}{2} \right\rfloor$ errores. El algoritmo de Sugiyama nos permitirá corregir estos t errores.

Como en cualquier algoritmo de decodificación, si suponemos que recibimos un mensaje $y(x)$, nosotros queremos obtener la palabra código original $c(x)$. Supondremos que $y(x)$ difiere de $c(x)$ a lo sumo en t coordenadas. Por tanto,

$$y(x) = c(x) + e(x),$$

donde $e(x)$ es el *vector de errores* cuyo peso de Hamming es $\nu \leq t$. Supongamos que los errores ocurren en unas coordenadas desconocidas $k_1, k_2, \ldots, k_\nu$. Entonces:

$$e(x) = e_{k_1} x^{k_1} + e_{k_2} x^{k_2} + \cdots e_{k_\nu} x^{k_\nu} \tag{2.3}$$

Una vez determinemos el vector de errores $e(x)$, lo que equivale a encontrar las coordenadas de error k_j y las magnitudes del error e_{k_j}, $j \in \{1, \ldots, \nu\}$, podremos decodificar el vector recibido como $c(x) = y(x) - e(x)$. Recordemos que por el Teorema 2.36, $c(x) \in \mathcal{C}$ si y solo si $c(\alpha^i) = 0$ para todo $i \in T$. En particular, dado que $t = \left\lfloor \frac{\delta - 1}{2} \right\rfloor$ y que T contiene a $\{b, b+1, \ldots, b+\delta-2\}$, se tiene que

$$y(\alpha^i) = c(\alpha^i) + e(\alpha^i) = e(\alpha^i), \quad \forall i \in \{b, b+1, \ldots, b+2t-1\}.$$

Para $i \in \{b, b+1, \ldots, b+2t-1\}$ definimos *el síndrome* S_i de $y(x)$ como el elemento $S_i = y(\alpha^i)$ en $\mathbb{F}_{q^m}$.

El primer paso del algoritmo de Sugiyama será calcular los síndromes para un vector $y(x)$ que se ha recibido. Estos síndromes nos conducen a un sistema de ecuaciones que involucran tanto a las coordenadas de error desconocidas como a las magnitudes de error. Nótese que de (2.3) los síndromes satisfacen

$$S_i = y(\alpha^i) = e(\alpha^i) = \sum_{j=1}^{\nu} e_{k_j} (\alpha^i)^{k_j} = \sum_{j=1}^{\nu} e_{k_j} (\alpha^{k_j})^i, \quad i \in \{b, b+1, \ldots, b+2t-1\}. \tag{2.4}$$

Con el objetivo de simplificar la notación, para $j \in \{1, \ldots, \nu\}$ definimos:

- $E_j = e_{k_j}$, que será la magnitud del error en la coordenada k_j.
- $X_j = \alpha^{k_j}$, que llamaremos número de coordenadas de error correspondiente a la coordenada de error k_j.

Por el Teorema 1.27, si $\alpha^i = \alpha^k$ con $i \in \{0, \ldots, n-1\}$, entonces $i = k$. Por tanto, conociendo X_j conoceremos también la coordenada de error k_j. Con la notación anterior, (2.4) se puede expresar como:

$$S_i = \sum_{j=1}^{\nu} E_j X_j^i, \quad i \in \{b, b+1 \ldots, b+2t-1\}.$$

La expresión anterior nos conduce al sistema de ecuaciones

$$\begin{cases} S_b = E_1 X_1^b + E_2 X_2^b + \cdots + E_\nu X_\nu^b, \\ S_{b+1} = E_1 X_1^{b+1} + E_2 X_2^{b+1} + \cdots + E_\nu X_\nu^{b+1}, \\ S_{b+2} = E_1 X_1^{b+2} + E_2 X_2^{b+2} + \cdots + E_\nu X_\nu^{b+2}, \\ \qquad\qquad \vdots \\ S_{b+2t-1} = E_1 X_1^{b+2t-1} + E_2 X_2^{b+2t-1} + \cdots + E_\nu X_\nu^{b+2t-1}. \end{cases} \tag{2.5}$$

Este sistema es no lineal para los X_j, y además, desconocemos tanto los valores de los E_j como de los X_j. La estrategia será calcular directamente los números de coordenadas de error. Una vez conozcamos los X_j, volveremos al sistema (2.5), que será un sistema lineal formado únicamente por los E_j. Para este fin, definimos el *polinomio localizador de errores* como

$$\sigma(x) = \prod_{j=1}^{\nu}(1 - xX_j) = (1 - xX_1)(1 - xX_2)\cdots(1 - xX_\nu) = 1 + \sum_{i=1}^{\nu} \sigma_i x^i.$$

Nótese que $\sigma(x) = 0$ si y solo si $x = X_j^{-1}$, $j \in \{1, \ldots, \nu\}$. Por tanto, calculando las raíces del polinomio localizador de errores, calcularemos los inversos de los números de coordenadas de error, y podremos obtener los X_j fácilmente.

Definición 2.43. El *polinomio evaluador de errores* se define como

$$\omega(x) = \sum_{j=1}^{\nu} E_j X_j^b \prod_{\substack{i=1 \\ i\neq j}}^{\nu}(1 - xX_i) = \sum_{j=1}^{\nu} E_j X_j^b \frac{\sigma(x)}{1 - xX_j}.$$

Observamos que $\mathrm{gr}(\sigma(x)) = \nu$ y $\mathrm{gr}(\omega(x)) \leq \nu - 1$. Ahora definimos el polinomio $S(x)$ de grado máximo $2t - 1$ como

$$S(x) = \sum_{i=0}^{2t-1} S_{i+b} x^i,$$

donde los S_i son los síndromes del vector recibido, $i \in \{b, b+1, \ldots, b+2t-1\}$.

Teorema 2.44. *Se cumple la siguiente relación*

$$\omega(x) \equiv \sigma(x)S(x) \mod x^{2t}.$$

Demostración. Obsérvese que

$$(1 - ax)\left(\sum_{i=0}^{m-1}(ax)^i\right) \equiv 1 \mod x^m.$$

Por tanto, operando módulo x^{2t},

$$\omega(x) = \sigma(x)\sum_{j=1}^{\nu} E_j X_j^b \frac{1}{1-xX_j} = \sigma(x)\sum_{j=1}^{\nu} E_j X_j^b \sum_{i=0}^{2t-1}(xX_j)^i$$
$$= \sigma(x)\sum_{i=0}^{2t-1}\sum_{j=1}^{\nu} E_j X_j^{i+b} x^i = \sigma(x)\sum_{i=0}^{2t-1} S_{i+b}x^i = \sigma(x)S(x).$$

□

Llegamos a esta ecuación, llamada *ecuación clave*.

$$\omega(x) \equiv \sigma(x)S(x) \mod x^{2t}. \tag{2.6}$$

El algoritmo de Sugiyama consistirá en resolver la ecuación (2.6).

La siguiente observación nos será útil después.

Lema 2.45. *Los polinomios $\omega(x)$ y $\sigma(x)$ son coprimos.*

Demostración. Las raíces de $\sigma(x)$ son precisamente X_j^{-1}, para $j \in \{1, \ldots, \nu\}$. Pero

$$\omega(X_j^{-1}) = E_j X_j \prod_{\substack{i=1 \\ i\neq j}}^{\nu}(1 - X_j^{-1}X_i) \neq 0,$$

lo que concluye la demostración del lema. □

Una vez tengamos las raíces de $\sigma(x)$, tendremos que invertirlas para determinar los números de coordenadas de error. Cuando tengamos los X_j, podremos resolver el sistema (2.5), obteniendo así los valores de E_j. Teniendo los valores de X_j y E_j, es fácil obtener los valores de k_j y e_{k_j}, los que nos proporciona directamente el vector de error $e(x)$. Por último, recordemos que el mensaje original $c(x)$, se obtiene como $c(x) = y(x) - e(x)$, lo que concluye la decodificación del mensaje.

En resumen, el algoritmo de Sugiyama consiste en:

Algoritmo 2.46 (Algoritmo de Sugiyama)**.**

(i) Calculamos el síndrome $S(x)$.

(ii) Sean $f(x) = x^{2t}$ y $s(x) = S(x)$. Fijamos $r_{-1}(x) = f(x), r_0(x) = s(x), b_{-1}(x) = 0$, y $b_0(x) = 1$.

(iii) *Encontrar* $h_i(x), r_i(x)$ *y* $b_i(x)$ *de manera inductiva para* $i = 1, 2, \ldots, I$, *hasta que* I *cumpla que* $\operatorname{gr}(r_{I-1}(x)) \geq t$ *y* $\operatorname{gr}(r_I(x)) < t$ *mediante las dos siguientes fórmulas:*

$$r_{i-2}(x) = r_{i-1}(x)h_i(x) + r_i(x), \text{ donde } \operatorname{gr}(r_i(x)) < \operatorname{gr}(r_{i-1}(x)),$$
$$b_i(x) = b_{i-2}(x) - h_i(x)b_{i-1}(x).$$

$\sigma(x)$ es un múltiplo escalar no nulo de $b_I(x)$, es decir, $\sigma(x) = \lambda b_I(x)$ para alguna constante λ no nula. Así, las raíces de $\sigma(x)$ son las mismas que las de $b_I(x)$.

(iv) *Calculamos las raíces de $b_I(x)$ y las invertimos para obtener los números de coordenadas de error X_j.*

(v) *Utilizamos los X_j para resolver el sistema 2.5 y así obtener las magnitudes de error E_j.*

(vi) *Calculamos el vector de errores $e(x)$ y se lo restamos al mensaje $y(x)$.*

Observemos que I está bien definido, ya que el grado de $r_i(x)$ decrece estrictamente. El siguiente lema nos será útil para demostrar posteriormente la eficacia del algoritmo de Sugiyama. La demostración puede consultarse en [15, p. 191].

Lema 2.47. *Usando la notación del algoritmo de Sugiyama 2.46, sea $a_{-1}(x) = 1$, $a_0(x) = 0$, y $a_i(x) = a_{i-2}(x) - h_i(x)a_{i-1}(x)$ para $i \geq 1$. Se verifican:*

(i) $a_i(x)f(x) + b_i(x)s(x) = r_i(x)$ *para* $i \geq -1$.

(ii) $b_i(x)r_{i-1}(x) - b_{i-1}(x)r_i(x) = (-1)^i f(x)$ *para* $i \geq 0$.

(iii) $a_i(x)b_{i-1}(x) - a_{i-1}(x)b_i(x) = (-1)^{i+1}$ *para* $i \geq 0$.

(iv) $\operatorname{gr}(b_i(x)) + \operatorname{gr}(r_{i-1}(x)) = \operatorname{gr}(f(x))$ *para* $i \geq 0$.

Ahora debemos verificar que el paso (ii) del algoritmo de Sugiyama funciona, es decir, debemos probar que $\sigma(x) = \lambda b_I(x), \lambda \neq 0$.

Teorema 2.48. *En las condiciones del algoritmo de Sugiyama 2.46 se cumple que existe un escalar no nulo λ tal que*

$$\sigma(x) = \lambda b_I(x).$$

Demostración. Utilizando el Lema 2.47 (i), tenemos que

$$a_I(x)x^{2t} + b_I(x)S(x) = r_I(x). \tag{2.7}$$

De la ecuación clave (2.6), también sabemos que para algún polinomio $a(x)$ se cumple

$$a(x)x^{2t} + \sigma(x)S(x) = \omega(x). \tag{2.8}$$

Multiplicamos (2.7) por $\sigma(x)$ y (2.8) por $b_I(x)$ para obtener

$$a_I(x)\sigma(x)x^{2t} + b_I(x)\sigma(x)S(x) = r_I(x)\sigma(x), \tag{2.9}$$
$$a_I(x)b_I(x)x^{2t} + \sigma(x)b_I(x)S(x) = \omega(x)b_I(x). \tag{2.10}$$

Operando módulo x^{2t}, esto implica que

$$r_I(x)\sigma(x) \equiv \omega(x)b_I(x) \mod x^{2t}. \tag{2.11}$$

Puesto que $\mathrm{gr}(\sigma(x)) \leq t$, por la elección de I,

$$\mathrm{gr}(r_I(x)\sigma(x)) = \mathrm{gr}(r_I(x)) + \mathrm{gr}(\sigma(x)) < t + t = 2t.$$

Por el Lema 2.47 (iv), la elección de I y el hecho de que $\mathrm{gr}(\omega(x)) < t$ se tiene que

$$\mathrm{gr}(\omega(x)b_I(x)) = \mathrm{gr}(\omega(x)) + \mathrm{gr}(b_I(x)) < t + \mathrm{gr}(b_I(x)) = t + (\mathrm{gr}(x^{2t}) - \mathrm{gr}(r_{I-1}(x))) \leq 3t - t = 2t.$$

Por tanto, (2.11) implica que $r_I(x)\sigma(x) = \omega(x)b_I(x)$. Esto, junto a (2.9) y (2.10), prueba que

$$a_I(x)\sigma(x) = a(x)b_I(x). \tag{2.12}$$

Sin embargo, el Lema 2.47 (iii) nos dice que $a_I(x)$ y $b_I(x)$ son coprimos y, por tanto, $a(x) = \lambda(x)a_I(x)$ por (2.12). Sustituyendo esto en (2.12) se tiene que

$$\sigma(x) = \lambda(x)b_I(x). \tag{2.13}$$

Sustituyendo de nuevo en (2.8)

$$\lambda(x)a_I(x)x^{2t} + \lambda(x)b_I(x)S(x) = \omega(x).$$

Así, (2.7) implica que

$$\omega(x) = \lambda(x)r_I(x) \tag{2.14}$$

Por el Lema 2.45, (2.13) y (2.14), $\lambda(x)$ debe ser una constante no nula, verificando el paso (ii) del algoritmo de Sugiyama. □

Algoritmo 1 Algoritmo de Sugiyama para códigos BCH

Entrada: Un polinomio recibido $y = \sum_{i=0}^{n-1} y_i x^i$ obtenido tras la transmisión de una palabra código $c \in \mathcal{C}$, donde $\mathcal{C}$ es un código BCH con capacidad de corrección de t errores.
Salida: Una palabra código c'.

Para $b \leq i \leq b + 2t - 1$ **hacer**
  $S_i \leftarrow y(\alpha^i)$
$S(x) \leftarrow \sum_{i=0}^{2t-1} S_{i+b} x^i$
$f(x) \leftarrow x^{2t}$
$s(x) \leftarrow S(x)$
$r_{-1}(x) \leftarrow f(x)$, $r_0(x) \leftarrow s(x)$
$b_{-1}(x) \leftarrow 0$, $b_0(x) \leftarrow 1$
$i \leftarrow 1$
Mientras $\text{gr}(r_{i-1}(x)) \geq t$ **o** $\text{gr}(r_i(x)) < t$ **hacer**
  $(\text{q}, \text{rem}) \leftarrow \text{quo-rem}(r_{i-2}, r_{i-1})$
  $r_{i-2}(x) \leftarrow r_{i-1}(x)$
  $r_{i-1}(x) \leftarrow \text{rem}$
  $b_i(x) \leftarrow b_{i-2}(x) - \text{q} \cdot b_{i-1}(x)$
  $b_{i-2}(x) \leftarrow b_{i-1}(x)$
  $b_{i-1}(x) \leftarrow b_i(x)$
  $i \leftarrow i + 1$
$I \leftarrow i$
Calcular las raíces de $b_I(x)$ e invertirlas para obtener las posiciones de error X_j
$k_j \leftarrow [\log_a(x_j)$ para cada x_j en $X_j]$
Utilizar los X_j para resolver el sistema de ecuaciones y obtener las magnitudes de error E_j
$e(x) \leftarrow \sum_j E_j x^{k_j}$
devolver $y(x) - e(x)$

A partir de este algoritmo, se ha realizado una implementación en SageMath [36] que puede consultarse en el enlace del apéndice A. Veamos un ejemplo donde se usa esta implementación.

Ejemplo 2.49. Sea $\mathcal{C}$ un $[15, 7]$-código BCH binario con distancia designada $\delta = 5$ y con conjunto característico $T = \{1, 2, 3, 4, 6, 8, 9, 12\}$.

Vamos a suponer que queremos enviar el mensaje $\mathbf{m} = 1001010$.

En primer lugar, lo codificamos mediante la matriz generadora.

```
sage: F = GF(2)
sage: C = BCHCode(F,15,5,offset = 1)
sage: C
> [15, 7] BCH Code over GF(2) with designed distance 5
sage: message = vector([1,0,0,1,0,1,0])
sage: G = C.generator_matrix()
sage: G
> [1 0 0 0 1 0 1 1 1 0 0 0 0 0 0]
  [0 1 0 0 0 1 0 1 1 1 0 0 0 0 0]
```

```
    [0 0 1 0 0 0 1 0 1 1 1 0 0 0 0]
    [0 0 0 1 0 0 0 1 0 1 1 1 0 0 0]
    [0 0 0 0 1 0 0 0 1 0 1 1 1 0 0]
    [0 0 0 0 0 1 0 0 0 1 0 1 1 1 0]
    [0 0 0 0 0 0 1 0 0 0 1 0 1 1 1]
sage: codeword = message * G
sage: codeword
> (1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0)
```

De esta forma, hemos obtenido la palabra código es $\mathbf{c} = 100111101010110$.

Una vez se ha codificado el mensaje original, vamos a suponer que se ha producido un error durante la transmisión de la palabra código, dicho error será $\mathbf{e} = 010000000000001$. Por tanto, la palabra que se ha recibido ha sido

$$\mathbf{y} = \mathbf{c} + \mathbf{e} = 100111101010110 + 010000000000001 = 110111101010111.$$

Vamos a utilizar el algoritmo de Sugiyama para decodificar la palabra que se ha recibido, y así obtener la palabra código original.

```
sage: error_vector = [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]
sage: received_message = coded_message + error_vector
sage: D = BCHSugiyamaDecoder(C)
sage: R = PolynomialRing(GF(2),'x')
sage: y = R(list(received_message))
sage: DEBUG = True
sage: D.decode_to_code(y)
> The code has a correction capability of 2  errors.
> List of syndromes:  [z4^3 + z4^2 + z4 + 1, z4^3 + z4, z4, z4^3]
> Xj : [z4^3 + z4^2 + 1, z4]
> The number of errors are : 2
> Position of the errors: [13, 1]
> Error magnitudes: (1, 1)
> Error vector: (1, 1)
> Codeword in polynomial form:  x^13 + x^12 + x^10 + x^8 + x^6 + x^5 + x
    ^4 + x^3 + 1
```

Observamos que el algoritmo de Sugiyama ha detectado dos errores, en las posiciones 1 y 13, ambas de magnitud 1. Por tanto, la palabra código corregida es $\mathbf{c} = 100111101010110$, tal y como buscábamos.

3. Códigos convolucionales

En este capítulo vamos a estudiar una clase de códigos muy distintos a los códigos de bloque vistos en el capítulo anterior, llamados *códigos convolucionales*. Recordemos que el propósito de los códigos es codificar la información de manera que se puedan *corregir los errores* que se han producido en la transmisión. La información que se suele enviar es demasiado larga como para ser procesada de una única vez, por tanto, esta información se divide en varios bloques, y en el caso de los códigos de bloque, se procesan de manera independiente.

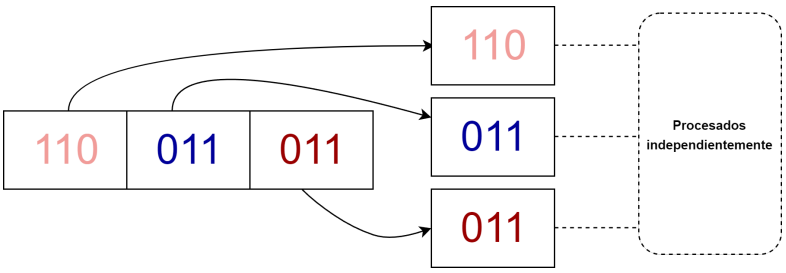

Figura 3.1.: Funcionamiento códigos de bloque.

Por otro lado, los códigos convolucionales utilizan un esquema de codificación que no depende únicamente del mensaje actual que se está transmitiendo, sino también de un cierto número de mensajes anteriores. Así, podríamos decir que, a diferencia de los códigos de bloque, un codificador de un código convolucional tiene "memoria".

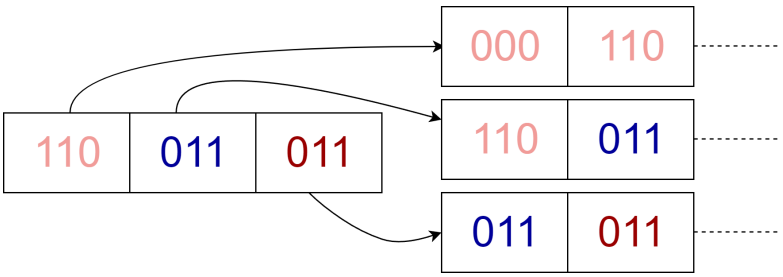

Figura 3.2.: Funcionamiento códigos convolucionales.

Los códigos convolucionales fueron presentados por primera vez por Peter Elias [5] en 1955 como una alternativa a los códigos de bloque. El primer algoritmo de decodificación para estos códigos fue la decodificación secuencial sugerida por Wozencraft [40] en 1957 y posteriormente desarrollada por Fano, quien en 1963 propuso un método de decodificación muy ingenioso [6]. Se desarrollaron muchos más algoritmos de decodificación para estos códigos, pero sin duda, el algoritmo de decodificación principal para estos códigos es el *algoritmo de Viterbi*, el cual es

un algoritmo de programación dinámica desarrollado por el ingeniero italiano A.J Viterbi [38] en 1967.

Si bien estos códigos se desarrollaron en los años 50, no fue hasta 1970 cuando Forney [7] desarrolló por primera vez una teoría algebraica sólida. No obstante, la teoría matemática existente para los códigos convolucionales no está tan desarrollada como lo está para los códigos de bloque. A lo largo de este capítulo presentaremos de varias formas los códigos convolucionales y estudiaremos cómo podemos codificar estos códigos mediante matrices generadoras.

Las principales fuentes para la redacción de este capítulo han sido [15], [7], [19] , [9] y [25].

3.1. Matrices generadoras y codificación de códigos convolucionales

Existen numerosas formas de definir los códigos convolucionales como veremos a lo largo del trabajo. Usualmente, los códigos convolucionales se han estudiado principalmente para el cuerpo $\mathbb{F}_2$, sin embargo, en los últimos años se han estudiado en ámbitos más generales, donde $\mathbb{F}$ es un cuerpo finito arbitrario.

Sea $\mathbb{F}$ un cuerpo finito y sea t[1] una indeterminada. Entonces, $\mathbb{F}[t]$ es el conjunto de todos los polinomios en la variable t con coeficientes en $\mathbb{F}$. Este conjunto es un *dominio de integridad* y, por tanto, podemos construir su *cuerpo de fracciones*, al cual denotaremos $\mathbb{F}(t)$.

Definición 3.1. Un (n,k)-*código convolucional* $\mathcal{C}$ es un subespacio k-dimensional de $\mathbb{F}(t)^n$.

Nótese que al ser $\mathbb{F}(t)$ un cuerpo infinito, $\mathcal{C}$ contendrá un número infinito de palabras código.

Definición 3.2. Sea $\mathcal{C}$ un (n,k)-código convolucional. El *ratio* R de $\mathcal{C}$ se define como $R = \frac{k}{n}$.

A diferencia de lo que ocurría con los códigos de bloque, no es usual considerar valores de n y k altos. Esto se debe a que la decodificación de códigos convolucionales es mucho más compleja.

Definición 3.3. Se dice que $G(t) \in \mathcal{M}_{k\times n}(\mathbb{F}(t))$ es una *matriz generadora* de $\mathcal{C}$ si sus filas forman una base de $\mathcal{C}$.

Si $G(t)$ es una matriz generadora, entonces también lo será cualquier matriz obtenida como el producto de $G(t)$ y un elemento no nulo de $\mathbb{F}(t)$. Por tanto, si multiplicamos $G(t)$ por el mínimo común múltiplo de los denominadores de todas las entradas de la matriz, obtendremos una matriz generadora de $\mathcal{C}$ cuyas entradas están formadas por polinomios. Por tanto, cualquier código convolucional tiene una matriz generadora cuyas entradas son polinómicas. Dichas matrices reciben el nombre de *matrices generadoras polinómicas* para $\mathcal{C}$.

[1] El símbolo t representa el operador delay en los códigos convolucionales. En otras referencias bibliográficas (ver por ejemplo [15]) se denota por D.

Ejemplo 3.4. Sea $G_1(t) = \begin{bmatrix} 1+t^2 & \frac{1}{1+t} \end{bmatrix}$ una matriz generadora de un $(2,1)$-código convolucional binario. Entonces la matriz $G_2(t) = (1+t)G_1(t) = \begin{bmatrix} 1+t+t^2+t^3 & 1 \end{bmatrix}$ es una matriz generadora polinómica de $\mathcal{C}$.

Utilizaremos estas matrices para codificar mensajes. Partiremos de la suposición de que estos mensajes son *finitos* [2] y los representaremos mediante polinomios $x(t) = \sum_{i=0}^{L-1} x(i)t^i \in \mathbb{F}[t]^k$.

Vamos a describir el procedimiento de codificación para un (n,k)-código convolucional $\mathcal{C}$ utilizando una matriz generadora polinómica $G(t)$. Para enfatizar la conexión entre $G(t)$ y el proceso de codificación, a menudo nos referiremos a $G(t)$ como *codificador*. Empezaremos suponiendo que $k = 1$, de esta manera, en cada momento $i = 0, 1, \ldots$ la entrada para el codificador $G(t)$ está formada por un único elemento $x(i) \in \mathbb{F}$.

Supongamos que tenemos un flujo de entrada de L elementos, cada uno de estos pertenecientes a $\mathbb{F}$. Estos elementos se representan como coeficientes de un polinomio $x(t)$, donde t es una indeterminada que representa el operador delay. El polinomio se define como $x(t) = \sum_{i=0}^{L-1} x(i)t^i$, donde $x(i)$ es el símbolo i del flujo de entrada. La codificación, al igual que con los códigos de bloque, se realiza mediante el producto $x(t)G(t) = c(t)$. La palabra código $c(t) = (c_1(t), \ldots, c_n(t))$ tiene n componentes, los cuales son polinomios [3] en t.

Ejemplo 3.5. Sea $\mathcal{C}$ un $(2,1)$-código convolucional binario y sea $G(t) = \begin{bmatrix} 1+t+t^2 & 1+t^2 \end{bmatrix}$ su matriz generadora. Vamos a codificar el mensaje $\mathbf{m} = 110101$, cuya longitud es $L = 6$. Este mensaje corresponde al polinomio $x(t) = \sum_{i=0}^{5} x(i)t^i = 1 + t + t^3 + t^5$. Lo codificamos utilizando la matriz generadora $G(t)$:

$$\begin{aligned} c(t) &= x(t)G(t) \\ &= (1+t+t^3+t^5) \begin{bmatrix} 1+t+t^2 & 1+t^2 \end{bmatrix} \\ &= \left(1+t^4+t^6+t^7,\ 1+t+t^2+t^7\right) \\ &= (c_1(t), c_2(t)). \end{aligned}$$

Si observamos detenidamente las multiplicaciones y las sumas realizadas para calcular $c_1(t)$ y $c_2(t)$, podemos ver que tanto la memoria como el operador delay t *juega un papel crucial*. Puesto que $c_1(t) = x(t)(1+t+t^2)$, en el momento i, observamos que $c_1(i) = x(i)(1+t+t^2) = x(i) + x(i-1) + x(i-2)$. De manera análoga, se tiene que $c_2(i) = x(i) + x(i-2)$, estas ecuaciones reciben el nombre de *ecuaciones del codificador*. Cada aparición de t^j retrasa la entrada en j unidades de tiempo. Diremos que la memoria del codificador es $M = 2$.

Definición 3.6. Sea $\mathcal{C}$ un (n,k)-código convolucional y sea $G(t) \in \mathcal{M}_{k\times n}(F[t])$ una matriz generadora polinómica de este código. Sean $g_{ij}(t)$ los elementos de $G(t)$. Se define la *memoria* M *de* $G(t)$ como el mayor grado de cualquier elemento de $G(t)$, es decir,

$$M = \max_{\substack{1\le i\le k \\ 1\le j\le n}} \left\{ \mathrm{gr}(g_{ij}(t)) \right\}.$$

[2] En teoría de códigos convolucionales es común trabajar con mensajes de longitud infinita, los cuales pueden representarse mediante series de Laurent formales en el cuerpo $\mathbb{F}((t))$.

[3] Si suponemos que el codificador $G(t)$ no es polinómico, entonces $c(t)$ puede ser un elemento del cuerpo de fracciones $\mathbb{F}(t)^n$.

Algo a tener en cuenta es que la memoria *depende exclusivamente del codificador* $G(t)$ y no del código $\mathcal{C}$. Recordemos que si multiplicamos la matriz $G(t)$ por cualquier elemento no nulo de $\mathbb{F}[t]$, seguirá siendo una matriz generadora para $\mathcal{C}$, sin embargo, su memoria puede cambiar.

Podemos generalizar el proceso de codificación anterior a un k arbitrario. En esta ocasión, la entrada en el instante $i = 0, 1, \dots$ está formada por los elementos $x_j(i) \in \mathbb{F}$, $j \in \{1, \dots, k\}$, los cuales forman el mensaje $\mathbf{x}(i) = (x_1(i), \dots, x_k(i))$. Al igual que antes, podemos reescribir cada x_j como un polinomio en la indeterminada t. De esta forma, la entrada resultante $\mathbf{x}(t)$ será un vector de k polinomios. La palabra código se calcula, de nuevo, como $\mathbf{x}(t)G(t) = \mathbf{c}(t) = (c_1(t), c_2(t), \cdots, c_n(t))$. Veamos un ejemplo.

Ejemplo 3.7. Sea $\mathcal{C}$ un $(4, 2)$-código convolucional binario y sea

$$G(t) = \begin{bmatrix} 1 & 1+t+t^2 & 1+t^2 & 1+t \\ 0 & 1+t & t & 1 \end{bmatrix}$$

una matriz generadora. Vamos a codificar el mensaje $\mathbf{m} = (11010, 10111)$, el cual corresponde al par polinómico $\mathbf{x}(t) = (1+t+t^3, 1+t^2+t^3+t^4)$. Lo codificamos mediante la matriz generadora $G(t)$:

$$\begin{aligned} \mathbf{x}(t)G(t) &= (1+t+t^3, 1+t^2+t^3+t^4) \begin{bmatrix} 1 & 1+t+t^2 & 1+t^2 & 1+t \\ 0 & 1+t & t & 1 \end{bmatrix} \\ &= (1+t+t^3, 1+t^2+t^4, 1+t^2+t^3+t^4, 0) = \mathbf{c}(t). \end{aligned}$$

Como el máximo grado de los elementos de $G(t)$ es 2, sabemos que su memoria M será también 2. Podemos también calcular las ecuaciones del codificador $G(t)$ de manera análoga al caso $k = 1$.

$$\begin{cases} c_1(i) = x_1(i), \\ c_2(i) = x_1(i) + x_1(i-1) + x_1(i-2) + x_2(i) + x_2(i-1), \\ c_3(i) = x_1(i) + x_1(i-2) + x_2(i-1), \\ c_4(i) = x_1(i) + x_1(i-1) + x_2(i). \end{cases}$$

Hemos visto como se puede codificar un mensaje mediante códigos convolucionales de forma matemática mediante la matriz generadora $G(t)$. Sin embargo, es interesante estudiar cómo podríamos construir un *codificador físico* [4] mediante registros de desplazamiento. Para codificar códigos convolucionales podemos usar registros de desplazamiento con retroalimentación lineal, un tipo particular de registros de desplazamiento en el que la entrada es un bit proveniente de aplicar una función de transformación lineal a un estado anterior. Los componentes principales de estos registros son los elementos de delay (también llamados flip-flops) y los sumadores binarios mostrados en la Figura 3.3.

[4] Nos centraremos en el caso en el que el cuerpo es $\mathbb{F}_2$ pues, al tratarse de 1's y 0's, su implementación es más sencilla.

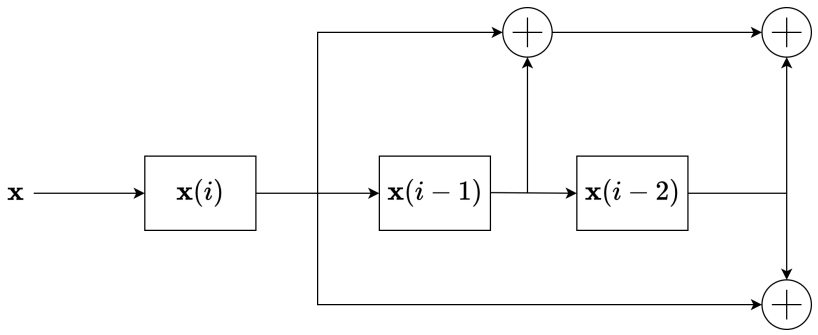

Figura 3.3.: Codificador físico para $G(t) = \begin{bmatrix} 1+t+t^2 & 1+t^2 \end{bmatrix}$.

3.2. Matrices generadoras canónicas

Hemos visto que un código convolucional $\mathcal{C}$ puede tener varias matrices generadoras, incluyendo matrices cuyas entradas son funciones racionales. Hasta ahora, nos hemos enfocado en las matrices generadoras polinómicas, pero en esta sección, definiremos una subclase de las matrices generadora polinómicas: las *matrices generadoras canónicas*.

Empecemos con unas definiciones importantes. Sea $G(t) = [g_{ij}(t)]$ una matriz polinómica de dimensión $k \times n$.

Definición 3.8. Se define *el grado m_i de la i-ésima fila de $G(t)$* como el máximo grado de cualquier elemento de la fila i, para $i \in \{1, \ldots, k\}$. Es decir,

$$m_i = \max_{1 \leq j \leq n} \text{gr}(g_{ij}(t)).$$

Definición 3.9. Se define el *grado externo* de $G(t)$, denotado extdeg($G(t)$), como la suma de los grados de las k filas de $G(t)$. Es decir,

$$\text{extdeg}(G(t)) = \sum_{i=1}^{k} m_i.$$

Ejemplo 3.10. Sea $G_1(t) = \begin{bmatrix} 1+t+t^2 & 1 \end{bmatrix}$ una matriz generadora de un $(2,1)$-código convolucional binario. Como solo tiene una fila, su grado externo y el grado de la fila 1 coinciden, y este es 2.

Sea $G_2(t) = \begin{bmatrix} 1 & 1+t+t^2 & 1+t^2 & 1+t \\ 0 & 1+t & t & 1 \end{bmatrix}$ una matriz generadora de un $(4,2)$-código convolucional binario. Entonces $m_1 = 2$, $m_2 = 1$ y, por tanto, extdeg($G(t)$) $= m_1 + m_2 = 3$.

Definición 3.11. Una *matriz generadora canónica* para un código convolucional $\mathcal{C}$ es una matriz generadora polinómica cuyo grado externo es el mínimo posible entre todas las posibles matrices generadoras polinómicas para $\mathcal{C}$.

Por definición, todos los códigos convolucionales cuentan con una matriz generadora canónica. Este grado externo mínimo recibe el nombre de *grado* del código $\mathcal{C}$.

Nótese que la matriz generadora canónica no es única (salvo el caso en el que el cuerpo sea $\mathbb{F}_2$), ya que podemos multiplicar esta matriz por cualquier elemento de $\mathbb{F}$ y seguiría conservando el grado externo.

Ejemplo 3.12. Vamos a demostrar que la matriz generadora $G_1(t)$ del ejemplo 3.10 es canónica para el $(2,1)$-código convolucional binario $\mathcal{C}$. En este caso, el grado externo de cualquier matriz generadora polinómica de $\mathcal{C}$ es el grado máximo de sus entradas (pues solo tiene una fila). Cualquier otra matriz generadora $G_1'(t)$ de $\mathcal{C}$ puede obtenerse de $G_1(t)$ multiplicando esta por un elemento de $\mathbb{F}(t)$, es decir, por una expresión del tipo $p(t)/q(t)$, donde $p(t), q(t) \in \mathbb{F}[t]$ (con $q(t) \neq 0$). Supondremos que $p(t)$ y $q(t)$ son polinomios coprimos (en caso contrario solo habría que simplificar). De esta forma, $G_1'(t) = \frac{p(t)}{q(t)} G_1(t)$.

Como las matrices generadoras canónicas son polinómicas, $q(t)$ debe dividir a los dos elementos de $G_1(t)$ y como el segundo elemento es 1, $q(t) = 1$. Por tanto, $G_1'(t) = p(t)G_1(t)$. De esta manera, para que no aumente el grado debe ocurrir que $\mathrm{gr}(p(t)) = 0$, o lo que es lo mismo, $p(t) \in \mathbb{F}$. Esto demuestra que $G_1(t)$ es una matriz generadora canónica.

El razonamiento del Ejemplo 3.12 nos dice que si $\mathcal{C}$ es un $(n,1)$-código convolucional, una matriz generadora polinómica $G(t)$ será canónica si y solo si el máximo común divisor de sus entradas es 1. Esto no es tan fácil cuando $k > 1$. A lo largo de esta sección buscaremos una condición *necesaria y suficiente* para que una matriz generadora polinómica sea una matriz generadora canónica de un (n,k)-código convolucional. Antes de demostrarlo vamos a introducir los siguientes conceptos.

Definición 3.13. Sea $G(t)$ una matriz polinómica de dimensión $k \times n$, con $k \leq n$. El *grado interno* de $G(t)$, denotado como $\mathrm{intdeg}(G(t))$, es el máximo grado de todos los menores [5] de $G(t)$ de dimensión $k \times k$. También se suele llamar *complejidad del código*. Un código de complejidad 0 es un código de bloque.

Definición 3.14. Se dice que una matriz generadora polinómica de un código convolucional $\mathcal{C}$ es *básica* si su grado interno es mínimo.

Definición 3.15. Una matriz polinómica $G(t)$ es *reducida* si entre todas las matrices de la forma $U(t)G(t)$, donde $U(t)$ es una matriz unimodular [6] de dimensión $k \times k$, $G(t)$ tiene el grado externo más pequeño.

En [27], McEliece dio seis formulaciones equivalentes para que una matriz sea básica y tres para que sea reducida. Presentaremos algunas de estas equivalencias y omitiremos las demostraciones. Estas pueden ser encontradas en [27, Apéndice A].

[5] Un menor de dimensión $k \times k$ de una matriz M de dimensión $k \times n$, $k \leq n$ es el determinante de una matriz formada por k columnas de M. La matriz M tiene $\binom{n}{k}$ menores.

[6] Una matriz U de dimensión $k \times k$ es *unimodular* si su determinante es 1.

Teorema 3.16. *Sea G una matriz generadora polinómica de un (n,k)-código convolucional. Equivalen las siguientes afirmaciones:*

(i) *La matriz G es básica.*

(ii) *El máximo común divisor de los menores $k \times k$ de $G(t)$ es 1.*

(iii) *Existe una matriz $K(t) \in \mathcal{M}_{n\times k}(\mathbb{F}[t])$ tal que $G(t)K(t) = I_k$.*

(iv) *Si $\mathbf{c}(t) \in \mathbb{F}[t]^n$ y $\mathbf{c}(t) = \mathbf{x}(t)G(t)$, entonces $\mathbf{x}(t) \in \mathbb{F}[t]^k$.*

Recordemos que la matriz $K(t)$ en (iii) es la inversa por la derecha de $G(t)$. La equivalencia (iii) del Teorema 3.16 establece que si $G(t)$ es un codificador básico, entonces siempre que la entrada sea polinomial, también lo será la salida. La prueba de que (iii) implica (iv) es simple. Si (iii) se verifica y $\mathbf{c}(t) = \mathbf{x}(t)G(t)$, entonces $\mathbf{x}(t) = \mathbf{x}(t)I_k = \mathbf{x}(t)G(t)K(t) = \mathbf{c}(t)K(t)$. Como $c(t)$ y $K(t)$ solo tienen entradas polinomiales, también las tendrá $\mathbf{x}(t)$.

Definición 3.17. El *grado* de un vector $\mathbf{v}(t) \in \mathbb{F}[t]^n$ es el mayor grado de cualquiera de sus componentes. [7]

Teorema 3.18. *Sea $G(t) \in \mathcal{M}_{k\times n}(\mathbb{F}[t])$ y sea $g_i(t)$ la i-ésima fila de $G(t)$. Equivalen las siguientes afirmaciones:*

(i) *La matriz G es reducida.*

(ii) $\text{intdeg}(G(t)) = \text{extdeg}(G(t))$.

(iii) *Para cualquier $\mathbf{x}(t) = (x_1(t), \ldots, x_k(t)) \in \mathbb{F}[t]^k$, se verifica*

$$\text{gr}(\mathbf{x}(t)G(t)) = \max_{i\leq i\leq k}\{\text{gr}(x_i(t)) + \text{gr}(g_i(t))\}.$$

Nuestro objetivo será probar que una matriz generadora polinómica de un (n,k)-código convolucional es canónica si y solo si es básica y reducida. Para ello, necesitaremos el siguiente lema.

Lema 3.19. *Sea $G(t) \in \mathcal{M}_{k\times n}(\mathbb{F}[t])$ una matriz generadora polinómica de un código convolucional $\mathcal{C}$, $k \leq n$. Sea $N(t)$ una matriz no singular de dimensión $k \times k$ con entradas en $\mathbb{F}[t]$. Se verifican las siguientes afirmaciones:*

(i) $\text{intdeg}(N(t)G(t)) = \text{intdeg}(G(t)) + \deg(\det(N(t)))$.

(ii) $\text{intdeg}(G(t)) \leq \text{intdeg}(N(t)G(t))$. *La igualdad se verifica si y solo si N es unimodular.*

(iii) $\text{intdeg}(G(t)) \leq \text{extdeg}(G(t))$.

[7] Nótese que esta definición es consistente con la defínción que hemos dado antes del grado de una matriz.

Demostración. Para probar (i) observamos que las submatrices de dimensión $k \times k$ de $N(t)G(t)$ son precisamente las submatrices $k \times k$ de $G(t)$ multiplicadas por $N(t)$ a la izquierda. Por tanto, los menores $k \times k$ de $N(t)G(t)$ son exactamente los menores $k \times k$ de $G(t)$ multiplicados cada uno por $\det(N(t))$. Lo que prueba (i).

La parte (ii) se deduce de (i).

Para (iii), supongamos que el grado de la i-ésima fila de $G(t)$ es m_i. Entonces, $\text{extdeg}(G(t))$ $m_1 + \cdots + m_k$. Cualquier menor $k \times k$ de $G(t)$ es una suma de productos de entradas de $G(t)$, con un factor de cada fila en cada producto. Por tanto, el mayor grado posible del producto es $m_1 + \cdots + m_k$, lo que prueba (iii). □

Con este resultado, podemos probar el teorema principal de esta sección, el cual caracteriza las matrices canónicas de un código convolucional.

Teorema 3.20. *Una matriz generadora polinómica de un (n, k)-código convolucional $\mathcal{C}$ es canónica si y solo si es básica y reducida.*

Demostración.

$\boxed{\Rightarrow}$

Sea G una matriz generadora canónica para $\mathcal{C}$. Sea d_0 el grado interno de las matrices generadoras básicas, que recordemos que es mínimo. Elegimos de entre todas las matrices generadoras básicas, una matriz $G_0(t)$ con grado externo mínimo.

Sea $U(t)$ una matriz unimodular de dimensión $k \times k$. Entonces utilizando el Lema 3.19,

$$U(t)G_0(t) = \text{intdeg}(G_0(t)) = d_0.$$

Por definición de $G_0(t)$, $\text{extdeg}(U(t)G_0(t)) \geq \text{extdeg}(G_0(t))$ (ya que $U(t)G_0(t)$ genera $\mathcal{C}$), lo que implica que $G_0(t)$ es reducida. Teniendo en cuenta que $G_0(t)$ tiene el grado interno más pequeño de entre todas las matrices generadoras polinómicas de $\mathcal{C}$, $\text{intdeg}(G_0(t)) \leq \text{intdeg}(G(t))$. Pero $\text{intdeg}(G(t)) \leq \text{extdeg}(G(t))$ por el Lema 3.19 y $\text{extdeg}(G(t)) \leq \text{extdeg}(G_0)$ por definición, ya que $G(t)$ es canónica. Por tanto,

$$\text{intdeg}(G_0(t)) \leq \text{intdeg}(G(t)) \leq \text{extdeg}(G(t)) \leq \text{extdeg}(G_0(t)). \tag{3.1}$$

Como $G_0(t)$ es reducida, $\text{intdeg}(G_0(t)) = \text{extdeg}(G_0(t))$ por el Teorema 3.18. Y, por tanto, se da la igualdad en (3.1).

De esta manera, $\text{intdeg}(G(t)) = \text{intdeg}(G_0(t)) = d_0$ probando que $G(t)$ tiene también grado interno mínimo entre todas las matrices generadoras polinómicas de $\mathcal{C}$, lo que implica que es básica. Como $\text{intdeg}(G(t)) = \text{extdeg}(G(t))$, obtenemos que $G(t)$ también es reducida.

$\boxed{\Leftarrow}$

Supongamos ahora que $G(t)$ es una matriz generadora polinómica de $\mathcal{C}$ reducida y básica. Sea $G_0(t)$ otra matriz generadora polinómica de $\mathcal{C}$. Por el Lema 3.19, $\text{extdeg}(G_0(t)) \geq \text{intdeg}(G_0(t))$. Como $G(t)$ es básica, $\text{intdeg}(G_0(t)) \geq \text{intdeg}(G(t))$. Y como también es reducida, $intdeg(G(t)) =$

extdeg$(G(t))$. Combinando estas desigualdades, llegamos a que extdeg$(G_0(t)) \geq$ extdeg$(G(t))$. Lo que implica que $G(t)$ es canónica, al ser $G_0(t)$ una matriz generadora polinómica arbitraria. □

3.3. Códigos convolucionales y módulos

Hasta ahora, hemos visto a los códigos convolucionales como subespacios k-dimensionales del cuerpo de fracciones $\mathbb{F}(t)^n$. Sin embargo, existen *varias definiciones de códigos convolucionales* y algunos autores han utilizado la teoría de módulos para describir los códigos convolucionales. En esta sección vamos a trabajar con algunos conceptos referentes a los códigos convolucionales usando módulos.

Definición 3.21. Un (n,k)*-código convolucional* es un sumando directo $\mathcal{C}$ del módulo $\mathbb{F}[t]^n$ de rango k.

En principio, parecen dos formas totalmente distintas de ver los códigos convolucionales. No obstante, ambas formas están íntimamente relacionadas, como nos muestra este resultado, mencionado en [11].

Proposición 3.22. *Sea $k \leq n$. La aplicación $D \to D \cap \mathbb{F}[t]^n$ establece una biyección entre el conjunto de los subespacios k-dimensionales de $\mathbb{F}(t)^n$ y el conjunto de todos los $\mathbb{F}[t]$-submódulos de $\mathbb{F}[t]^n$ con rango k que son sumandos directos de $\mathbb{F}[t]^n$.*

Esta proposición es un resultado de teoría de módulos y es un refinamiento del Teorema 3 de [7].

Definición 3.23. Una *matriz generadora o codificadora* es cualquier matriz $G(t) \in \mathcal{M}_{k\times n}(\mathbb{F}[t])$ de rango k que genere un código convolucional $\mathcal{C}$. Por tanto,

$$\mathcal{C} = Im(G(t)) = \{u(t)G(t) \mid u(t) \in \mathbb{F}[t]^k\}$$

y el vector $u(t)G(t) \in \mathbb{F}[t]^n$ es la palabra código asociada al mensaje $u(t) \in \mathbb{F}[t]^k$.

Observamos que las matrices generadoras de los códigos convolucionales vistos como sumandos directos del módulo $\mathbb{F}[t]^n$ son muy parecidas a las que definimos al principio de este capítulo. La única diferencia es que mientras que en las anteriores permitíamos coeficientes racionales del cuerpo de fracciones $\mathbb{F}(t)$, en estas matrices solo permitimos coeficientes polinómicos del DIP $\mathbb{F}[t]$. Sin embargo, recordemos que cualquier matriz generadora con coeficientes en $\mathbb{F}(t)$ se puede transformar en otra matriz que genere el mismo código con coeficientes en $\mathbb{F}[t]$ multiplicando por el mínimo común múltiplo de los denominadores de los coeficientes.

Definición 3.24. Una *matriz de paridad de un código convolucional* $\mathcal{C}$ es cualquier matriz $H(t) \in \mathcal{M}_{n\times(n-k)}(\mathbb{F}[t])$ que satisfaga

$$\mathcal{C} = \ker H(t) = \{v \in \mathbb{F}^n \mid v(t)H(t) = 0\}.$$

Definición 3.25. Sea V un submódulo de $\mathbb{F}[t]^n$. Entonces

(i) Se dice que V es *no catastrófico* si (1.2) se satisface para todo $v \in \mathbb{F}[t]^n$ y todo $\lambda \in \mathbb{F}[t] \backslash t\mathbb{F}[t]$.

(ii) Se dice que V es libre de delay si (1.2) se satisface para todo $v \in \mathbb{F}[t]^n$ y $\lambda = t$.

Un submódulo V no catastrófico y libre de delay es, por definición, *un código convolucional*, en base a la Proposición 1.49, que nos asegura que V es un sumando directo de $\mathbb{F}[t]^n$.

3.4. Distancias en códigos convolucionales

Al igual que los códigos de bloque, la distancia en los códigos convolucionales es una medida muy importante, pues nos permite evaluar la capacidad de un código para detectar y corregir los errores que se han producido en la transmisión. En lo que sigue, consideraremos a $\mathcal{C}$ un sumando directo del módulo libre $\mathbb{F}[t]^n$ de rango k.

Para los códigos de bloque utilizábamos la distancia de Hamming. Dados dos mensajes, $\mathbf{x}, \mathbf{y} \in \mathbb{F}^n$ su distancia de Hamming es el número de coordenadas en las que difieren $\mathbf{x}$ e $\mathbf{y}$. Vimos que también se podía definir utilizando el peso de Hamming, que consistía en el número de coordenadas no nulas de una palabra $\mathbf{x} \in \mathbb{F}^n$. Así, $d(x, y) = wt(x - y)$.

Podemos extender esta definición a los códigos convolucionales de la siguiente forma.

Definición 3.26. Sea $c(t) = \sum_{i=0}^{\mathrm{gr}(c(t))} c_i t^i \in \mathbb{F}[t]^n$. Definimos su peso de Hamming como

$$wt(c(t)) = \sum_{i=0}^{\mathrm{gr}(c(t))} wt(c_i).$$

Definición 3.27. Sean $c_1(t), c_2(t) \in \mathbb{F}[t]^n$. Definimos su distancia de Hamming como

$$d(c_1(t), c_2(t)) = wt(c_1(t) - c_2(t)).$$

Definición 3.28. La *distancia libre* de un código convolucional $\mathcal{C}$ es la mínima distancia entre dos palabras código distintas. Es decir,

$$d_{\mathrm{free}}(\mathcal{C}) = \min_{c_1(t), c_2(t) \in \mathcal{C}} \{d(c_1(t), c_2(t)) \mid c_1(t) \neq c_2(t)\}.$$

Durante la transmisión en un canal q-ario simétrico [8] se pueden producir varios errores. No obstante, mediante la distancia libre, podemos determinar el número de errores que podemos detectar y corregir.

[8] Donde q es el número de elementos del cuerpo finito $\mathbb{F}$.

Teorema 3.29. *Sea $\mathcal{C}$ un código convolucional con distancia libre d. Entonces, $\mathcal{C}$ puede detectar $d-1$ errores y puede corregir hasta*

$$t = \left\lfloor \frac{d-1}{2} \right\rfloor.$$

Demostración. Análoga a la realizada para los códigos de bloque en 2.16. □

Lema 3.30. *La distancia libre de un código convolucional $\mathcal{C}$ se puede calcular como*

$$d_{free} = \min_{c(t)\in\mathcal{C}} \{wt(c(t)) \mid c(t) \neq 0\}$$

Demostración. Demostración análoga a la realizada en 2.19. □

Además de la distancia libre, los códigos convolucionales también poseen otra noción de distancia.

Definición 3.31. Sea $x(t) \in \mathbb{F}[t]^n$ con $\mathrm{gr}(c(t)) = \gamma$, entonces $x(t) = x_0 + +x_1 t + \cdots + x_\gamma t^\gamma$ con $x_i \in \mathbb{F}^n$ para $t \in \{0, \ldots, \gamma\}$ y $c_i = \mathbf{0}$ para $i \geq \gamma + 1$. Definimos la truncación j-ésima de $c(t)$ como

$$c_{[0,j]}(t) = c_0 + c_1 t + \cdots + c_j t^j.$$

Definición 3.32. Para $j \in \mathbb{N}_0$, la distancia de la j-ésima de un código convolucional $\mathcal{C}$ se define como

$$d_j^c(\mathcal{C}) := \min_{c(t)\in\mathcal{C}} \{wt(c_{[0,j]}(t)) \mid c_0 \neq \mathbf{0}\}.$$

Al igual que con los códigos de bloque, existen límites superiores para las distancias de los códigos convolucionales.

Recordemos que para los códigos de bloque, disponíamos de la cota de Singleton 2.20. En el caso de los códigos convolucionales, también existe un límite superior para la distancia libre.

Sea $G(t)$ una matriz generadora de un código convolucional $\mathcal{C}$. Esta matriz podemos separarla en una suma finita de matrices de distintos grados, denotaremos por $G_\infty \in M_{k\times n}(\mathbb{F}[t])$ a la matriz de mayor grado de $G(t)$. En general, G_∞ no tiene rango k.

Sin embargo, cada módulo $\mathcal{C}$ de rango k tiene una matriz generadora $G'(t)$ tal que los grados de las filas decrecen y G'_∞ tiene rango k [33]. Además, en este caso, $\mathrm{intdeg}(G'(t)) = \sum_{i=1}^k v_i$, donde v_i es el grado de la i-ésima fila de $G(t)$. Si una matriz $G(t)$ cumple estas condiciones, diremos que está en *forma propia*. Los índices ordenados $v_1 \geq \cdots \geq v_k$ son invariantes del código convolucional y se denominan *índices de Kronecker* del código convolucional $\mathcal{C}$.

Los siguientes resultados se pueden encontrar en [33].

El siguiente lema será fundamental para encontrar una cota de d_{free}.

Lema 3.33. *Sea $\mathcal{C}$ un (n,k)-código convolucional, sea δ su grado interno y sea $G(t)$ una matriz generadora polinómica en forma propia. Sea ℓ el número de índices v_i entre los índices ordenados $v_1 \geq \cdots \geq v_k$ tales que $v_i = v_k$. Entonces la distancia libre satisface*

$$d_{free} \leq n(v_k + 1) - \ell + 1. \tag{3.2}$$

Utilizando el lema anterior, es fácil probar el siguiente teorema, el cual nos da una cota de la distancia libre de un código convolucional.

Teorema 3.34 (Cota de Singleton Generalizada)**.** *Sea $\mathcal{C}$ un (n,k)-código convolucional y sea δ su grado interno. Entonces,*

$$d_{free}(\mathcal{C}) \leq (n-k)\left(\left\lfloor \frac{\delta}{k} \right\rfloor + 1\right) + \delta + 1. \tag{3.3}$$

Demostración. La cota (3.2) es mayor conforme aumenta v_k y disminuye ℓ. El valor más alto posible de v_k es $\left\lfloor \frac{\delta}{k} \right\rfloor$ ya que recordemos que v_k es el grado más pequeño posible de una fila y como $G(t)$ está en forma propia, $\delta = \sum_{i=1}^{k} v_i$. Un valor de v_k mayor significaría que $\sum_{i=1}^{k} v_i \geq \delta$ lo que es imposible.

Partimos de que $v_{k-\ell+1} = v_{k-\ell+2} = \cdots = v_k = \left\lfloor \frac{\delta}{k} \right\rfloor$. Sabemos también que $\delta = \sum_{i=1}^{k} v_i = \sum_{i=1}^{k-\ell} v_i + \ell \left\lfloor \frac{\delta}{k} \right\rfloor$. Debemos minimizar los v_i, pero deben ser mayores que $\left\lfloor \frac{\delta}{k} \right\rfloor$. Por tanto, los valores óptimos que minimizan ℓ son $v_1 = v_2 = \cdots = v_{k-\ell} = \left\lfloor \frac{\delta}{k} \right\rfloor + 1$, $v_{k-\ell+1} = v_{k-\ell+2} = \cdots = v_k = \left\lfloor \frac{\delta}{k} \right\rfloor$. De esta forma,

$$\delta = \sum_{i=1}^{k} v_i = (k-\ell)(\left\lfloor \frac{\delta}{k} \right\rfloor + 1) + \ell \left\lfloor \frac{\delta}{k} \right\rfloor = k \left\lfloor \frac{\delta}{k} \right\rfloor + k - \ell \Rightarrow \ell = k - \delta + k \left\lfloor \frac{\delta}{k} \right\rfloor.$$

Sustituyendo este valor de ℓ en la cota (3.2) obtenemos la cota (3.3).

□

4. Códigos Convolucionales Cíclicos

En el capítulo anterior hemos estudiado los códigos convolucionales. Al igual que con los códigos de bloque, podríamos pensar en definir una nueva subclase de estos códigos que cumpla una propiedad de ciclicidad. No obstante, como ya se comentó, la teoría matemática de los códigos convolucionales está mucho menos desarrollada que la de los códigos de bloque y esta brecha es aún mayor para la teoría matemática de los códigos cíclicos convolucionales.

En 1976, Piret [29] estableció una base matemática para la teoría de códigos cíclicos convolucionales (CCC). Uno de sus descubrimientos principales fue que la ciclicidad clásica, tal y como la conocemos para los códigos de bloque, no funciona bien para los códigos convolucionales. Sin embargo, descubrió que trabajando en álgebras no conmutativas sí que existía una sorprendente analogía con los códigos cíclicos de bloque, ya que se podían definir los CCCs como ideales principales de un anillo no conmutativo.

Pocos años después, Roos [32] observó que estas álgebras no conmutativas que usó Piret eran un caso particular de las extensiones de Ore, definidas por Ore en 1933 [28]. De esta forma, Roos consideró álgebras más generales para la construcción de los CCCs. Sin embargo, estas ideas no fueron profundamente desarrolladas durante esos años, quizás porque los investigadores no estaban muy familiarizados con los polinomios de Ore y su manipulación. No fue hasta 2004 que Fluesing-Luerssen y Schmale [10] desarrollaron un estudio detallado de (como ellos lo llamaron) los σ-CCCs, en el que se trabaja con la noción de código convolucional $\mathcal{C}$ como un sumando directo del módulo libre $\mathbb{F}[t]^n$.

En este capítulo, vamos a definir el anillo de polinomios de Ore, los cuales nos permitirán construir unos nuevos tipos de CCCs, llamados códigos convolucionales cíclicos sesgados (SCCC), que a diferencia de los σ-CCCs, se considera a los códigos convolucionales $\mathcal{C}$ como un subespacio vectorial k-dimensional del cuerpo de fracciones $\mathbb{F}(t)^n$. Esto nos permitirá utilizar herramientas desarrolladas para polinomios de Ore que solo están disponibles cuando se trabaja con anillos de división.

Las principales referencias bibliográficas para la redacción de este capítulo han sido [8] y [18] para los anillos de polinomios sesgados, además de [9], [13] y [11] para los SCCC.

4.1. Anillos de polinomios sesgados

En esta sección se van a introducir los anillos de polinomios sesgados, los cuales son esenciales para definir los códigos convolucionales cíclicos sesgados. Estos polinomios, también conocidos como polinomios de Ore, fueron estudiados por Oystein Ore en [28]. La principal particularidad de estos era que sus coeficientes podían pertenecer a un anillo de división arbitrario, conmutativo o no conmutativo, con la restricción de que el grado del producto de dos polinomios sea igual a la suma de los grados de los factores

Definición 4.1. Sea $\mathcal{R}$ un anillo, σ un endomorfismo de anillos de $\mathcal{R}$ y δ una σ-derivación por la izquierda de $\mathcal{R}$, es decir, δ es aditivo y para $a, b \in \mathcal{R}$ se verifica que

$$\delta(ab) = \sigma(a)\delta(b) + \delta(a)b.$$

Entonces el anillo $\mathcal{R}[t;\sigma,\delta]$ de los polinomios en $\mathcal{R}[t]$ de la forma

$$a_0 + a_1 t + \cdots + a_n t^n,$$

donde $a_i \in \mathcal{R}$, con la igualdad y suma usuales y donde la multiplicación verifica que

$$ta = \sigma(a)t + \delta(a),\ \forall a \in \mathcal{R},$$

se conoce como el *anillo de polinomios de Ore* o *anillo de polinomios sesgados* y sus elementos reciben el nombre de *polinomios sesgados o de Ore.*

Se puede verificar que, efectivamente, $\mathcal{R}[t;\sigma,\delta]$ con las operaciones que hemos definido *es un anillo*. Para ello, debemos verificar que cumple todas las propiedades de anillo descritas en 1.1. Puesto que estamos utilizando la suma usual, solo debemos demostrar que se verifica la propiedad asociativa para la multiplicación. Se va a omitir la demostración, ya que los polinomios de Ore no son el objeto principal de este trabajo, no obstante, esta se puede encontrar en [18].

4.1.1. Anillo de polinomios sesgados sobre un cuerpo

Para el estudio de los códigos cíclicos convolucionales sesgados nos será de utilidad un caso particular de los anillos de polinomios sesgados, donde el anillo $\mathcal{R}$ es un cuerpo $\mathbb{K}$ arbitrario, $\sigma \in \mathrm{Aut}(\mathbb{K})$ y $\delta = 0$. Denotaremos a estos anillos por $\mathbb{K}[t;\sigma]$.

En el anillo $\mathbb{K}[t;\sigma]$ la multiplicación es de la forma

$$ta = \sigma(a)t,\ \forall a \in \mathbb{K}.$$

Si $\sigma = Id$, entonces $\mathbb{K}[t;\sigma] = \mathbb{K}[t]$, es decir, es un anillo de polinomios conmutativo sobre un cuerpo $\mathbb{K}$ en el sentido clásico. En general, los grupos aditivos de los anillos $\mathbb{K}[t]$ y $\mathbb{K}[t;\sigma]$ son idénticos, mientras que la multiplicación en $\mathbb{K}[t;\sigma]$ de dos polinomios sesgados $f, g \in \mathbb{K}[t;\sigma]$ viene dada por

$$\left(\sum_{i=0}^{n} a_i t^i\right)\left(\sum_{j=0}^{m} b_j t^j\right) = \sum_{i,j} a_i \sigma^i(b_j) t^{i+j}, \tag{4.1}$$

siendo $f(t) = a_0 + a_1 t + \cdots + a_n t^n$ y $g(t) = b_0 + b_1 t + \cdots + b_m t^m$.

Dado un polinomio sesgado $f \in \mathbb{K}[t;\sigma]$, $f \neq 0$, su grado se define de manera usual, es decir, como el mayor exponente de la variable t que aparece en la expresión de $f(t)$ como suma de monomios. Se denota como $\mathrm{gr}(f)$. Además, si $f = 0$, $\mathrm{gr}(f) := -\infty$. El elemento $a \in \mathbb{K}$ que

multiplica a la variable de mayor grado recibe el nombre de *coeficiente líder* y se denota como $\mathrm{cl}(f)$.

Esta definición no depende del lado en el que se encuentren los coeficientes, ya que σ es un automorfismo. De esta forma, en base a la identidad (4.1) obtenemos que dados dos polinomios sesgados $f, g \neq 0$

$$\begin{aligned} \mathrm{gr}(fg) &= \mathrm{gr}(f) + \mathrm{gr}(g), \\ \mathrm{gr}(f+g) &\leq \max\{\mathrm{gr}(f), \mathrm{gr}(g)\}. \end{aligned}$$

Esto implica que $\mathbb{K}[t;\sigma]$ es un dominio de integridad, ya que dados $f, g \in \mathbb{K}[t;\sigma]$, se tiene que $fg = 0 \Leftrightarrow \mathrm{gr}(fg) = \mathrm{gr}(f) + \mathrm{gr}(g) = -\infty \Leftrightarrow f = 0$ ó $g = 0$. Por tanto, $\mathbb{K}[t;\sigma]$ no tiene divisores de cero distintos del cero y, por tanto, es un *dominio de integridad no conmutativo*.

4.1.2. División

Al ser $\mathbb{K}[t;\sigma]$ un dominio de integridad no conmutativo, podemos definir algoritmos de división en $\mathbb{K}[t;\sigma]$ a la izquierda y a la derecha, de forma que para cada $f, g \in \mathbb{K}[t;\sigma], g \neq 0$, existen $q, r \in \mathbb{K}[t;\sigma]$ tales que al dividir por la izquierda obtenemos

$$f = qg + r,$$

y al dividir por la derecha,

$$f = gq + r.$$

Algoritmo 2 División Euclídea por la Izquierda en $\mathbb{K}[t;\sigma]$

Entrada: $f, g \in \mathbb{K}[t;\sigma]$ con $g \neq 0$.
Salida: $q, r \in \mathbb{K}[t;\sigma]$ tal que $f = qg + r$ y $\mathrm{gr}(r) < \mathrm{gr}(g)$.

1: $q \longleftarrow 0$
2: $r \longleftarrow f$
3: **Mientras** $\mathrm{gr}(g) \leq \mathrm{gr}(r)$ **hacer**
4: $\quad a \longleftarrow cl(r)\sigma^{\mathrm{gr}(r)-\mathrm{gr}(g)}(cl(g)^{-1})$
5: $\quad q \longleftarrow q + at^{\mathrm{gr}(r)-\mathrm{gr}(g)}$
6: $\quad r \longleftarrow r - at^{\mathrm{gr}(r)-\mathrm{gr}(g)}g$

Los polinomios r y q obtenidos como la salida del Algoritmo 4.1.2 reciben el nombre de *resto por la izquierda* y *cociente por la izquierda* respectivamente. Usaremos la notación $r = \mathrm{lrem}(f, g)$ y $q = \mathrm{lquo}(f, g)$. Se asumen las mismas convenciones y notaciones para el algoritmo de la división por la derecha.

De ahora en adelante, para simplificar la notación, llamaremos $R = \mathbb{K}[t, \sigma]$. Como estamos asumiendo que σ es un automorfismo y existen los algoritmos de división por la izquierda y por la derecha, R es tanto un DIP por la izquierda como por la derecha y, por tanto, es un DIP.

Algoritmo 3 División Euclídea por la Derecha en $\mathbb{K}[t;\sigma]$

Entrada: $f, g \in \mathbb{K}[t;\sigma]$ con $g \neq 0$.
Salida: $q, r \in \mathbb{K}[t;\sigma]$ tal que $f = gq + r$ y $\mathrm{gr}(r) < \mathrm{gr}(g)$.

```
q ⟵ 0
r ⟵ f
Mientras gr(g) ≤ gr(r) hacer
    a ⟵ σ^{-gr(g)}(cl(g)^{-1} cl(r))
    q ⟵ q + a t^{gr(r)-gr(g)}
    r ⟵ r - g a t^{gr(r)-gr(g)}
```

Sea I un ideal bilátero de R, entonces será de la forma $I = Rf = f^*R$ y para cualquier $g \in R$, existen $g', \tilde{g} \in R$ tales que $fg = g'f$ y $gf^* = f^*\tilde{g}$. Los elementos f tales que para cualquier $g \in R$ existen $g', \tilde{g} \in R$ tales que $fg = g'f$ y $gf = f\tilde{g}$ se llaman elementos *biláteros* y además $Rf = fR$ es un ideal.

El siguiente Teorema nos permite determinar los elementos biláteros y, por tanto, los ideales de R. Este teorema puede encontrarse en [18, p. 5].

Teorema 4.2. *Sea $R = \mathbb{K}[t;\sigma]$. Se verifican las siguientes afirmaciones.*

(i) Los elementos biláteros de R son los elementos $ac(t)t^n$ donde $a \in \mathbb{K}$, $n \in \mathbb{N}_0$ y $c(t) \in \mathcal{Z}(R)$, siendo $\mathcal{Z}(R)$ el centro [1] de R.

(ii) $\mathcal{Z}(R) = Inv(\sigma)$, donde $Inv(\sigma) = \{f \in \mathbb{K} : \sigma(d) = d\}$.

(iii) Supongamos que σ tiene orden finito n, es decir, $\sigma^n = Id$. Entonces, el centro $\mathcal{Z}(R)$ está formado por los polinomios de la forma

$$\gamma_0 + \gamma_1 t^r + \gamma_2 t^{2r} + \cdots \gamma_s t^{sr},$$

donde $\gamma_i \in \mathbb{K}$.

Dados $f, g \in R$, $Rf \subseteq Rg$, $Rg \neq 0$, significa que g es un divisor a la derecha de f y lo denotamos por $g \mid_d f$. Equivalentemente, podemos decir que f es un múltiplo a la izquierda de g.

Se tiene que $Ra = Rb \neq 0$ si y solo si $f \mid_d g$ y $g \mid_d f$. Así, $f = hg$ y $g = sf$ y, por tanto, $g = shg$. Por tanto, $sh = 1$, lo que implica que $hs = 1$ ya que R es un dominio de integridad. Por tanto, h y s son unidades. Se dice entonces que f y g son asociados por la izquierda, en el sentido de que $g = uf, u \in \mathcal{U}(R)$.

Tenemos que $Rf + Rg = Rh$. Entonces $h \mid_d f$ y $h \mid_d g$. Además, si $e \mid_d f$ y $e \mid_d g$, entonces $Re \supset Rf$ y $Re \supset Rg$, por lo que $Re \supset Rh$ y $e \mid_d h$. De esta manera, h es un *máximo común divisor por la derecha* de f y h y lo denotamos por $h = (f, g)_r$.

Dados $a, b \neq 0$, se dice que un anillo $\mathcal{A}$ satisface la *condición de Ore-Wedderburn por la izquierda* si $Ra \cap Rb \neq 0$. En [18, p.4] se demuestra que R cumple dicha condición. Tenemos así que $Rf \cap Rg = Rm$, por tanto $m = g'f = f'g \neq 0$. Además, si $f \mid_d n$ y $g \mid_d n$ entonces $Rm = Rf \cap Rb \supset Rn$, por tanto $m \mid_d n$. De esta forma, m es un *mínimo común múltiplo por la izquierda*

[1] El centro de un anillo $\mathcal{R}$ es el conjunto $\mathcal{Z}(\mathcal{R}) = \{c \in \mathcal{R} : cr = rc,\ \forall r \in \mathcal{R}\}$

de f y g y se denota por $[f,g]_\ell$. Tanto el máximo común divisor como el mínimo común múltiplo son únicos, salvo una multiplicación por la izquierda de una unidad.

Tanto $(f,g)_r$ como $[f,g]_\ell$ pueden ser calculados usando una versión apropiada del algoritmo extendido de Euclides por la Izquierda, el cual se detalla explícitamente a continuación.

Algoritmo 4 Algoritmo Extendido de Euclides por la Izquierda (AEEI) en $\mathbb{K}[t;\sigma]$

Entrada: $f,g \in \mathbb{K}[t;\sigma]$ con $f,g \neq 0$.
Salida: $n \in \mathbb{N}$, $u_i, v_i, q_i, f_i \in \mathbb{K}[t;\sigma]$ tales que $f_i = u_i f + v_i g$, $q_i = \text{lquo}(f_{i-1}, f_i)$, $f_n = (f,g)_r$, $u_n f = -v_n g = [f,g]_\ell$ para $1 \leq i \leq n+1$.

$u_0, v_1 \longleftarrow 1$
$u_1, v_0 \longleftarrow 1$
$f_0 \longleftarrow f$
$f_1 \longleftarrow g$
$i \longleftarrow 1$
Mientras $f_i \neq 0$ **hacer**
  $q_i \longleftarrow \text{lquo}(f_{i-1}, f_i)$
  $u_{i+1} \longleftarrow u_{i-1} - q_i u_i$
  $v_{i+1} \longleftarrow v_{i-1} - q_i v_i$
  $f_{i+1} \longleftarrow f_{i-1} - q_i f_i$
  $n \longleftarrow i$
  $i \longleftarrow i+1$

De igual manera, tanto $(f,g)_\ell$ como $[f,g]_r$ se pueden calcular utilizando una versión del Algoritmo Extendido de Euclides por la Derecha.

Algoritmo 5 Algoritmo Extendido de Euclides por la Derecha (AEED) en $\mathbb{K}[t;\sigma]$

Entrada: $f,g \in \mathbb{K}[t;\sigma]$ con $f,g \neq 0$.
Salida: $\{u_i, v_i, r_i\}_{i=0,\ldots,h,h+1}$ tales que $r_i = f u_i + g v_i$ para cualquier $i \in \{0,\ldots,h,h+1\}$, $r_h = (f,g)_\ell$ y $u_{h+1} f = [f,g]_r$.

$u_0, v_1 \longleftarrow 1$
$u_1, v_0 \longleftarrow 1$
$r_0 \longleftarrow f, r_1 \longleftarrow g$
$q \longleftarrow 0, \text{rem} \longleftarrow 0$
$i \longleftarrow 1$
Mientras $r_i \neq 0$ **hacer**
  $q, \text{rem} \longleftarrow \text{rquo-rem}(r_{i-1}, r_i)$
  $r_{i+1} \longleftarrow \text{rem}$
  $u_{i+1} \longleftarrow u_{i-1} - u_i q_i$
  $v_{i+1} \longleftarrow v_{i-1} - v_i q_i$
  $i \longleftarrow i+1$

Lema 4.3. *Sean $f,g \in \mathbb{K}[t;\sigma]$ y $\{u_i, v_i, r_i\}_{i=0,\ldots,h}$ los coeficientes obtenidos tras aplicar el AEED 5 a los polinomios sesgados f y g. Denotamos $R_0 = \begin{bmatrix} u_0 & u_1 \\ v_0 & v_1 \end{bmatrix}$, $Q_i = \begin{bmatrix} 0 & 1 \\ 1 & -q_i \end{bmatrix}$ y $R_i = R_0 Q_1 \cdots Q_i$ para $i \in \{0,\ldots,h\}$. Entonces, para cualquier $i \in \{0,\ldots,h\}$ se verifican las siguientes afirmaciones:*

(i) $[f\ g]R_i = [r_{i-1}\ r_i]$.

(ii) $R_i = \begin{bmatrix} u_i & u_{i+1} \\ v_i & v_{i+1} \end{bmatrix}$.

(iii) $fu_i + gv_i = r_i$.

(iv) R_i *tiene inversa por la izquierda y por la derecha.*

(v) $(u_i, v_i)_r = 1$.

(vi) $\mathrm{gr}(f) = \mathrm{gr}(r_{i-1}) + \mathrm{gr}(v_i) \quad (i \geq 1)$.

Demostración. Para probar *(i)* vamos a proceder por inducción. El caso $i = 0$ es trivial, ya que $[f\ g]R_0 = [f\ g]\begin{bmatrix} u_0 & u_1 \\ v_0 & v_1 \end{bmatrix} = [fu_0 + gv_0 \ \ fu_1 + gv_1] = [r_0\ r_1]$.

Supongámoslo cierto para n y probémoslo para $n+1$.

$$[f\ g]R_{n+1} = [f\ g]R_0Q_1 \cdots Q_nQ_{n+1} = [f\ g]R_nQ_{n+1} = [r_n\ r_{n+1}]Q_{n+1}.$$

Usando que $Q_{n+1} = \begin{bmatrix} 0 & 1 \\ 1 & -q_{n+1} \end{bmatrix}$, obtenemos que $[r_n\ r_{n+1}]Q_{n+1} = \begin{bmatrix} r_{n+1} & r_n - r_{n+1}q_{n+1} \end{bmatrix}$.

Y por definición, $r_{n+2} = r_n - r_{n+1}q_{n+1}$, obteniéndose lo que buscábamos.

Para *(ii)*, procedemos de forma similar utilizando inducción.

El caso $i = 0$ se tiene por definición. Supongámoslo cierto para n y probémoslo para $n+1$.

$$R_{n+1} = R_nQ_{n+1} = \begin{bmatrix} u_n & u_{n+1} \\ v_n & v_{n+1} \end{bmatrix}\begin{bmatrix} 0 & 1 \\ 1 & -q_{n+1} \end{bmatrix} = \begin{bmatrix} u_{n+1} & u_n - u_{n+1}q_{n+1} \\ v_{n+1} & v_n - v_{n+1}q_{n+1} \end{bmatrix} = \begin{bmatrix} u_{n+1} & u_{n+2} \\ v_{n+1} & v_{n+2} \end{bmatrix}.$$

La afirmación *(iii)* se sigue de *(i)* y *(ii)*.

Para *(iv)*, observemos que $T_i = \begin{bmatrix} q_i & 1 \\ 1 & 0 \end{bmatrix}$ es una inversa por la izquierda y por la derecha de Q_i. Por tanto, $S_i = T_i \cdots T_1R_0$ es una inversa por la izquierda y por la derecha de R_i.

Para *(v)*, si $R_i^{-1} = S_i = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, entonces

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} u_i & u_{i+1} \\ v_i & v_{i+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},$$

por tanto, existen $a, b \in \mathbb{K}[t;\sigma]$ verificando $au_i + bv_i = 1$ y entonces $(u_i, v_i)_r = 1$.

Por último, vamos a probar la afirmación *(vi)* por inducción.

Para $i = 1$, $r_0 = f$, $v_1 = 1$ y la igualdad es trivial.

Observemos que por el propio procedimiento del algoritmo $\mathrm{gr}(r_i) < \mathrm{gr}(r_{i-1})$ y $\mathrm{gr}(v_{i-1}) < \mathrm{gr}(v_i)$. Además, puesto que $r_{i+1} = r_{i-1} - r_iq_i$ y $v_{i+1} = v_{i-1} - v_iq_i$ para cualquier i, se tiene que $\mathrm{gr}(r_{i-1}) = \mathrm{gr}(r_i) + \mathrm{gr}(q_i)$ y $\mathrm{gr}(v_{i-1}) = \mathrm{gr}(v_i) + \mathrm{gr}(v_i)$ para cualquier i. Ahora, por

la hipótesis de inducción, $\mathrm{gr}(f) = \mathrm{gr}(r_{i-1}) + \mathrm{gr}(v_i) = \mathrm{gr}(r_i) + \mathrm{gr}(q_i) + \mathrm{gr}(v_{i+1}) - \mathrm{gr}(q_i) = \mathrm{gr}(r_i) + \mathrm{gr}(v_{i+1})$, como queríamos.

□

4.1.3. Norma

El siguiente concepto (ver [23]) nos será útil a la hora de calcular los restos obtenidos al dividir polinomios sesgados.

Definición 4.4. Sea $\gamma \in \mathbb{K}$, la *j-ésima* norma de γ es

$$N_j(\gamma) = \gamma\sigma(\gamma)\cdots\sigma^{j-1}(\gamma), \quad j > 0, \quad N_0(\gamma) = 1.$$

La noción de j-norma también admite una versión con índices negativos dada por

$$N_{-j}(\gamma) = \gamma\sigma^{-1}(\gamma)\cdots\sigma^{-j+1}(\gamma).$$

Proposición 4.5. *Sea $\gamma \in \mathbb{K}$ y $g(x) = \sum_{i=0}^{r} g_i x_i \in R$. Entonces se verifican las siguientes afirmaciones.*

(i) El resto por la izquierda de la división por la izquierda de g por $x - \gamma$ es $\sum_{i=0}^{r} g_i N_i(\gamma)$.

(ii) El resto por la derecha de la división por la derecha de g por $x - \gamma$ es $\sum_{i=0}^{r} \sigma^{-i}(g_i) N_{-i}(\gamma)$.

(iii) $N_j(\sigma^k(\gamma)) = \sigma^k(N_j(\gamma))$ para cualquier $j, k \in \mathbb{N}$.

Demostración. Para probar *(i)* procedemos como en [23, Lema 2.4].

Vamos a demostrar en primer lugar por inducción la siguiente proposición:

$$\text{Para todo } n \geq 0, \quad x^n - N_n(\gamma) \in R \cdot (x - \gamma). \tag{4.2}$$

El caso base es evidente, ya que $x^0 - N_0(\gamma) = 0 \in R \cdot (x - \gamma)$. Supongámoslo cierto para n.

Para $n + 1$ tenemos que

$$\begin{aligned} x^{n+1} - N_{n+1}(\gamma) &= x^{n+1} - \sigma(N_n(\gamma))(\gamma) \\ &= x^{n+1} - \sigma(N_n(\gamma))(\gamma) + \sigma(N_n(\gamma))x - \sigma(N_n(\gamma))x \\ &= x^{n+1} + \sigma(N_n(\gamma))(x - \gamma) - \sigma(N_n(\gamma))x \\ &= \sigma(N_n(\gamma))(x - \gamma) + xx^n - xN_n(\gamma) \\ &= \sigma(N_n(\gamma))(x - \gamma) + x(x^n - N_n(\gamma)) \in R \cdot (x - \gamma). \end{aligned}$$

Tenemos que si $x - \gamma$ divide por la izquierda a $g(x)$, entonces $g(x) = q(x)(x - \gamma) + r$, observemos que utilizando la proposición (4.2)

$$g(x) - \sum_{i=0}^{r} g_i N_i(\gamma) = \sum_{i=0}^{r} g_i x^i - \sum_{i=0}^{r} g_i N_i(\gamma) = \sum_{i=0}^{r} g_i (x^i - N_i(\gamma)) \in R \cdot (x - \gamma),$$

y, por tanto, $r = \sum_{i=0}^{r} g_i N_i(\gamma)$.

Para *(ii)* procedemos de manera análoga.

Probaremos por inducción que

$$\text{Para todo } n \geq 0, \quad x^n - N_{-n}(\gamma) \in (x - \gamma) \cdot R. \tag{4.3}$$

El caso base es de nuevo trivial, pues $x^0 - N_0(\gamma) = 0 \in (x - \gamma) \cdot R$.

Para $n + 1$ tenemos que

$$\begin{aligned} x^{n+1} - N_{-n-1}(\gamma) &= x^{n+1} - \gamma\sigma^{-1}(N_{-n}(\gamma)) \\ &= x^{n+1} - \gamma\sigma^{-1}(N_{-n}(\gamma)) + x\sigma^{-1}(N_{-n}(\gamma)) - x\sigma^{-1}(N_{-n}(\gamma)) \\ &= x^{n+1} + (x - \gamma)\sigma^{-1}(N_{-n}(\gamma)) - x\sigma^{-1}(N_{-n}(\gamma)) \\ &= (x - \gamma)\sigma^{-1}(N_{-n}(\gamma)) + x^n x - N_{-n}(\gamma)x \\ &= (x - \gamma)\sigma^{-1}(N_{-n}(\gamma)) + (x^n - N_{-n}(\gamma))x \in (x - \gamma) \cdot R. \end{aligned}$$

Donde se ha utilizado que para todo $a \in \mathbb{K}$, se tiene que $ax = x\sigma^{-1}(a)$ en R.

De esta forma, observamos que si $x - \gamma$ divide por la derecha a $g(x)$, entonces $g(x) = (x - \gamma)q(x) + r$. Utilizando la proposición (4.3)

$$\begin{aligned} g(x) - \sum_{i=0}^{r} \sigma^{-i}(g_i) N_{-i}(\gamma) &= \sum_{i=0}^{r} g_i x^i - \sum_{i=0}^{r} \sigma^{-i}(g_i) N_{-i}(\gamma) = \sum_{i=0}^{r} x^i \sigma^{-i}(g_i) - \sum_{i=0}^{r} \sigma^{-i}(g_i) N_{-i}(\gamma) \\ &= \sum_{i=0}^{r} (x^i - N_{-i}(\gamma)) \sigma^{-i}(g_i) \in (x - \gamma) \cdot R \end{aligned}$$

y, por tanto, $r = \sum_{i=0}^{r} \sigma^{-i}(g_i) N_{-i}(\gamma)$.

Para probar *(iii)* tenemos que

$$\begin{aligned} N_j(\sigma^k(\gamma)) &= \sigma^k(\gamma)\sigma^{k+1}(\gamma) \cdots \sigma^{k+j-1}(\gamma) \\ &= \sigma^k(\gamma\sigma(\gamma) \cdots \sigma^{j-1}(\gamma)) \\ &= \sigma^k(N_j(\gamma)). \end{aligned}$$

□

4.2. Ciclicidad en códigos convolucionales

En esta sección estudiaremos los códigos convolucionales cíclicos sesgados, los cuales se definen mediante un anillo de polinomios sesgados del cuerpo de fracciones $\mathbb{F}(t)$.

Comenzaremos con un resumen de algunos intentos previos para definir los códigos convolucionales cíclicos.

4.2.1. Primeros enfoques

Recordemos que un código de bloque $\mathcal{C} \subseteq \mathbb{F}^n$ se dice cíclico si es invariante bajo un desplazamiento cíclico, es decir, si

$$(v_0, v_1, \ldots, v_{n-1}) \in \mathcal{C} \Rightarrow (v_{n-1}, v_0, \ldots, v_{n-2}) \in \mathcal{C}, \tag{4.4}$$

o, equivalentemente, si $\mathcal{C}S \subseteq \mathcal{C}$, donde

$$S = \begin{bmatrix} 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \\ 1 & 0 & \cdots & 0 \end{bmatrix} \in \mathcal{M}_n(\mathbb{F}). \tag{4.5}$$

Para los códigos cíclicos de bloque utilizamos su representación polinomial, la cual está basada en el $\mathbb{F}$-isomorfismo

$$\mathfrak{p} : \mathbb{F}^n \longrightarrow \mathcal{R}_n, \quad v = (v_0, v_1, \ldots, v_{n-1}) \mapsto \mathfrak{p}(v) = \sum_{i=0}^{n-1} v_i x^i,$$

donde $\mathcal{R}_n = \mathbb{F}[x]/\langle x^n - 1 \rangle$.

De esta forma, la aplicación $\mathfrak{p}$ nos permite representar un desplazamiento cíclico multiplicando por x. Como consecuencia, un código cíclico de bloque $\mathcal{C}$ puede ser representado como un ideal $\mathfrak{p}(\mathcal{C})$ en $\mathcal{R}_n$, de esta forma, un código de bloque $\mathcal{C}$ es cíclico si y solo si

$$a \in \mathfrak{p}(\mathcal{C}) \Rightarrow xa \in \mathfrak{p}(\mathcal{C}).$$

Por tanto, podríamos pensar en definir los códigos convolucionales cíclicos de manera parecida. Sea

$$\mathfrak{p} : \mathbb{F}[z]^n \longrightarrow \mathcal{R}_n[z], \quad \sum_{i \geq 0} z^i v_i \mapsto \sum_{i \geq 0} z^i \mathfrak{p}(v_i),$$

un isomorfismo de $\mathbb{F}[z]$-módulos por la izquierda, con inversa $\mathfrak{v} := \mathfrak{p}^{-1}$.

Sería natural definir la ciclicidad de los códigos convolucionales al igual que con los códigos de bloque, es decir, exigiendo invarianza bajo desplazamientos cíclicos. Sin embargo, la siguiente proposición nos hace rechazar automáticamente esta idea.

Proposición 4.6. *[9, p. 194] Sea $\mathcal{C} \subseteq \mathbb{F}[z]^n$ un código que satisface (4.4) para todo $(v_0, \ldots, v_{n-1}) \in \mathcal{C}$. Entonces $\mathcal{C}$ es un código de bloque.*

Demostración. Por suposición, $\mathcal{C}S \subseteq \mathcal{C}$, donde S es una matriz como en (4.5). El polinomio minimal de S está dado por $x^n - 1$. Sea $x^n - 1 = \pi_1 \cdots \pi_r$ una factorización en factores irreducibles distintos en $\mathbb{F}$ (la cual existe según el Teorema 2.31). De esta forma, obtenemos la descomposición

$$\mathbb{F}[z]^n = \ker \pi_1(S) \oplus \ker \pi_2(S) \oplus \cdots \oplus \ker \pi_r(S)$$

de $\mathbb{F}[z]^n$ en $\mathbb{F}[z]$-submódulos los cuales son sumandos directos S-invariantes minimales. Pero recordemos que $\mathcal{C}$ también es un sumando directo, por tanto, obtenemos que

$$\mathcal{C} = \bigoplus_{i \in T} \ker \pi_i(S), \quad \text{donde } T = \{i : \ker \pi_i(S) \cap \mathcal{C} \neq \{0\}\}.$$

Puesto que $\mathbb{F}^n S = \mathbb{F}^n$, los $\mathbb{F}[z]$-submódulos $\ker \pi_i(S)$ están generados por $\ker \pi_i(S) \cap \mathbb{F}^n$ lo que nos lleva directamente a una matriz generadora constante, y, por tanto, a un codificador constante para $\mathcal{C}$. De esta manera, en base a la definición 3.13, $\mathcal{C}$ es un código de bloque ya que su complejidad es 0. □

El resultado de la Proposición 4.6 llevó a Piret [29] a dar una noción más general y compleja de ciclicidad en códigos convolucionales. En vez de una invarianza por desplazamientos de $\mathcal{C}$ bajo una matriz de desplazamiento S (4.5), Piret introdujo un tipo diferente de ciclicidad y llamó a un código convolucional $\mathcal{C}$ cíclico si existía un $m \in \mathbb{N}$, coprimo con la longitud n del código tal que

$$\sum_{i \geq 0} z^v v_i \in \mathcal{C} \Rightarrow \sum_{i \geq 0} z^i v_i S^{(m^i)} \in \mathcal{C}.$$

En forma polinomial, es decir, en el anillo de polinomios $\mathcal{R}_n[z]$, esto significa que

$$\sum_{i \geq 0} z^i \mathfrak{p}(v_i) \in \mathfrak{p}(\mathcal{C}) \Rightarrow \sum_{i \geq 0} z^i x^{(m^i)} \mathfrak{p}(v_i) \in \mathfrak{p}(\mathcal{C}).$$

El hecho de que m y n sean coprimos garantiza, no solo que el polinomio minimal de S^m es el mismo que el de S, es decir, $x^n - 1$, sino que también que la aplicación $x \longmapsto x^m$ induce un $\mathbb{F}$-automorfismo σ de $\mathcal{R}_n$.

La noción de ciclicidad de Piret fue generalizada por Roos [32] de una forma natural, considerando $\mathbb{F}$-automorfismos σ de $\mathcal{R}_n$ arbitrarios utilizando extensiones de Ore.

Definición 4.7. Un σ-código convolucional cíclico $\mathcal{C}$ (σ-CCC) es un sumando directo de $\mathbb{F}[z]^n$ tal que es un ideal por la izquierda de $\mathcal{R}_n[z;\sigma]$.

Observamos que $\mathcal{R}_n[z;\sigma]$ puede tener varios divisores de 0 y, por tanto, no es un anillo de división (ya que $\mathcal{R}_n$ no es un cuerpo). Tampoco es un DIP, por lo que no disponemos de operaciones como la división euclídea, el mínimo común múltiplo o el máximo común divisor. Como consecuencia, la decodificación utilizando estructuras cíclicas puede llegar a ser bastante complicada.

4.2.2. Códigos convolucionales cíclicos sesgados

Hemos visto que la noción clásica de ciclicidad para los códigos de bloque no sirve para los códigos convolucionales. También se ha visto que utilizar la extensión de Ore del anillo $\mathcal{R}_n[z]$ puede tener ciertos problemas, ya que no disponemos de las herramientas matemáticas desarrolladas para las extensiones de Ore cuando estas son un dominio de ideales principales.

Recordemos que los códigos convolucionales podían ser vistos como subespacios vectoriales k-dimensionales del cuerpo de fracciones $\mathbb{F}(t)^n$, siendo $\mathbb{F}$ un cuerpo finito arbitrario. Dado un $\mathbb{F}(t)$-automorfismo σ, consideramos su extensión de Ore $\mathbb{F}(t)[x;\sigma]$. Al ser $\mathbb{F}(t)$ un cuerpo, $\mathbb{F}(t)[x;\sigma]$ es un DIP y todos los algoritmos detallados en la sección 4.1.1 son válidos.

El grupo $\mathrm{Aut}(\mathbb{F}(t))$ es isomorfo a $\mathrm{PGL}(2,\mathbb{F})$, el grupo lineal proyectivo de las matrices 2×2 sobre $\mathbb{F}$ [37, p. 198], por tanto, es un grupo finito. En particular, σ tiene un orden finito n_σ bajo composición, es decir, $\sigma^{n_\sigma} = Id$, $\forall \sigma \in \mathrm{Aut}(\mathbb{F}(t)), n_\sigma \in \mathbb{N}$.

Nuestro objetivo es describir los códigos convolucionales cíclicos como un subespacio de $\mathbb{F}(t)^n$ asociado a un ideal por la izquierda del anillo cociente

$$\mathcal{R}_n = \mathbb{F}(t)[x;\sigma]/\langle x^n - 1\rangle.$$

En efecto, cada ideal por la izquierda $I \leq \mathcal{R}_n$ nos da un código convolucional $\mathcal{C} = \mathfrak{v}(I)$ de longitud n, donde $\mathfrak{v} : \mathcal{R}_n \mapsto \mathbb{F}(t)^n$ es la biyección asociada a la base $\mathcal{B} = \{1, x, \ldots, x^{n-1}\}$, $\mathfrak{v}^{-1} = \mathfrak{p}$. De esta forma, la longitud de los códigos convolucionales coincide con el orden de σ.

Definición 4.8. Un *código convolucional cíclico sesgado (SCCC)* $\mathcal{C}$ es un código convolucional cuya imagen $\mathfrak{p}(\mathcal{C})$ bajo

$$\mathbb{F}(t)^n \xrightarrow{\mathfrak{p}} \frac{\mathbb{F}(t)[x;\sigma]}{\langle x^n - 1\rangle}$$

es un ideal por la izquierda.

La estructura algebraica de los SCCC es muy diferente de la de los códigos cíclicos de bloque. Estos últimos son ideales del anillo cociente $\mathbb{F}[x]/\langle x^n - 1\rangle$ siempre y cuando n y la característica de $\mathbb{F}$ sean coprimos. El Teorema 2.32 nos dice que existe una correspondencia biyectiva entre los códigos cíclicos y los divisores mónicos del polinomio $x^n - 1$. Debemos estudiar la estructura de $\mathcal{R}_n$ para poder entender los SCCCs. El Teorema 4.2 nos dice que el centro de $\mathbb{F}(t)[x;\sigma]$ es el

anillo de polinomios conmutativo $\mathbb{F}(t)^\sigma[x^n]$, donde $\mathbb{F}(t)^\sigma$ denota el subcuerpo fijo, es decir, los elementos $a \in \mathbb{F}(t)$ tales que $\sigma(a) = a$.

Teorema 4.9. *El anillo $\mathcal{R}_n$ es isomorfo al anillo de matrices $\mathcal{M}_n(\mathbb{F}(t)^\sigma)$.*

Demostración. Puesto que el polinomio $x^n - 1$ es irreducible en $\mathbb{F}(t)^\sigma[x^n]$ (de hecho es lineal), tenemos que $\langle x^n - 1\rangle$ es un ideal bilátero primo de $R = \mathbb{F}(t)[x;\sigma]$. Por tanto, R_n es un anillo Artiniano simple por [17, Th. 13, p. 40]. Por el Teorema de Artin-Wedderburn (ver [3, Th 1.1]), obtenemos que $\mathcal{R}_n \cong \mathcal{M}_n(D)$, donde D es el álgebra de división formada por el conjunto de endomorfismos de un módulo por la izquierda simple sobre $\mathcal{R}_n$. Puesto que $x - 1$ es un factor irreducible por la derecha de $x^n - 1$ en R, tomamos $M = R/R(x-1)$. El endomorfismo de anillos de M es isomorfo a $\mathbb{F}(t)^\sigma$. En efecto, un isomorfismo de anillos explícito se puede definir de la siguiente forma.

Dado un endomorfismo $\mathcal{R}_n$-lineal $f : M \mapsto M$, escribimos $f(1 + R(x-1)) = \psi(f) + R(x-1)$, donde $\psi(f) \in \mathbb{F}(t)$ está determinado de manera única. Unos cálculos sencillos muestran que $\psi(f) \in \mathbb{F}(t)^\sigma$, y que la asignación $f \mapsto \psi$ es el isomorfismo buscado. Con este isomorfismo, se tiene que $\mathcal{R}_n \cong \mathcal{M}_n(\mathbb{F}(t)^\sigma)$. □

Como consecuencia del teorema anterior, para cada n existe un SCCC de dimensión $k \leq n$.

Estas estructuras cíclicas sobre los códigos convolucionales presentan muchas ventajas computaciones, ya que al trabajar en la extensión de Ore de un cuerpo disponemos de muchas herramientas aritméticas.

Veamos un ejemplo de SCCC.

Ejemplo 4.10. Sea $\sigma : \mathbb{F}_2(t) \to \mathbb{F}_2(t)$ un automorfismo de orden 2 definido por $\sigma(t) = 1/t$. Observamos que $x^2 + 1 = (x + 1/t^2)(x + t^2)$ en la extensión de Ore $\mathbb{F}_2(t)[x;\sigma]$. Por tanto, el ideal por la izquierda generado por $g = x + t^2$ es un $\mathbb{F}_2(t)$-subespacio propio de dimensión uno de $\mathcal{R} = \mathbb{F}_2(t)[x;\sigma]/\langle x^2 - 1\rangle$. De hecho, $\mathcal{C} = \mathfrak{v}(\mathcal{R}g)$ es un SCCC cuya matriz generadora es

$$\begin{bmatrix} t^2 & 1 \\ 1 & \frac{1}{t^2} \end{bmatrix}$$

,ya que $1(t^2 + x) = t^2 + x$ y $x(t^2 + x) = xt^2 + x^2 \equiv 1 + (1/t^2)x \mod x^2 - 1$.

4.2.3. Construcción de SCCCs

$\mathcal{R}_n$ es un DIP y, por tanto, todo ideal por la izquierda es principal. De esta manera, cualquier SCCC está generado por un divisor a la derecha de $x^n - 1$. Por tanto, al igual que con los códigos de bloque, debemos encontrar divisores por la derecha de este polinomio. Si estos factores pertenecen a $\mathbb{F}[x]$, los códigos construidos serán códigos cíclicos de bloque. Al ser $\mathcal{R}_n$ no conmutativo, existen descomposiciones de $x^n - 1$ con coeficientes no constantes en $\mathbb{F}(t)$ (ver 4.10). El problema es que hasta ahora no se ha descubierto un algoritmo de factorización eficiente para polinomios de Ore sobre $\mathbb{F}(t)$. Por tanto, vamos a proporcionar un procedimiento específico para $x^n - 1$. Empecemos con un método para encontrar divisores a la derecha lineales.

Proposición 4.11. *Sea $\beta \in \mathbb{F}(t)$, entonces $x - \beta$ divide por la derecha a $x^n - 1$ si y solo si $\beta = \sigma(c)c^{-1}$, para algún $c \in \mathbb{F}(t)$ no nulo.*

Demostración. Sea $R = \mathbb{F}(t)[x;\sigma]$. Si $x - \beta$ divide por la derecha a $x^n - 1$, entonces $R/R(x - \beta) \cong R/R(x - 1)$ como $\mathcal{R}$-módulos a la izquierda y $\beta = \sigma(c)c^{-1}$ para algún $c \in \mathbb{F}(t)$ no nulo [24, Proposición 2.4]. Recíprocamente, dado $c \in \mathbb{F}(t)$ no nulo y $\beta = \sigma(c)c^{-1}$, se puede deducir una versión de [4, Lema 4] para polinomios sesgados sobre $\mathbb{F}(t)$ de [18, Proposición 1.3.11] usando que $\sigma^i(c)c^{-1} = N_i(\sigma(c)c^{-1})$, lo que nos da que $x - \beta$ es un divisor a la derecha de $x^n - 1$. □

El Teorema 4.9 nos dice que $x^n - 1$ se puede descomponer como el mínimo común múltiplo por la izquierda de polinomios lineales (correspondientes a la representación del ideal cero de $\mathcal{R}_n$ como intersección de n módulos por la izquierda maximales). Para encontrar esta descomposición de $x^n - 1$ (y, por tanto, los divisores a la derecha no lineales), nuestra estrategia será calcular $\beta \in \mathbb{F}(t)$ de manera que

$$[x - \beta, x - \sigma(\beta), \ldots, x - \sigma^{n-1}(\beta)]_\ell = x^n - 1 \tag{4.6}$$

No todo $\beta \in \mathbb{F}(t)\backslash\mathbb{F}$ satisface (4.6). Usando [2, Proposición 1], (4.6) se satisface si y solo si el determinante de

$$\begin{bmatrix} 1 & \beta & \beta\sigma(\beta) & \cdots & \beta\sigma(\beta)\cdots\sigma^{n-2}(\beta) \\ 1 & \sigma(\beta) & \sigma(\beta)\sigma^2(\beta) & \cdots & \sigma(\beta)\sigma^2(\beta)\cdots\sigma^{n-1}(\beta) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \sigma^{n-1}(\beta) & \sigma^{n-1}(\beta)\beta & \cdots & \sigma^{n-1}(\beta)\beta\cdots\sigma^{n-3}(\beta) \end{bmatrix}$$

no es cero. Usando la Proposición 4.11, $\beta = \sigma(c)c^{-1}$, para un $c \in \mathbb{F}(t)$ no nulo. De esta forma, sustituyendo obtenemos

$$\begin{vmatrix} 1 & \sigma(c)c^{-1} & \sigma(c)c^{-1}\sigma(\sigma(c)c^{-1}) & \cdots & \sigma(c)c^{-1}\sigma(\sigma(c)c^{-1})\cdots\sigma^{n-2}(\sigma(c)c^{-1}) \\ 1 & \sigma(\sigma(c)c^{-1}) & \sigma(\sigma(c)c^{-1})\sigma^2(\sigma(c)c^{-1}) & \cdots & \sigma(\sigma(c)c^{-1})\sigma^2(\sigma(c)c^{-1})\cdots\sigma^{n-1}(\sigma(c)c^{-1}) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \sigma^{n-1}(\sigma(c)c^{-1}) & \sigma^{n-1}(\sigma(c)c^{-1})\sigma(c)c^{-1} & \cdots & \sigma^{n-1}(\sigma(c)c^{-1})\sigma(c)c^{-1}\cdots\sigma^{n-3}(\sigma(c)c^{-1}) \end{vmatrix} =$$

$$= \begin{vmatrix} 1 & \sigma(c)c^{-1} & \sigma^2(c)c^{-1} & \cdots & \sigma^{n-1}(c)c^{-1} \\ 1 & \sigma^2(c)\sigma(c)^{-1} & \sigma^3(c)\sigma(c)^{-1} & \cdots & c\sigma(c)^{-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & c\sigma^{n-1}(c)^{-1} & \sigma(c)\sigma^{n-1}(c)^{-1} & \cdots & \sigma^{n-2}(c)\sigma^{n-1}(c)^{-1} \end{vmatrix} \neq 0 \Leftrightarrow$$

$$\Leftrightarrow \begin{vmatrix} c & \sigma(c) & \sigma^2(c) & \cdots & \sigma^{n-1}(c) \\ \sigma(c) & \sigma^2(c) & \sigma^3(c) & \cdots & c \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \sigma^{n-1}(c) & c & \sigma(c) & \cdots & \sigma^{n-2}(c) \end{vmatrix} \neq 0$$

Equivalentemente, $\{c, \sigma(c), \ldots, \sigma^{n-1}(c)\}$ es una base normal de la extensión de cuerpos $\mathbb{F}(t)^\sigma \subseteq \mathbb{F}(t)$. El subcuerpo $\mathbb{F}(t)^\sigma = \mathbb{F}(s)$, donde $s \in \mathbb{F}(t)$ se puede calcular como se muestra en [12, Algoritmo 1]. En nuestro caso particular, el grupo elegido es el grupo cíclico $\{1, \sigma, \ldots, \sigma^{n-1}\}$, por tanto, el elemento $s \in \mathbb{F}(t)$ se puede obtener eligiendo cualquier coeficiente no constante del polinomio conmutativo $\prod_{i=0}^{n-1}(y - \sigma^i(t))$ en $\mathbb{F}(t)[y]$. La existencia de $c \in \mathbb{F}(t)$ para la construcción de dicha base está asegurada por el Teorema de la Base Normal. Realmente, c es un vector cíclico de σ visto como una aplicación $\mathbb{F}(t)^\sigma$-lineal y una forma de calcularlo es seguir el proceso descrito en [16, pp. 196-197 y pp. 293-294], aunque en la práctica, *una búsqueda aleatoria* será suficiente. Diferentes elecciones de β producen descomposiciones distintas de $x^n - 1$. De esta forma, para construir códigos SCCC de una dimensión dada, debemos calcular $\beta = \sigma(c)c^{-1}, c \in \mathbb{F}(t)$, de manera que el determinante de la matriz

$$\begin{bmatrix} c & \sigma(c) & \sigma^2(c) & \cdots & \sigma^{n-1}(c) \\ \sigma(c) & \sigma^2(c) & \sigma^3(c) & \cdots & c \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \sigma^{n-1}(c) & c & \sigma(c) & \cdots & \sigma^{n-2}(c) \end{bmatrix}$$

sea distinto de 0. De esta forma se verifica (4.6) y basta con tomar cualquier subconjunto $\{i_1, i_2, \ldots, i_k\} \subset \{0, 1, \ldots, n-1\}$ y definir

$$f = [x - \sigma^{i_1}(\beta), x - \sigma^{i_2}(\beta), \ldots, x - \sigma^{i_k}(\beta)]_\ell.$$

La imagen bajo $\mathfrak{v}$ del ideal por la izquierda generado por f es un SCCC de dimensión $n - k$ y longitud n.

4.2.4. Códigos convolucionales sesgados Reed-Solomon

Utilizando la construcción desarrollada en la sección anterior, vamos a construir códigos convolucionales cíclicos sesgados con una cierta distancia designada de Hamming. Debido a la analogía con los códigos de bloque Reed-Solomon definidos en 2.3.1, estos códigos reciben el nombre de *códigos convolucionales sesgados Reed-Solomon (RS)*.

El siguiente lema es un caso particular de [22, Corolario 4.13] y lo usaremos bastante a lo largo de esta sección.

Lema 4.12. *Sea L un cuerpo, σ un automorfismo de L de orden finito n, y $K = L^\sigma$ el subcuerpo fijo bajo σ. Sea $\{\alpha_0, \ldots, \alpha_{n-1}\}$ una K-base de L. Entonces, para todo $t \leq n$ y todo subconjunto $\{k_0 < k_1 < \cdots k_{t-1}\} \subseteq \{0, 1, \ldots, n-1\}$, el determinante de la matriz*

$$\begin{bmatrix} \alpha_{k_0} & \alpha_{k_1} & \cdots & \alpha_{k_{t-1}} \\ \sigma(\alpha_{k_0}) & \sigma(\alpha_{k_1}) & \cdots & \sigma(\alpha_{k_{t-1}}) \\ \vdots & \vdots & \ddots & \vdots \\ \sigma^{t-1}(\alpha_{k_0}) & \sigma^{t-1}(\alpha_{k_1}) & \cdots & \sigma^{t-1}(\alpha_{k_{t-1}}) \end{bmatrix}$$

es distinto de 0.

Demostración. Lo probaremos por inducción sobre t. El caso $t = 1$ es evidente. Suponemos que el lema se satisface para $t \geq 1$. Debemos de verificar que cualquier matriz

$$\Delta = \begin{bmatrix} \alpha_{k_0} & \alpha_{k_1} & \cdots & \alpha_{k_t} \\ \sigma(\alpha_{k_0}) & \sigma(\alpha_k) & \cdots & \sigma(\alpha_{k_t}) \\ \vdots & \vdots & \ddots & \vdots \\ \sigma^t(\alpha_{k_0}) & \sigma^t(\alpha_k) & \cdots & \sigma^t(\alpha_{k_t}) \end{bmatrix}$$

de dimensión $(t+1) \times (t+1)$ tiene un determinante $|\Delta| \neq 0$.

Supongamos que $|\Delta| = 0$, por hipótesis de inducción, las primeras t columnas de Δ son linealmente independientes y, por tanto existen $a_0, a_1, \ldots, a_{t-1} \in L$ tales que

$$(\alpha_{k_t}, \sigma(\alpha_{k_t}), \ldots, \sigma^t(\alpha_{k_t})) = \sum_{j=0}^{t-1} a_j (\alpha_{k_j}, \sigma(\alpha_{k_j}), \ldots, \sigma^t(\alpha_{k_j})).$$

Es decir, $a_0, \ldots, a_{t-1}$ satisfacen el sistema lineal

$$\begin{cases} \alpha_{k_t} = a_0\alpha_{k_0} + \cdots + a_{t-1}\alpha_{k_{t-1}} \\ \sigma(\alpha_{k_t}) = a_0\sigma(\alpha_{k_0}) + \cdots + a_{t-1}\sigma(\alpha_{k_{t-1}}) \\ \qquad\qquad \vdots \\ \sigma^t(\alpha_{k_t}) = a_0\sigma^t(\alpha_{k_0}) + \cdots + a_{t-1}\sigma^t(\alpha_{k_{t-1}}). \end{cases} \tag{4.7}$$

Para cualquier $j = 0, \ldots, t-1$, restamos en (4.7) la ecuación $j+1$ transformada por σ^{-1} de la ecuación j. Esto nos lleva al sistema lineal homogéneo

$$\begin{cases} 0 = \left(a_0 - \sigma^{-1}(a_0)\right) \alpha_{k_0} \\ \qquad + \cdots + \left(a_{t-1} - \sigma^{-1}(a_{t-1})\right) \alpha_{k_{t-1}} \\ 0 = \left(a_0 - \sigma^{-1}(a_0)\right) \sigma(\alpha_{k_0}) \\ \qquad + \cdots + \left(a_{t-1} - \sigma^{-1}(a_{t-1})\right) \sigma(\alpha_{k_{t-1}}) \\ \qquad\qquad \vdots \\ 0 = \left(a_0 - \sigma^{-1}(a_0)\right) \sigma^{t-1}(\alpha_{k_0}) \\ \qquad + \cdots + \left(a_{t-1} - \sigma^{-1}(a_{t-1})\right) \sigma^{t-1}(\alpha_{k_{t-1}}). \end{cases}$$

Por hipótesis de inducción, las primeras t columnas de Δ son linealmente independientes y, por tanto, $(a_j - \sigma^{-1}(a_j)) = 0$, para todo $j \in \{0, \ldots, t-1\}$. De esta forma, $a_0, \ldots, a_{t-1} \in K$, ya que $a_j = \sigma(a_j)$. En consecuencia, la primera ecuación de 4.7 nos da dependencia lineal sobre K de la K-base $\{\alpha_0, \ldots, \alpha_{n-1}\}$, lo que es una contradicción, ya que al ser una base es linealmente independiente. Por tanto $|\Delta| \neq 0$. □

Utilizando la Proposición 4.5(i), el resto de la división por la izquierda de un polinomio $g = \sum_{i=0}^{r} g_i x^i$ por $x - \gamma$ es $\sum_{i=0}^{r} g_i N_i(\gamma)$, donde N_i denota la i-ésima norma de $\gamma \in \mathbb{F}(t)$. Siempre que $x - \gamma$ divida por la derecha a g, diremos que γ es una *raíz por la derecha de* g. Recordemos que la noción de j-norma también admite una versión para índices negativos, dada por

$$N_{-j}(\gamma) = \gamma \sigma^{-1}(\gamma) \cdots \sigma^{-j+1}(\gamma).$$

Así, utilizando la Proposición 4.5(ii) el resto de la división por la derecha de un polinomio $g = \sum_{i=0}^{r} g_i x^i$ por $x - \gamma$ puede escribirse como $\sum_{i=0}^{r} \sigma^{-i}(g_i) N^{-i}(\gamma)$. Si $x - \gamma$ divide por la izquierda a g diremos que γ es una *raíz por la izquierda de* g.

El siguiente lema nos permitirá definir los códigos convolucionales sesgados RS.

Lema 4.13. *Sea α un elemento del cuerpo de fracciones $\mathbb{F}(t)$ tal que $\{\alpha, \sigma(\alpha), \ldots, \sigma^{n-1}(\alpha)\}$ es una base de $\mathbb{F}(t)$ como un $\mathbb{F}(t)^\sigma$-espacio vectorial. Sea $\beta = \alpha^{-1}\sigma(\alpha)$. Para cualquier subconjunto $T = \{t_1 < t_2 < \cdots < t_m\} \subseteq \{0, 1, \ldots, n-1\}$, los polinomios*

$$g^\ell = \left[x - \sigma^{t_1}(\beta), x - \sigma^{t_2}(\beta), \ldots, x - \sigma^{t_m}(\beta)\right]_\ell$$

y

$$g^r = \left[x - \sigma^{t_1}(\beta^{-1}), x - \sigma^{t_2}(\beta^{-1}), \ldots, x - \sigma^{t_m}(\beta^{-1})\right]_r$$

tienen grado m. Por tanto, si $x - \sigma^s(\beta) \mid_r g^\ell$ o $x - \sigma^s(\beta^{-1}) \mid_\ell g^r$, entonces $s \in T$.

Demostración. Supongamos que $\mathrm{gr}(g^\ell) < m$, por tanto, $g^\ell = \sum_{i=0}^{m-1} g_i x^i$. Como g es un múltiplo por la izquierda de $x - \sigma^{t_j}(\beta)$ para cualquier $j \in \{1, \ldots, m\}$, se tiene por la Proposición 4.5(i) que

$$\sum_{i=0}^{m-1} g_i N_i(\sigma^{t_j}(\beta)) = 0 \tag{4.8}$$

para cualquier $j \in \{1, \ldots, m\}$.

Esto es un sistema lineal homogéneo cuya matriz de coeficiente es la transpuesta de la matriz

$$M = \begin{bmatrix} N_0(\sigma^{t_1}(\beta)) & N_0(\sigma^{t_2}(\beta)) & \cdots & N_0(\sigma^{t_m}(\beta)) \\ N_1(\sigma^{t_1}(\beta)) & N_1(\sigma^{t_2}(\beta)) & \cdots & N_1(\sigma^{t_m}(\beta)) \\ N_2(\sigma^{t_1}(\beta)) & N_2(\sigma^{t_2}(\beta)) & \cdots & N_2(\sigma^{t_m}(\beta)) \\ \vdots & \vdots & \ddots & \vdots \\ N_{m-1}(\sigma^{t_1}(\beta)) & N_{m-1}(\sigma^{t_2}(\beta)) & \cdots & N_{m-1}(\sigma^{t_m}(\beta)) \end{bmatrix}.$$

Observemos que $N_i(\sigma^{t_j}(\beta)) = \sigma^{t_j}(N_i(\beta)) = \sigma^{t_j}(\alpha^{-1})\sigma^{t_j+i}(\alpha)$ para todo $j \in \{1,\dots,m\}$ y $i \in \{0,\dots,m-1\}$. Por tanto, $|M| = 0$ si y solo si el determinante de la matriz M',

$$\begin{bmatrix} \sigma^{t_1}(\alpha) & \sigma^{t_2}(\alpha) & \cdots & \sigma^{t_m}(\alpha) \\ \sigma^{t_1+1}(\alpha) & \sigma^{t_2+1}(\alpha) & \cdots & \sigma^{t_m+1}(\alpha) \\ \sigma^{t_1+2}(\alpha) & \sigma^{t_2+2}(\alpha) & \cdots & \sigma^{t_m+2}(\alpha) \\ \vdots & \vdots & \ddots & \vdots \\ \sigma^{t_1+m-1}(\alpha) & \sigma^{t_2+m-1}(\alpha) & \cdots & \sigma^{t_m+m-1}(\alpha) \end{bmatrix}$$

o, equivalentemente,

$$\begin{bmatrix} \sigma^{t_1}(\alpha) & \sigma^{t_2}(\alpha) & \cdots & \sigma^{t_m}(\alpha) \\ \sigma(\sigma^{t_1}(\alpha)) & \sigma(\sigma^{t_2}(\alpha)) & \cdots & \sigma(\sigma^{t_m}(\alpha)) \\ \sigma^2(\sigma^{t_1}(\alpha)) & \sigma^2(\sigma^{t_2}(\alpha)) & \cdots & \sigma^2(\sigma^{t_m}(\alpha)) \\ \vdots & \vdots & \ddots & \vdots \\ \sigma^{m-1}(\sigma^{t_1}(\alpha)) & \sigma^{m-1}(\sigma^{t_2}(\alpha)) & \cdots & \sigma^{m-1}(\sigma^{t_m}(\alpha)) \end{bmatrix}$$

es cero. No obstante, el Lema 4.12 nos asegura que $|M'| \neq 0$, por tanto, la única solución del sistema lineal (4.8) es $g_0 = g_1 = \cdots = g_{m-1} = 0$, lo que es una contradicción. Por tanto, $\mathrm{gr}(g^\ell) = m$. Para el otro polinomio, procedemos de forma similar, si suponemos que $\mathrm{gr}(g^r) < m$ y $g^r = \sum_{i=0}^{m-1} g_i x^i$ obtenemos el sistema lineal

$$\sum_{i=0}^{m-1} \sigma^{-i}(g_i) N_{-i}(\sigma^{t_j}(\beta^{-1})) = 0$$

para cualquier $j \in \{1,\dots,m\}$. □

Observamos ahora que $N_{-i}(\sigma^{t_j}(\beta^{-1})) = \sigma^{t_j}(\alpha^{-1})\sigma^{t_j-i+1}(\alpha)$ para $i \in \{0,\dots,m-1\}$ y $j \in \{1,\dots,m\}$. Entonces, por el Lema 4.12, el sistema tiene una única solución $\sigma^{-i}(g_i) = 0$ para $\{0,\dots,m-1\}$, así, $g_0 = \cdots = g_{m-1} = 0$. De nuevo, esto es una contradicción y $\mathrm{gr}(g^r) = m$.

Definición 4.14. Sean $\alpha, \beta \in \mathbb{F}(t)$ verificando las condiciones del Lema 4.13. Un código convolucional sesgado Reed-Solomon (RS) de distancia designada $\delta \leq n$ es un SCCC generado por

$$[x - \sigma^r(\beta), x - \sigma^{r+1}(\beta), \dots, x - \sigma^{r+\delta-2}(\beta)]_\ell,$$

para algún $r \geq 0$.

El siguiente teorema nos asegura que los códigos convolucionales sesgados RS son MDS con respecto a la distancia de Hamming.

Teorema 4.15. *Sea $\mathcal{C}$ un código convolucional sesgado RS de distancia designada δ. La distancia de Hamming de $\mathcal{C}$ es δ.*

Demostración. Sea

$$g = [x - \sigma^r(\beta), x - \sigma^{r+1}(\beta), \ldots, x - \sigma^{r+\delta-2}(\beta)]_\ell,$$

un generador de $\mathcal{C}$ como un ideal izquierdo de $\mathcal{R}_n$. Una matriz de paridad de $\mathcal{C}$ es

$$H = \begin{bmatrix} N_0(\sigma^r(\beta)) & N_0(\sigma^{r+1}(\beta)) & \cdots & N_0(\sigma^{r+\delta-2}(\beta)) \\ N_1(\sigma^r(\beta)) & N_1(\sigma^{r+1}(\beta)) & \cdots & N_1(\sigma^{r+\delta-2}(\beta)) \\ N_2(\sigma^r(\beta)) & N_2(\sigma^{r+1}(\beta)) & \cdots & N_2(\sigma^{r+\delta-2}(\beta)) \\ \vdots & \vdots & \ddots & \vdots \\ N_{n-1}(\sigma^r(\beta)) & N_{n-1}(\sigma^{r+1}(\beta)) & \cdots & N_{n-1}(\sigma^{r+\delta-2}(\beta)) \end{bmatrix},$$

ya que sus columnas nos dan las evaluaciones por la derecha de las raíces. Debemos probar que cualquier $(\delta-1)$-menor de H no es nulo. Vamos a proceder de manera parecida a como lo hicimos en el Lema 4.13. Observemos que $N_i(\sigma^k(\beta)) = \sigma^k(N_i(\beta)) = \sigma^k(\alpha^{-1})\sigma^{i+k}(\alpha)$ para cualesquiera enteros i y k. Por tanto, dada una submatriz M de dimensión $(\delta-1)\times(\delta-1)$,

$$\begin{bmatrix} N_{k_1}(\sigma^r(\beta)) & N_{k_1}(\sigma^{r+1}(\beta)) & \cdots & N_{k_1}(\sigma^{r+\delta-2}(\beta)) \\ N_{k_2}(\sigma^r(\beta)) & N_{k_2}(\sigma^{r+1}(\beta)) & \cdots & N_{k_2}(\sigma^{r+\delta-2}(\beta)) \\ N_{k_3}(\sigma^r(\beta)) & N_{k_3}(\sigma^{r+1}(\beta)) & \cdots & N_{k_3}(\sigma^{r+\delta-2}(\beta)) \\ \vdots & \vdots & \ddots & \vdots \\ N_{k_{\delta-1}}(\sigma^r(\beta)) & N_{k_{\delta-1}}(\sigma^{r+1}(\beta)) & \cdots & N_{k_{\delta-1}}(\sigma^{r+\delta-2}(\beta)) \end{bmatrix},$$

con $\{k_1 < k_2 < \cdots < k_{\delta-1}\} \subset \{0, 1, \ldots, n-1\}$, $|M| = 0$ si y solo si $|M'| = 0$, donde M' es la matriz

$$\begin{bmatrix} \sigma^{k_1+r}(\alpha) & \sigma^{k_1+r+1}(\alpha) & \cdots & \sigma^{k_1+r+\delta-2}(\alpha) \\ \sigma^{k_2+r}(\alpha) & \sigma^{k_2+r+1}(\alpha) & \cdots & \sigma^{k_2+r+\delta-2}(\alpha) \\ \sigma^{k_3+r}(\alpha) & \sigma^{k_3+r+1}(\alpha) & \cdots & \sigma^{k_3+r+\delta-2}(\alpha) \\ \vdots & \vdots & \ddots & \vdots \\ \sigma^{k_{\delta-1}+r}(\alpha) & \sigma^{k_{\delta-1}+r+1}(\alpha) & \cdots & \sigma^{k_{\delta-1}+r+\delta-2}(\alpha) \end{bmatrix} =$$

$$= \begin{bmatrix} \sigma^{k_1+r}(\alpha) & \sigma(\sigma^{k_1+r}(\alpha)) & \cdots & \sigma^{\delta-2}(\sigma^{k_1+r}(\alpha)) \\ \sigma^{k_2+r}(\alpha) & \sigma(\sigma^{k_2+r}(\alpha)) & \cdots & \sigma^{\delta-2}(\sigma^{k_2+r}(\alpha)) \\ \sigma^{k_3+r}(\alpha) & \sigma(\sigma^{k_3+r}(\alpha)) & \cdots & \sigma^{\delta-2}(\sigma^{k_3+r}(\alpha)) \\ \vdots & \vdots & \ddots & \vdots \\ \sigma^{k_{\delta-1}+r}(\alpha) & \sigma(\sigma^{k_{\delta-1}+r}(\alpha)) & \cdots & \sigma^{\delta-2}(\sigma^{k_{\delta-1}+r}(\alpha)) \end{bmatrix}$$

Como $\{\alpha, \sigma(\alpha), \ldots, \sigma^{n-1}(\alpha)\}$ es una base de la extensión de cuerpos $\mathbb{F}(t)^\sigma \subset \mathbb{F}(t)$, por el Lema 4.12, $|M'| \neq 0$. □

Cada función racional $f(t) \in \mathbb{F}(t)$ puede ser vista como una serie de potencias formal de Laurent. Así, cada palabra código $v(t) \in \mathbb{F}(t)^n$ se puede expresar mediante una serie de Laurent con coeficientes en $\mathbb{F}^n$. De esta forma,

$$v(t) = \sum_{i=m}^{+\infty} v_i t^i, v_i \in \mathbb{F}^n, m \in \mathbb{Z}.$$

Por tanto, podemos definir el *peso* de $v(t) \in \mathcal{C}$ como la suma de los pesos de Hamming de todos sus coeficientes. Dado un código convolucional, es decir, un $\mathbb{F}(t)$-subespacio vectorial $\mathcal{C}$ de $\mathbb{F}(t)^n$, su distancia libre se define como el mínimo de los pesos de todas las palabras código $v(t) \in \mathcal{C}$. Es evidente que $\mathrm{wt}(v(t))$ será finito si y solo si $v(t)$ es un polinomio de Laurent.

Esta discusión es un poco distinta a la que hicimos en la sección 3.4. Esto se debe a que usamos la definición de código convolucional como sumando directo del módulo libre $\mathbb{F}[t]^n$. Sin embargo, dado $k \in \{0, \ldots, n\}$ la Proposición 3.22 nos dice que la aplicación $\mathcal{C} \mapsto \mathcal{C} \cap \mathbb{F}[t]^n$ establece una correspondencia biyectiva entre el conjunto de todos los $\mathbb{F}(t)$-subespacios vectoriales k-dimensionales de $\mathbb{F}(t)^n$ y el conjunto de todos los $\mathbb{F}[t]$-submódulos de rango k de $\mathbb{F}[t]^n$ que son sumandos directos. Por tanto,

$$d_{\text{free}}(\mathcal{C}) = \min\{\mathrm{wt}(v(t)) : 0 \neq v(t) \in \mathcal{C} \cap \mathbb{F}[t]^n\}.$$

De esta forma, la distancia libre de $\mathcal{C}$ es igual a la distancia libre de $\mathcal{C} \cap \mathbb{F}[t]^n$ tal y como la definimos en 3.28.

El siguiente resultado nos da una cota para la distancia libre de un código convolucional sesgado RS.

Corolario 4.16. *Sea $\mathcal{C}$ un código convolucional sesgado RS de longitud n y dimensión k. Entonces*

$$n - k + 1 \leq d_{free}(\mathcal{C}) \leq (n - k)(\lfloor m/k \rfloor + 1) + m + 1,$$

donde m es el grado interno de $\mathcal{C} \cap \mathbb{F}[t]^n$.

Demostración. Un código convolucional sesgado RS de longitud n y dimensión k tiene distancia designada de Hamming $\delta = n - k + 1$. Por el Teorema 4.15, la distancia de Hamming de $\mathcal{C}$ es $n - k + 1$. La distancia libre de $\mathcal{C}$ es siempre más grande que su distancia de Hamming y menor que la cota generalizada de Singleton por el Teorema 3.34.

□

5. Algoritmo de Sugiyama para códigos convolucionales sesgados RS

Una de las principales razones de por qué los códigos cíclicos de bloque son útiles es porque poseen una teoría matemática que nos permite diseñar algoritmos de decodificación muy eficientes. Un ejemplo es el algoritmo de Sugiyama estudiado en la sección 2.4 para la decodificación de códigos de bloque BCH.

Al tratar con códigos convolucionales, el algoritmo de Viterbi es, con diferencia, el algoritmo de decodificación más utilizado. Este algoritmo nos permite decodificar códigos convolucionales en canales binarios simétricos o con ruido gaussiano blanco aditivo. Hasta ahora, hemos visto que dotar a los códigos convolucionales de una estructura cíclica requiere de una multiplicación no conmutativa. Además, muchas de las propuestas de códigos cíclicos convolucionales no han conseguido aprovechar sus estructuras algebraicas para encontrar algoritmos de decodificación eficientes y prácticos, y se han limitado a proponer alternativas al algoritmo de Viterbi. Por ejemplo, los σ-CCC códigos propuestos por Roos [32] no disponen del algoritmo de Euclides para la división.

En este capítulo vamos a estudiar un algoritmo de decodificación para los códigos convolucionales sesgados Reed-Solomon y es una exposición de los resultados descritos en [11].

A lo largo de este capítulo, $\mathcal{C}$ será un código convolucional sesgado RS de distancia designada δ, generado como un ideal por la izquierda de $\mathcal{R}_n$ por el polinomio

$$g = \left[x - \sigma^r(\beta), x - \sigma^{r+1}(\beta), \ldots, x - \sigma^{r+\delta-2}(\beta)\right]_\ell$$

para algún $r \geq 0$, donde β se elige como en la Definición 4.14. Por el Teorema 4.15, la distancia de Hamming de $\mathcal{C}$ es exactamente δ.

Sea $\tau = \left\lfloor \frac{\delta-1}{2} \right\rfloor$ el número máximo de errores que el código puede corregir. Por simplicidad, supondremos que $r = 0$. Esto no es una restricción, ya que podemos escribir $\beta' = \sigma^r(\beta)$. De esta manera, $\beta' = (\alpha')^{-1}\sigma(\alpha')$, donde $\alpha' = \sigma^r(\alpha)$, y α' nos da también una base normal. Por tanto,

$$g = \left[x - \beta', x - \sigma(\beta'), \ldots, x - \sigma^{\delta-2}(\beta')\right]_\ell.$$

Sea $c \in \mathcal{C}$ una palabra código que se transmite por un canal con ruido. Entonces, se recibirá un polinomio $y = c + e$, donde

$$e = e_1 x^{k_1} + e_2 x^{k_2} + \cdots + e_\nu x^{k_\nu}, \quad \nu \leq \tau.$$

Definimos el *polinomio localizador de errores* como

$$\lambda = \left[1 - \sigma^{k_1}(\beta)x, 1 - \sigma^{k_2}(\beta)x, \ldots, 1 - \sigma^{k_\nu}(\beta)x\right]_r.$$

Veamos en primer lugar que λ determina las posiciones que no tienen error. Para ello, veamos primero un lema que nos será de utilidad.

Lema 5.1. *Para cualquier subconjunto $\{t_1, \ldots, t_m\} \subseteq \{0, 1, \ldots, n-1\}$, se cumple la igualdad*

$$\left[1 - \sigma^{t_1}(\beta)x, \ldots, 1 - \sigma^{t_m}(x)\right]_r = \left[x - \sigma^{t_1 - 1}(\beta^{-1}), \ldots, x - \sigma^{t_m - 1}(\beta^{-1})\right]_r.$$

Demostración. Para cualquier $a \in \mathbb{F}(t)$, se cumple que

$$1 - ax = (x - \sigma^{-1}(a^{-1}))(-\sigma^{-1}(a)) \Leftrightarrow x - \sigma^{-1}(a^{-1}) = (1 - ax)(-\sigma^{-1}(a^{-1})).$$

Por tanto, los polinomios del enunciado se dividen por la izquierda entre sí. De esta forma, el mínimo común múltiplo por la derecha coincide.

□

Proposición 5.2. *El polinomio $1 - \sigma^d(\beta)x$ divide por la izquierda al polinomio localizador de errores λ si y solo si $x - \sigma^{d-1}(\beta^{-1})$ divide por la izquierda a λ si y solo si $d \in \{k_1, \ldots, k_\nu\}$.*

Demostración. Por el Lema 5.1, $1 - \sigma^d(\beta)x$ divide por la izquierda a λ si y solo si $x - \sigma^{d-1}(\beta^{-1})$ divide por la izquierda a λ. Ahora, por el Lema 4.13, $x - \sigma^{d-1}(\beta^{-1})$ divide por la izquierda a λ si y solo si $d \in \{k_1, \ldots, k_\nu\}$. □

Por tanto, una vez se conoce el polinomio localizador de errores λ, las posiciones donde se encuentran errores pueden ser localizadas siguiendo la siguiente regla: $d \in \{0, 1, \ldots, n-1\}$ es una posición de error si y solo si $\sigma^{d-1}(\beta^{-1})$ es una raíz por la izquierda de λ. Nótese que λ se puede sustituir por cualquier polinomio en $\mathbb{F}(t)[x;\sigma]$ obtenido de la multiplicación por la derecha de λ por un elemento $a \in \mathbb{F}(t)\backslash\{0\}$.

Dado $j \in \{1, \ldots, \nu\}$, $\lambda = (1 - \sigma^{k_j}(\beta)x)p_j$ para algún polinomio $p_j \in \mathbb{F}(t)[x;\sigma]$ con grado $\nu - 1$, definimos el *polinomio evaluador de errores* como $\omega = \sum_{j=1}^{\nu} e_j \sigma^{k_j}(\alpha) p_j$. Una vez conocemos el polinomio localizador de errores y el polinomio evaluador de errores, podemos calcular los valores $e_1, e_2, \ldots, e_\nu$ resolviendo un sistema lineal y de esta manera, determinar completamente el error e. Observamos también que $\mathrm{gr}(\omega) < \nu$.

Por último, para cada $i \in \{0, \ldots, n-1\}$, el i-ésimo síndrome S_i de un polinomio recibido $y = \sum_{j=0}^{n-1} y_j x^j$ se define como el resto de la división por la izquierda de y por $x - \sigma^i(\beta)$. Observamos que S_i es la evaluación por la derecha de y en $\sigma^i(\beta)$. Siempre que $i \in \{0, \ldots, 2\tau - 1\}$ las evaluaciones por la derecha de la palabra código c son cero, y se sigue que

$$
\begin{aligned}
S_i &= \sum_{j=0}^{n-1} y_j N_j(\sigma^i(\beta)) \\
&= \sum_{j=1}^{\nu} e_j N_{k_j}(\sigma^i(\beta)) \\
&= \sum_{j=1}^{\nu} e_j \sigma^i(N_{k_j}(\beta)) \\
&= \sum_{j=1}^{\nu} e_j \sigma^i(\alpha^{-1}) \sigma^{k_j+i}(\alpha) \\
&= \sigma^i(\alpha^{-1}) \sum_{j=1}^{\nu} e_j \sigma^{k_j+i}(\alpha).
\end{aligned}
$$

Por tanto,

$$
\sigma^i(\alpha) S_i = \sum_{j=1}^{\nu} e_j \sigma^{k_j+i}(\alpha) \tag{5.1}
$$

y llamamos al polinomio $S = \sum_{i=0}^{2\tau-1} \sigma^i(\alpha) S_i x^i$ el *polinomio de síndromes* de y.

Teorema 5.3. *El polinomio localizador de errores λ y el polinomio evaluador de errores ω satisfacen la siguiente ecuación clave no conmutativa*

$$
\omega = S\lambda + x^{2\tau} u,
$$

donde $u \in \mathbb{F}(t)[x;\sigma]$ es de grado menor que ν.

Demostración. En primer lugar, observamos que $\mathbb{F}(t)[x;\sigma]$ puede verse como un subanillo del anillo sesgado de serie de potencias formales $\mathbb{F}(t)[[x;\sigma]]$ (ver [26, Capítulo 1, Sección 4]). Dado $1 - ax \in \mathbb{F}(t)[x;\sigma]$ con $a \in \mathbb{F}(t)$ se tiene que en el anillo $\mathbb{F}(t)[[x;\sigma]]$, $(1-ax)^{-1} = \sum_{i\geq 0} N_i(a) x^i$. Por tanto,

$$
p_j = (1 - \sigma^{k_j}(\beta)x)^{-1}\lambda = \sum_{i\geq 0} N_i(\sigma^{k_j}(\beta)) x^i \lambda
$$

para cualquier $j \in \{1, \ldots, \nu\}$. Entonces,

$$
\begin{aligned}
\omega &= \sum_{j=1}^{\nu} e_j \sigma^{k_j}(\alpha) p_j \\
&= \sum_{j=1}^{\nu} e_j \sigma^{k_j}(\alpha) \sum_{i\geq 0} N_i(\sigma^{k_j}(\beta)) x^i \lambda \\
&= \sum_{i\geq 0} \left(\sum_{j=1}^{\nu} e_j \sigma^{k_j}(\alpha) N_i(\sigma^{k_j}(\beta)) \right) x^i \lambda \\
&= \sum_{i\geq 0} \left(\sum_{j=1}^{\nu} e_j \sigma^{k_j}(\alpha) \sigma^{k_j}(N_i(\beta)) \right) x^i \lambda \qquad \text{(Proposición 4.5)} \\
&= \sum_{i\geq 0} \left(\sum_{j=1}^{\nu} e_j \sigma^{k_j}(\alpha) \sigma^{k_j}(\alpha^{-1}\sigma^i(\alpha)) \right) x^i \lambda \\
&= \sum_{i\geq 0} \left(\sum_{j=1}^{\nu} e_j \sigma^{k_j+i}(\alpha) \right) x^i \lambda \\
&= \sum_{i=0}^{2\tau-1} \left(\sum_{j=1}^{\nu} e_j \sigma^{k_j+i}(\alpha) \right) x^i \lambda + \sum_{i\geq 2\tau} \left(\sum_{j=1}^{\nu} e_j \sigma^{k_j+i}(\alpha) \right) x^i \lambda \\
&= \sum_{i=0}^{2\tau-1} \sigma^i(\alpha) S_i x_i \lambda + x^{2\tau} \sum_{h\geq 0} \left(\sum_{j=1}^{\nu} \sigma^{-2\tau}(e_j) \sigma^{k_j+h}(\alpha) \right) x^h \lambda \qquad \text{(donde se ha utilizado (5.1))} \\
&= S\lambda + x^{2\tau} \sum_{j=1}^{\nu} \sigma^{-2\tau}(e_j) \sum_{h\geq 0} \sigma^{k_j+h}(\alpha) x^h \lambda \\
&= S\lambda + x^{2\tau} \sum_{j=1}^{\nu} \sigma^{-2\tau}(e_j) \sigma^{k_j}(\alpha) \sum_{h\geq 0} N_h(\sigma^{k_j}(\alpha)) x^h \lambda \\
&= S\lambda + x^{2\tau} \sum_{j=1}^{\nu} \sigma^{-2\tau}(e_j) \sigma^{k_j}(\alpha) (1 - \sigma^{k_j}(\alpha)x)^{-1} \lambda \\
&= S\lambda + x^{2\tau} \sum_{j=1}^{\nu} \sigma^{-2\tau}(e_j) \sigma^{k_j}(\alpha) p_j \\
&= S\lambda + x^{2\tau} u,
\end{aligned}
$$

donde $u = \sum_{j=1}^{\nu} \sigma^{-2\tau}(e_j)\sigma^{k_j}(\alpha)p_j$.

□

Procedemos ahora a resolver la ecuación clave. Para ello, haremos algo parecido a lo que hicimos en la sección 2.4, con la diferencia de que esta vez utilizaremos el algoritmo extendido de Euclides por la derecha (AEED), el cual está detallado en 5.

Teorema 5.4. *La ecuación clave no conmutativa*

$$x^{2\tau}u + S\lambda = \omega \tag{5.2}$$

es un múltiplo por la izquierda de la ecuación

$$x^{2\tau}u_I + Sv_I = r_I \tag{5.3}$$

donde u_I, v_I y r_I son los coeficientes de Bezout devueltos por el AEED con $x^{2\tau}$ y S como entradas, e I es el índice determinado por las condiciones $\mathrm{gr}(r_{I-1}) \geq \tau$ y $\mathrm{gr}(r_I) < \tau$. En particular, $\lambda = v_I g$ y $\omega = r_I g$ para algún $g \in \mathbb{F}(t)[x;\sigma]$.

Demostración. Recordemos que $\mathrm{gr}(S) < 2\tau$, $\mathrm{gr}(\lambda) \leq \tau$ y $\mathrm{gr}(\omega) < \nu \leq \tau$, consecuentemente $\mathrm{gr}(u) < \tau$. Por otro lado, por el Lema 4.3*(vi)*, $\mathrm{gr}(v_I) + \mathrm{gr}(r_{I-1}) = 2\tau$, por tanto, $\mathrm{gr}(v_I) \leq \tau$.

Consideremos el mínimo común múltiplo por la derecha $[\lambda, v_I]_r = \lambda a = v_I b$, donde $a, b \in \mathbb{F}(t)[x;\sigma]$ con $\mathrm{gr}(a) \leq \mathrm{gr}(v_I) \leq \tau$ y $\mathrm{gr}(b) \leq \mathrm{gr}(\lambda) \leq \tau$. Entonces, $(a,b)_r = 1$. Por tanto, multiplicando por la derecha (5.2) por a y (5.3) por b obtenemos:

$$x^{2\tau}ua + S\lambda a = \omega a \tag{5.4}$$

y

$$x^{2\tau}u_I b + Sv_I b = r_I b \tag{5.5}$$

Restando las ecuaciones (5.4) y (5.5) obtenemos

$$x^{2\tau}(ua - u_I b) = \omega a - r_I b$$

Esta igualdad, comparando grados, solo puede darse si $ua = u_I b$ y $\omega a = r_I b$. Como $(a,b)_r = 1$, se tiene que $[u, u_I]_r = ua = u_I b$ y $[\omega, a]_r = \omega a = r_I b$. En particular, $\mathrm{gr}(a) \leq \mathrm{gr}(r_I) < \tau$. Consideramos el mínimo común múltiplo por la izquierda $[a,b]_\ell = a'a = b'b$. Como $[\lambda, v_I]_r$ es un múltiplo por la izquierda de a y b, existe $m \in \mathbb{F}(t)[x;\sigma]$ tal que $[\lambda, v_I]_r = m[a,b]_\ell$. Así, $\lambda a = v_I b = ma'a = mb'b$. Por tanto, $\lambda = ma'$ y $v_I = mb'$, y por ser mínimo común múltiplo, $(\lambda, v_I)_\ell = m$. De manera análoga, se prueba que existen $m', m'' \in \mathbb{F}(t)[x;\sigma]$ tales que $u_I = m'b'$ y $u = m'a'$, y que $r_I = m''b'$ y $\omega = m''a$. Sin embargo, por el Lema 4.3*(v)*, $(u_I, v_I)_r = 1$, por tanto $b' = 1$. De esta forma, $b = a'a$ y obtenemos que $\lambda = v_I a'$,$\omega = r_I a'$ y $u = u_I a'$. Lo que demuestras que (5.2) es un múltiplo por la izquierda de la ecuación (5.3), tal y como buscábamos.

□

Observamos que si $(\lambda, \omega)_r = 1$, entonces el Teorema 5.4 nos da un procedimiento algorítmico para calcular tanto el polinomio localizador de errores como el polinomio evaluador de errores. Sin embargo, a diferencia de lo que pasaba con los códigos de bloque, existen algunos polinomios no conmutativos que pueden tener máximo común divisor no trivial. En [11] se demuestra que, como consecuencia del Lema 4.13, esta condición es equivalente a que el grado de v_I sea mayor que el número de posiciones de error encontradas. No obstante, es bastante raro que

el algoritmo 6 falle al decodificar. Existen métodos para tratar estos errores y decodificar los mensajes aun cuando ocurra este error, al que denominamos *error de ecuación clave*. Sin embargo, en [11, Th. 19] se demuestra que como consecuencia de trabajar en el cuerpo infinito $\mathbb{F}(t)$, la probabilidad teórica de que esto ocurra es cero y, por tanto, no merece la pena implementar métodos para corregirlo.

Algoritmo 6 Algoritmo de Sugiyama para códigos convolucionales sesgados RS

Entrada: Un polinomio recibido $y = \sum_{i=0}^{n-1} y_i x^i$ obtenido tras la transmisión de una palabra código $c \in \mathcal{C}$, donde $\mathcal{C}$ es un código convolucional RS sesgado generado por $g = \left[\{x - \sigma^i(\beta)\}_{i=0,\dots,\delta-2}\right]$ con capacidad de corrección de $\tau = \lfloor \frac{\delta-1}{2} \rfloor$ errores.
Salida: Una palabra código c', o *error de ecuación clave*.

Para $0 \leq i \leq 2\tau - 1$ **hacer**
  $S_i \leftarrow \sum_{j=0}^{n-1} y_j N_j(\sigma^i(\beta))$
$S \leftarrow \sum_{i=0}^{2\tau-1} \sigma^i(\alpha) S_i x^i$
si $S = 0$ **entonces**
  devolver y
$\{u_i, v_i, r_i\}_{i=0,\dots,l} \leftarrow \text{AEED}(x^{2\tau}, S)$
$I \leftarrow$ primera iteración en AEED con $\text{gr}(r_i) < \tau$
$pos \leftarrow \emptyset$
Para $0 \leq i \leq n - 1$ **hacer**
  si $\sigma^{i-1}(\beta^{-1})$ es una raíz por la izquierda de v_I **entonces**
    $pos = pos \cup \{i\}$
si $\deg v_I > \text{Cardinal}(pos)$ **entonces**
  devolver *error ecuación clave*
Para $j \in pos$ **hacer**
  $p_j \leftarrow \text{rquo}(v_I, 1 - \sigma^j(\beta)x)$
Resolver el sistema lineal $r_I = \sum_{j \in pos} e_j \sigma^j(\alpha) p_j$
$e \leftarrow \sum_{j \in pos} e_j x^j$
devolver $y - e$

Ejemplo 5.5. Sea $\mathbb{F} = \mathbb{F}_8$ el cuerpo finito de ocho elementos generado sobre $\mathbb{F}_2$ por un elemento primitivo a con $a^3 + a + 1 = 0$. Sea $\sigma : \mathbb{F}(t) \mapsto \mathbb{F}(t)$ el automorfismo definido por $\sigma(t) = \frac{1}{t+a}$.

El orden de σ es 9 y, por tanto, operaremos en el anillo $\mathcal{R} = \mathbb{F}(t)[x;\sigma]/\langle x^9 - 1\rangle$.

Tenemos que el elemento $\alpha = t$ da lugar a una base normal $\{\alpha, \sigma(\alpha), \dots, \sigma^8(\alpha)\}$ ya que

$$\begin{vmatrix} t & \sigma(t) & \sigma^2(t) & \cdots & \sigma^8(t) \\ \sigma(t) & \sigma^2(t) & \sigma^3(t) & \cdots & t \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \sigma^8(t) & t & \sigma(t) & \cdots & \sigma^7(t) \end{vmatrix} \neq 0.$$

Si $\beta = \alpha^{-1}\sigma(\alpha) = \frac{1}{t^2+at}$, el polinomio $g = \left[\{x - \sigma^i(\beta)\}_{i=0,1,2,3,4}\right]_\ell$ genera un $[9,5]$-código convolucional sesgado Reed-Solomon $\mathcal{C}$. Su distancia de Hamming, en virtud del Teorema 4.15, es $\delta = 5$ y puede corregir hasta $\tau = 2$ errores.

El polinomio generador de $\mathcal{C}$ es

$$\begin{aligned} g = x^4 &+ \left(\frac{(a^2+1)t^5 + (a+1)t^4 + at + a^2 + a + 1}{t^5 + (a^2+1)t^4 + (a^2+a+1)t + a^2 + a} \right) x^3 \\ &+ \left(\frac{(a+1)t^6 + (a+1)t^5 + (a^2+a+1)t^4 + t^2 + t + a^2}{t^6 + t^5 + at^4 + (a^2+a+1)t^2 + (a^2+a+1)t + a^2 + 1} \right) x^2 \\ &+ \left(\frac{a^2t^7 + (a^2+a+1)t^6 + (a^2+1)t^5 + (a^2+a+1)t^4}{t^7 + at^6 + t^5 + (a^2+a)t^4 + (a^2+a+1)t^3 + (a^2+1)t^2 + (a^2+a+1)t + a^2} \right) x \\ &+ \left(\frac{(a^2+1)t^6 + (a^2+a)t^5 + (a^2+a)t^4 + t^3 + (a^2+1)t^2 + t + a}{t^7 + (a^2+a+1)t^6 + at^5 + a^2t^4 + (a^2+a+1)t^3 + (a+1)t^2 + (a^2+1)t + 1} \right) \end{aligned}$$

Supongamos que se recibe el mensaje

$$\begin{aligned} y = x^4 &+ \left(\frac{(a^2+1)t^5 + (a+1)t^4 + at + a^2 + a + 1}{t^5 + (a^2+1)t^4 + (a^2+a+1)t + a^2 + a} \right) x^3 \\ &+ \left(\frac{(a+1)t^6 + (a+1)t^5 + (a^2+a+1)t^4 + t^2 + t + a^2}{t^6 + t^5 + at^4 + (a^2+a+1)t^2 + (a^2+a+1)t + a^2 + 1} \right) x^2, \end{aligned}$$

el cual consiste en el polinomio generador sin los términos de grado 1 y 0. De esta manera, hay errores en dos posiciones y el algoritmo de Sugiyama debería ser capaz de corregirlos.

En primer lugar, calculamos el polinomio de síndromes:

$$\begin{aligned} S = \sum_{i=0}^{2\tau-1} \sigma^i(\alpha) S_i x^i = &\left(\frac{(a^2+a+1)t^7 + (a+1)t^6 + a^2t^5 + (a^2+1)t^4 + (a^2+1)t^3 + t^2 + at + a^2 + 1}{t^7 + at^6 + t^5 + (a^2+a)t^4 + (a^2+a+1)t^3 + (a^2+1)t^2 + (a^2+a+1)t + a^2} \right) x^3 \\ &+ \left(\frac{(a^2+a)t^8 + (a^2+a)t^7 + at^6 + at^5 + a^2t^4 + (a^2+1)t^3 + (a+1)t^2 + at + a^2 + a + 1}{t^8 + (a+1)t^7 + (a+1)t^6 + (a^2+a+1)t^5 + t^4 + at^3 + at^2 + (a+1)t + a^2} \right) x^2 \\ &+ \left(\frac{at^7 + (a+1)t^6 + a^2t^5 + (a+1)t^3 + (a^2+1)t^2 + (a^2+1)t + a^2 + 1}{t^7 + (a+1)t^5 + at^4 + (a^2+a+1)t^3 + at + a^2 + 1} \right) x \\ &+ \left(\frac{(a^2+1)t^7 + a^2t^6 + a^2t^5 + t^4 + a^2t^3 + a^2t^2 + t}{t^7 + (a+1)t^5 + at^4 + (a^2+a+1)t^3 + at + a^2 + 1} \right) \end{aligned}$$

Ahora, aplicamos el algoritmo extendido de Euclides por la derecha hasta que obtengamos un resto de grado menor que $\tau = 2$, y obtenemos que las raíces por la izquierda de v_I en el conjunto $\{\sigma^{i-1}(\beta^{-1})\}_{i=0,\dots,8}$ son $\sigma^{-1}(\beta^{-1})$ y $\sigma^0(\beta^{-1}) = \beta^{-1}$. Por tanto, existen errores en las posiciones 0 y 1.

Ahora debemos resolver el sistema lineal

$$r_I = e_0\sigma^0(t)p_0 + e_1\sigma(t)p_1.$$

Una vez resuelto, obtenemos que los valores del error son:

$$e_0 = \frac{(a^2+1)t^6 + (a^2+a)t^5 + (a^2+a)t^4 + t^3 + (a^2+1)t^2 + t + a}{t^7 + (a^2+a+1)t^6 + at^5 + a^2t^4 + (a^2+a+1)t^3 + (a+1)t^2 + (a^2+1)t + 1}$$

$$e_1 = \frac{a^2t^7 + (a^2+a+1)t^6 + (a^2+1)t^5 + (a^2+a+1)t^4}{t^7 + at^6 + t^5 + (a^2+a)t^4 + (a^2+a+1)t^3 + (a^2+1)t^2 + (a^2+a+1)t + a^2}$$

Por tanto, $e = e_0 + e_1 x$ y el polinomio corregido es:

$$\begin{aligned} y - e = x^4 &+ \left(\frac{(a^2+1)t^5 + (a+1)t^4 + at + a^2 + a + 1}{t^5 + (a^2+1)t^4 + (a^2+a+1)t + a^2 + a} \right) x^3 \\ &+ \left(\frac{(a+1)t^6 + (a+1)t^5 + (a^2+a+1)t^4 + t^2 + t + a^2}{t^6 + t^5 + at^4 + (a^2+a+1)t^2 + (a^2+a+1)t + a^2 + 1} \right) x^2 \\ &+ \left(\frac{a^2t^7 + (a^2+a+1)t^6 + (a^2+1)t^5 + (a^2+a+1)t^4}{t^7 + at^6 + t^5 + (a^2+a)t^4 + (a^2+a+1)t^3 + (a^2+1)t^2 + (a^2+a+1)t + a^2} \right) x \\ &+ \left(\frac{(a^2+1)t^6 + (a^2+a)t^5 + (a^2+a)t^4 + t^3 + (a^2+1)t^2 + t + a}{t^7 + (a^2+a+1)t^6 + at^5 + a^2t^4 + (a^2+a+1)t^3 + (a+1)t^2 + (a^2+1)t + 1} \right) = g, \end{aligned}$$

tal y como buscábamos.

Conclusiones

Este trabajo ha logrado cumplir satisfactoriamente todos los objetivos que fueron planteados en la introducción.

En el ámbito de las Matemáticas, se han estudiado en profundidad las nociones básicas de los códigos de bloque. También se ha explorado la estructura algebraica de los códigos convolucionales y su ciclicidad mediante los anillos de polinomios sesgados, así como los códigos convolucionales sesgados RS. Además, se ha expuesto en detalle el algoritmo de Sugiyama para los códigos convolucionales sesgados RS, demostrando su eficacia para la corrección de errores.

En el ámbito de la Ingeniería Informática, se ha implementado en SageMath el algoritmo de Sugiyama para la decodificación de códigos BCH y códigos convolucionales sesgados RS. También se ha implementado un método para la construcción y codificación de estos códigos. Además, hemos probado su eficacia mediante herramientas como PyTest.

Con la realización de este trabajo se han consolidado conocimientos adquiridos a lo largo del doble grado, especialmente en teoría de anillos, teoría de cuerpos y teoría de la información, así como se han incorporado otros nuevos: teoría de módulos, álgebra no conmutativa y teoría de códigos.

Como posible vía futura del trabajo, sería interesante contribuir al proyecto SageMath integrando las implementaciones de los algoritmos de decodificación de Sugiyama y los sistemas de construcción y codificación de códigos convolucionales sesgados RS en el repositorio oficial de SageMath.

En resumen, el Trabajo de Fin de Grado ha sido más que una tarea que requiere tiempo y esfuerzo; ha sido una experiencia académica muy satisfactoria.

A. Implementación en SageMath del Algoritmo de Sugiyama

En este anexo vamos a documentar las clases desarrolladas para la implementación en SageMath del algoritmo de Sugiyama, tanto en su versión para códigos BCH como para códigos convolucionales sesgados Reed-Solomon.

En el caso del algoritmo de Sugiyama para códigos BCH se ha implementado un decodificador para códigos BCH, `BCHSugiyamaDecoder`, que hereda de la clase `Decoder`. Por otro lado, para la construcción de códigos convolucionales sesgados RS se ha implementado la clase `SkewRSConvolutionalCode` que hereda de `AbstractLinearCode` y utiliza una implementación de los polinomios sesgados de SageMath. Para la codificación de mensajes, se ha creado la clase `SkewRSConvolutionalEncoder`, que hereda de `Encoder` y es capaz de proporcionar una palabra código a partir de un mensaje mediante el polinomio generador del código. Por último, se ha implementado un decodificador `SkewRSSugiyamaDecoder` que hereda de `Decoder` y utiliza el algoritmo de Sugiyama para decodificar las palabras código recibidas.

El uso de estas clases es sencillo, únicamente se deben cargar los ficheros proporcionados mediante el comando `load()` de SageMath tal y como se indica a continuación:

```
sage: load("bch-sugiyama.sage")
sage: load("sccc-sugiyama.sage")
```

A.1. Decodificador para códigos BCH basado en algoritmo de Sugiyama

En esta sección vamos a describir la clase `BCHSugiyamaDecoder`, con la cual podemos *decodificar códigos BCH* utilizando el algoritmo de Sugiyama para códigos BCH descrito en 2.46.

`class BCHSugiyamaDecoder(self,code)`

Hereda de: `Decoder`

Construye un decodificador para códigos BCH utilizando el algoritmo de Sugiyama para códigos BCH.

ARGUMENTOS

`code` Código BCH asociado a este decodificador, instancia de `BCHCode`.

EJEMPLOS

```
sage: F = GF(2)
sage: C = BCHCode(F,15,7,offset = 1)
sage: D = BCHSugiyamaDecoder(C)
sage: D
> Sugiyama decoder for [15, 5] BCH Code over GF(2) with designed
    distance 7
```

correction_capability(self)

Calcula el número de errores que el código asociado a `self` puede corregir.

Salida

Número de errores que el código puede corregir.

Ejemplos

```
sage: F = GF(2)
sage: C = BCHCode(F,15,7,offset = 1)
sage: D = BCHSugiyamaDecoder(C)
sage: D.correction_capability()
> 3
```

decode_to_code(self,word)

Corrige los errores de `word` y devuelve una palabra código asociada al código de `self`.

Argumentos

`word` Polinomio que representa el mensaje recibido que queremos decodificar.

Salida

Palabra código decodificada, correspondiente a la palabra código corregida.

Ejemplos

```
sage: F = GF(2)
sage: C = BCHCode(F,15,7,offset = 1)
sage: D = BCHSugiyamaDecoder(C)
sage: message = vector([1,0,1,0,1])
sage: codeword = message * C.generator_matrix()
sage: codeword
> (1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1)
sage: R = PolynomialRing(GF(2) ,'x')
sage: polynomial_codeword = R(list(codeword))
sage: polynomial_codeword
> x^14 + x^9 + x^7 + x^4 + x^3 + x + 1
sage: error = vector([1, 1, 0, 0, 0, 0, 0, 0 , 0, 1, 0, 0, 0, 0,
    0])
sage : received_message = codeword + error
sage : received_message
```

```
> (0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1)
sage: y = R(list(received_message))
sage: y
> x^14 + x^7 + x^4 + x^3
sage: decoded_codeword = D.decode_to_code(y)
sage: decoded_codeword
> x^14 + x^9 + x^7 + x^4 + x^3 + x + 1
sage: decoded_codeword == polynomial_codeword
> True
```

A.2. Clase para códigos convolucionales sesgados RS

Esta clase implementa un sistema de construcción de códigos convolucionales sesgados Reed-Solomon. Es decir, proporciona métodos para el cálculo del código convolucional cíclico sesgado definido por un polinomio generador perteneciente al anillo de polinomios de Ore sobre el cuerpo de fracciones correspondiente.

`class SkewRSConvolutionalCode(self,generator_pol=None,roots=None,inverse_aut=None)`
Hereda de: `AbstractLinearCode`

Representación de un código convolucional sesgado Reed-Solomon.

Argumentos

`generator_pol` (por defecto: `None`) Polinomio generador del código.

`roots` (por defecto: `None`) Una lista de raíces cuyo mínimo común múltiplo por la izquierda conforma el polinomio generador del código.

`inverse_aut` (por defecto: `None`) El inverso del automorfismo utilizado para definir la extensión de Ore.

`alpha` (por defecto: `None`) Un elemento tal que $\{\alpha, \sigma(\alpha), \ldots, \sigma^{n-1}(\alpha)\}$ forma una base normal de la extensión de cuerpos $\mathbb{F}(t)^{\sigma} \subseteq \mathbb{F}(t)$, siendo n es la longitud del código.

Ejemplos

```
sage: F = GF(8, 'a')
sage: R = PolynomialRing(F,'t')
sage: K.<t> = FractionField(R)
sage: aut = K.hom([(t+a)/t])
sage: inverse_aut = K.hom([a/(t-1)])
sage: P.<x> = SkewPolynomialRing(K, aut)
sage: P
> Ore Polynomial Ring in x over Fraction Field of Univariate Polynomial
    Ring in t over Finite Field in a of size 2^3 twisted by t |--> (t +
    a)/t
sage: alpha = t
sage: beta = alpha^(-1)*aut(alpha)
```

```
sage: roots = []
sage: for i in range(4):
sage:   aut_i = aut^i
sage:   roots.append(x - aut_i(beta))
sage: roots
> [x + (t + a)/t^2, x + ((a + 1)*t^2 + a*t)/(t^2 + a^2), x + (a*t^2 + (a
    + 1)*t + a^2 + 1)/(t^2 + a + 1),x + (a*t^2 + (a + 1)*t + a^2)/(t^2
   + a)]
sage: C = SkewRSConvolutionalCode(roots = roots,inverse_aut =
   inverse_aut, alpha = alpha)
sage: C
> [7,3] Skew Reed-Solomon Convolutional Code on Ore Polynomial Ring in x
    over Fraction Field of Univariate Polynomial Ring in t over Finite
   Field in a of size 2^3 twisted by t |--> (t + a)/t with designed
   distance 5.
```

generator_polynomial(self)

Devuelve un polinomio generador del código asociado.

Ejemplos

```
sage: F = GF(8, 'a')
sage: R = PolynomialRing(F,'t')
sage: K.<t> = FractionField(R)
sage: aut = K.hom([(t+a)/t])
sage: P.<x> = SkewPolynomialRing(K, aut)
sage: roots = [x + (t + a)/t^2, x + ((a + 1)*t^2 + a*t)/(t^2 + a^2)
    , x + (a*t^2 + (a + 1)*t + a^2 + 1)/(t^2 + a + 1), x + (a*t^2 +
   (a + 1)*t + a^2)/(t^2 + a)]
sage: C = SkewRSConvolutionalCode(roots = roots)
sage: C.generator_polynomial()
> x^4 + (((a^2 + a)*t + a^2)/(t^4 + a^2 + a + 1))*x^3 + ((a^2*t^6 +
    a*t^5 + a*t^4 + (a^2 + 1)*t^2 + (a^2 + a + 1)*t + a^2 + a + 1)
   /(t^5 + t^4 + (a^2 + a + 1)*t + a^2 + a + 1))*x^2 + (((a + 1)*t
   ^7 + a^2*t^6 + a^2*t^5 + (a^2 + a + 1)*t^3 + (a^2 + a)*t^2 + (a
   ^2 + a)*t)/(t^6 + a*t^5 + (a + 1)*t^4 + (a^2 + a + 1)*t^2 + (a^2
    + 1)*t + a))*x + (a^2*t^6 + (a^2 + a)*t^5 + a^2*t^4 + a^2*t^3 +
    a*t)/(t^6 + (a^2 + 1)*t^5 + t^4 + (a^2 + a + 1)*t^2 + (a^2 + a)
   *t + a^2 + a + 1)
```

polynomial_ring(self)

Devuelve el anillo de polinomios sesgados asociado al código.

Ejemplos

```
sage: F = GF(8, 'a')
sage: R = PolynomialRing(F,'t')
sage: K.<t> = FractionField(R)
sage: aut = K.hom([(t+a)/t])
```

```
sage: P.<x> = SkewPolynomialRing(K, aut)
sage: roots = [x + (t + a)/t^2, x + ((a + 1)*t^2 + a*t)/(t^2 + a^2)
    , x + (a*t^2 + (a + 1)*t + a^2 + 1)/(t^2 + a + 1), x + (a*t^2 +
    (a + 1)*t + a^2)/(t^2 + a)]
sage: C = SkewRSConvolutionalCode(roots = roots)
sage: C.polynomial_ring()
> Ore Polynomial Ring in x over Fraction Field of Univariate
    Polynomial Ring in t over Finite Field in a of size 2^3 twisted
    by t |--> (t + a)/t
```

base_field(self)

Devuelve el cuerpo de fracciones base del código.

EJEMPLOS

```
sage: F = GF(8, 'a')
sage: R = PolynomialRing(F,'t')
sage: K.<t> = FractionField(R)
sage: aut = K.hom([(t+a)/t])
sage: P.<x> = SkewPolynomialRing(K, aut)
sage: roots = [x + (t + a)/t^2, x + ((a + 1)*t^2 + a*t)/(t^2 + a^2)
    , x + (a*t^2 + (a + 1)*t + a^2 + 1)/(t^2 + a + 1), x + (a*t^2 +
    (a + 1)*t + a^2)/(t^2 + a)]
sage: C = SkewRSConvolutionalCode(roots = roots)
sage: C.base_field()
> Fraction Field of Univariate Polynomial Ring in t over Finite
    Field in a of size 2^3
```

length(self)

Devuelve la longitud del código.

EJEMPLOS

```
sage: F = GF(8, 'a')
sage: R = PolynomialRing(F,'t')
sage: K.<t> = FractionField(R)
sage: aut = K.hom([(t+a)/t])
sage: P.<x> = SkewPolynomialRing(K, aut)
sage: roots = [x + (t + a)/t^2, x + ((a + 1)*t^2 + a*t)/(t^2 + a
    ^2), x + (a*t^2 + (a + 1)*t + a^2 + 1)/(t^2 + a + 1), x + (a*t
    ^2 + (a + 1)*t + a^2)/(t^2 + a)]
sage: C = SkewRSConvolutionalCode(roots = roots)
sage: C.length()
> 7
```

dimension(self)

Devuelve la dimensión del código.

EJEMPLOS

```
sage: F = GF(8, 'a')
sage: R = PolynomialRing(F,'t')
sage: K.<t> = FractionField(R)
sage: aut = K.hom([(t+a)/t])
sage: P.<x> = SkewPolynomialRing(K, aut)
sage: roots = [x + (t + a)/t^2, x + ((a + 1)*t^2 + a*t)/(t^2 + a^2)
    , x + (a*t^2 + (a + 1)*t + a^2 + 1)/(t^2 + a + 1), x + (a*t^2 +
    (a + 1)*t + a^2)/(t^2 + a)]
sage: C = SkewRSConvolutionalCode(roots = roots)
sage: C.dimension()
> 3
```

designed_distance(self)

Devuelve la distancia mínima prevista del código

Ejemplos

```
sage: F = GF(8, 'a')
sage: R = PolynomialRing(F,'t')
sage: K.<t> = FractionField(R)
sage: aut = K.hom([(t+a)/t])
sage: P.<x> = SkewPolynomialRing(K, aut)
sage: roots = [x + (t + a)/t^2, x + ((a + 1)*t^2 + a*t)/(t^2 + a^2)
    , x + (a*t^2 + (a + 1)*t + a^2 + 1)/(t^2 + a + 1), x + (a*t^2 +
    (a + 1)*t + a^2)/(t^2 + a)]
sage: C = SkewRSConvolutionalCode(roots = roots)
sage: C.designed_distance()
> 5
```

automorphism(self)

Devuelve el automorfismo asociado al anillo de polinomios sesgados.

Ejemplos

```
sage: F = GF(8, 'a')
sage: R = PolynomialRing(F,'t')
sage: K.<t> = FractionField(R)
sage: aut = K.hom([(t+a)/t])
sage: P.<x> = SkewPolynomialRing(K, aut)
sage: roots = [x + (t + a)/t^2, x + ((a + 1)*t^2 + a*t)/(t^2 + a^2)
    , x + (a*t^2 + (a + 1)*t + a^2 + 1)/(t^2 + a + 1), x + (a*t^2 +
    (a + 1)*t + a^2)/(t^2 + a)]
sage: C = SkewRSConvolutionalCode(roots = roots)
sage: C.automorphism()
> Ring endomorphism of Fraction Field of Univariate Polynomial Ring
     in t over Finite Field in a of size 2^3
Defn: t |--> (t + a)/t
```

inverse_automorphism(self)

Devuelve el inverso del automorfismo asociado al anillo de polinomios sesgados.

EJEMPLOS

```
sage: F = GF(8, 'a')
sage: R = PolynomialRing(F,'t')
sage: K.<t> = FractionField(R)
sage: aut = K.hom([(t+a)/t])
sage: inverse_aut = a/(t-1)
sage: P.<x> = SkewPolynomialRing(K, aut)
sage: roots = [x + (t + a)/t^2, x + ((a + 1)*t^2 + a*t)/(t^2 + a^2)
    , x + (a*t^2 + (a + 1)*t + a^2 + 1)/(t^2 + a + 1), x + (a*t^2 +
    (a + 1)*t + a^2)/(t^2 + a)]
sage: C = SkewRSConvolutionalCode(roots = roots,inverse_aut =
    inverse_aut)
sage: C.inverse_automorphism()
> Ring endomorphism of Fraction Field of Univariate Polynomial Ring
     in t over Finite Field in a of size 2^3
Defn: t |--> a/(t + 1)
```

A.3. Codificador de códigos convolucionales sesgados RS

En esta sección presentaremos la implementación de un codificador para los códigos convolucionales sesgados RS. Esta clase nos proporcionará un método para codificar un mensaje.

`class SkewRSConvolutionalEncoder(self,code)`

Hereda de: `Encoder`

Representación de un codificador para un código convolucional sesgado Reed-Solomon mediante su polinomio generador.

ARGUMENTOS

code Código asociado a este codificador.

EJEMPLOS

```
sage: F = GF(8, 'a')
sage: R = PolynomialRing(F,'t')
sage: K.<t> = FractionField(R)
sage: aut = K.hom([(t+a)/t])
sage: P.<x> = SkewPolynomialRing(K, aut)
sage: roots = [x + (t + a)/t^2, x + ((a + 1)*t^2 + a*t)/(t^2 + a^2), x
    + (a*t^2 + (a + 1)*t + a^2 + 1)/(t^2 + a + 1), x + (a*t^2 + (a + 1)
    *t + a^2)/(t^2 + a)]
sage: C = SkewRSConvolutionalCode(roots = roots)
sage: SkewRSConvolutionalEncoder(C)
```

```
> SkewRSConvolutionalEncoder for a [7,3] Skew Reed-Solomon
    Convolutional Code on Ore Polynomial Ring in x over Fraction Field
    of Univariate Polynomial Ring in t over Finite Field in a of size
    2^3 twisted by t |--> (t + a)/t with designed distance 5.
```

encode(self,message_poly)

Codifica un mensaje dado en forma polinomial.

ARGUMENTOS

`message_poly` : Mensaje en forma polinomial.

SALIDA

Palabra código resultante de la codificación del mensaje.

EJEMPLOS

```
sage: F = GF(8, 'a')
sage: a = F.gen()
sage: R = PolynomialRing(F,'t')
sage: K.<t> = FractionField(R)
sage: aut = K.hom([(t+a)/t])
sage: P.<x> = SkewPolynomialRing(K, aut)
sage: roots = [x + (t + a)/t^2, x + ((a + 1)*t^2 + a*t)/(t^2 + a
    ^2), x + (a*t^2 + (a + 1)*t + a^2 + 1)/(t^2 + a + 1), x + (a*t
    ^2 + (a + 1)*t + a^2)/(t^2 + a)]
sage: C = SkewRSConvolutionalCode(roots = roots)
sage: E = SkewRSConvolutionalEncoder(C)
sage: E.encode(x + a)
> x^5 + ((a^2*t^4 + a*t^3 + a)/(t^4 + 1))*x^4 + ((a^2*t^10 + a^2*
    t^9 + (a^2 + a)*t^8 + a^2*t^6 + (a^2 + a + 1)*t^4 + a*t^2 + (a
    ^2 + a)*t + a)/(t^9 + (a^2 + a)*t^5 + (a^2 + a + 1)*t))*x^3 +
    ((t^10 + (a^2 + 1)*t^9 + (a^2 + 1)*t^8 + a*t^7 + t^6 + t^5 + a
    ^2*t^3 + t^2 + (a^2 + a)*t + a^2)/(t^10 + t^9 + (a^2 + a)*t^6
    + (a^2 + a)*t^5 + (a^2 + a + 1)*t^2 + (a^2 + a + 1)*t))*x^2 +
    (((a^2 + a)*t^11 + (a + 1)*t^10 + (a + 1)*t^9 + (a^2 + a + 1)*
    t^7 + (a + 1)*t^6 + t^5 + (a^2 + a + 1)*t^4 + a^2*t^3 + a^2*t
    ^2 + t + a + 1)/(t^10 + a*t^9 + (a + 1)*t^8 + (a^2 + a)*t^6 +
    (a^2 + a + 1)*t^5 + t^4 + (a^2 + a + 1)*t^2 + (a^2 + 1)*t + a)
    )*x + ((a + 1)*t^6 + (a^2 + a + 1)*t^5 + (a + 1)*t^4 + (a + 1)
    *t^3 + a^2*t)/(t^6 + (a^2 + 1)*t^5 + t^4 + (a^2 + a + 1)*t^2 +
     (a^2 + a)*t + a^2 + a + 1)
```

unencode(self,codeword_poly)

Devuelve el mensaje correspondiente a `codeword_poly`.

ARGUMENTOS

`codeword_poly` : Palabra código en forma polinomial.

Salida

Mensaje correspondiente a la palabra código.

Ejemplos

```
sage: F = GF(8, 'a')
sage: a = F.gen()
sage: R = PolynomialRing(F,'t')
sage: K.<t> = FractionField(R)
sage: aut = K.hom([(t+a)/t])
sage: P.<x> = SkewPolynomialRing(K, aut)
sage: roots = [x + (t + a)/t^2, x + ((a + 1)*t^2 + a*t)/(t^2 + a
    ^2), x + (a*t^2 + (a + 1)*t + a^2 + 1)/(t^2 + a + 1), x + (a*t
    ^2 + (a + 1)*t + a^2)/(t^2 + a)]
sage: C = SkewRSConvolutionalCode(roots = roots)
sage: E = SkewRSConvolutionalEncoder(C)
sage: codeword = x^5 + ((a^2*t^4 + a*t^3 + a)/(t^4 + 1))*x^4 + ((
    a^2*t^10 + a^2*t^9 + (a^2 + a)*t^8 + a^2*t^6 + (a^2 + a + 1)*t
    ^4 + a*t^2 + (a^2 + a)*t + a)/(t^9 + (a^2 + a)*t^5 + (a^2 + a
    + 1)*t))*x^3 + ((t^10 + (a^2 + 1)*t^9 + (a^2 + 1)*t^8 + a*t^7
    + t^6 + t^5 + a^2*t^3 + t^2 + (a^2 + a)*t + a^2)/(t^10 + t^9 +
     (a^2 + a)*t^6 + (a^2 + a)*t^5 + (a^2 + a + 1)*t^2 + (a^2 + a
    + 1)*t))*x^2 + (((a^2 + a)*t^11 + (a + 1)*t^10 + (a + 1)*t^9 +
     (a^2 + a + 1)*t^7 + (a + 1)*t^6 + t^5 + (a^2 + a + 1)*t^4 + a
    ^2*t^3 + a^2*t^2 + t + a + 1)/(t^10 + a*t^9 + (a + 1)*t^8 + (a
    ^2 + a)*t^6 + (a^2 + a + 1)*t^5 + t^4 + (a^2 + a + 1)*t^2 + (a
    ^2 + 1)*t + a))*x + ((a + 1)*t^6 + (a^2 + a + 1)*t^5 + (a + 1)
    *t^4 + (a + 1)*t^3 + a^2*t)/(t^6 + (a^2 + 1)*t^5 + t^4 + (a^2
    + a + 1)*t^2 + (a^2 + a)*t + a^2 + a + 1)
sage: E.unencode(codeword)
> x + a
```

A.4. Decodificador para códigos convolucionales cíclicos sesgados RS basado en algoritmo de Sugiyama

En esta sección se va a explicar la implementación en SageMath del decodificador para códigos convolucionales sesgados RS mediante el algoritmo de Sugiyama expuesto en 5.

`class SkewRSSugiyamaDecoder(self,code)`
Hereda de: `Decoder`

Construye un decodificador para códigos convolucionales de Reed-Solomon sesgados basado en el algoritmo de Sugiyama para códigos convolucionales sesgados RS.

Argumentos

`code` Código asociado al decodificador.

Ejemplos

```
sage: F = GF(8, 'a')
sage: a = F.gen()
sage: R = PolynomialRing(F,'t')
sage: K.<t> = FractionField(R)
sage: aut = K.hom([(t+a)/t])
sage: P.<x> = SkewPolynomialRing(K, aut)
sage: roots = [x + (t + a)/t^2, x + ((a + 1)*t^2 + a*t)/(t^2 + a^2), x +
     (a*t^2 + (a + 1)*t + a^2 + 1)/(t^2 + a + 1), x + (a*t^2 + (a + 1)*t
     + a^2)/(t^2 + a)]
sage: C = SkewRSConvolutionalCode(roots = roots)
sage: SkewRSSugiyamaDecoder(C)
> <A Sugiyama-like algorithm for [7,3] Skew Reed-Solomon Convolutional
    Code on Ore Polynomial Ring in x over Fraction Field of Univariate
    Polynomial Ring in t over Finite Field in a of size 2^3 twisted by t
     |--> (t + a)/t with designed distance 5 with correction capability
    of 2 errors>
```

correction_capability(self)

Devuelve la capacidad de corrección de errores del código.

Ejemplos

```
sage: F = GF(8, 'a')
sage: a = F.gen()
sage: R = PolynomialRing(F,'t')
sage: K.<t> = FractionField(R)
sage: aut = K.hom([(t+a)/t])
sage: P.<x> = SkewPolynomialRing(K, aut)
sage: roots = [x + (t + a)/t^2, x + ((a + 1)*t^2 + a*t)/(t^2 + a^2), x +
     (a*t^2 + (a + 1)*t + a^2 + 1)/(t^2 + a + 1), x + (a*t^2 + (a + 1)*t
     + a^2)/(t^2 + a)]
sage: C = SkewRSConvolutionalCode(roots = roots)
sage: D = SkewRSSugiyamaDecoder(C)
sage: D.correction_capability()
> 2
```

decode_to_code(self,word)

Corrige los errores de `word` y devuelve una palabra código del código asociado a `self`.

Argumentos

`word` Polinomio que representa el mensaje recibido que queremos decodificar.

Ejemplos

```
    sage: F = GF(8, 'a')
    sage: a = F.gen()
```

```
sage: R = PolynomialRing(F,'t')
sage: K.<t> = FractionField(R)
sage: aut = K.hom([(t+a)/t])
sage: P.<x> = SkewPolynomialRing(K, aut)
sage: roots = [x + (t + a)/t^2, x + ((a + 1)*t^2 + a*t)/(t^2 +
    a^2), x + (a*t^2 + (a + 1)*t + a^2 + 1)/(t^2 + a + 1), x + (
    a*t^2 + (a + 1)*t + a^2)/(t^2 + a)]
sage: C = SkewRSConvolutionalCode(roots = roots)
sage: D = SkewRSSugiyamaDecoder(C)
sage: g = C.generator_polynomial()
sage: error = ((t + 2*a)/(1+t))*x^3 + (t/(2*t + 1))*x
sage: received_word = g - error
sage: decoded_word = D.decode_to_code(received_word)
sage: decoded_word
> x^4 + (((a^2 + a)*t + a^2)/(t^4 + a^2 + a + 1))*x^3 + ((a^2*t
    ^6 + a*t^5 + a*t^4 + (a^2 + 1)*t^2 + (a^2 + a + 1)*t + a^2 +
     a + 1)/(t^5 + t^4 + (a^2 + a + 1)*t + a^2 + a + 1))*x^2 +
    (((a + 1)*t^7 + a^2*t^6 + a^2*t^5 + (a^2 + a + 1)*t^3 + (a^2
     + a)*t^2 + (a^2 + a)*t)/(t^6 + a*t^5 + (a + 1)*t^4 + (a^2 +
     a + 1)*t^2 + (a^2 + 1)*t + a))*x + (a^2*t^6 + (a^2 + a)*t^5
     + a^2*t^4 + a^2*t^3 + a*t)/(t^6 + (a^2 + 1)*t^5 + t^4 + (a
    ^2 + a + 1)*t^2 + (a^2 + a)*t + a^2 + a + 1)
sage: decoded_word == g
> True
```

A.5. Funciones auxiliares

Para la implementación de las clases anteriores, fue necesario el uso de funciones auxiliares. En esta sección, vamos a dar una breve descripción de ellas.

left_lcm(poly_list)
Calcula el mínimo común múltiplo por la izquierda de una lista de polinomios sesgados.

ARGUMENTOS

poly_list Lista de polinomios para los cuales se calculará el LCM por la izquierda.

SALIDA

El MCM por la izquierda de la lista de polinomios dada.

EJEMPLOS

```
sage: F = GF(8, 'a')
sage: a = F.gen()
sage: R = PolynomialRing(F,'t')
sage: K.<t> = FractionField(R)
sage: aut = K.hom([(t+a)/t])
sage: P.<x> = SkewPolynomialRing(K, aut)
```

```
sage: roots = [x + (t + a)/t^2, x + ((a + 1)*t^2 + a*t)/(t^2 + a^2), x +
     (a*t^2 + (a + 1)*t + a^2 + 1)/(t^2 + a + 1), x + (a*t^2 + (a + 1)*t
     + a^2)/(t^2 + a)]
sage: left_lcm(roots)
> x^4 + (((a^2 + a)*t + a^2)/(t^4 + a^2 + a + 1))*x^3 + ((a^2*t^6 + a*t
    ^5 + a*t^4 + (a^2 + 1)*t^2 + (a^2 + a + 1)*t + a^2 + a + 1)/(t^5 + t
    ^4 + (a^2 + a + 1)*t + a^2 + a + 1))*x^2 + (((a + 1)*t^7 + a^2*t^6 +
     a^2*t^5 + (a^2 + a + 1)*t^3 + (a^2 + a)*t^2 + (a^2 + a)*t)/(t^6 + a
    *t^5 + (a + 1)*t^4 + (a^2 + a + 1)*t^2 + (a^2 + 1)*t + a))*x + (a^2*
    t^6 + (a^2 + a)*t^5 + a^2*t^4 + a^2*t^3 + a*t)/(t^6 + (a^2 + 1)*t^5
    + t^4 + (a^2 + a + 1)*t^2 + (a^2 + a)*t + a^2 + a + 1)
```

order_of_automorphism(K, sigma)
Calcula el orden del automorfismo `sigma` en el cuerpo `K`.

ARGUMENTOS

`K` El cuerpo sobre el cual está definido el automorfismo.

`sigma` El automorfismo cuyo orden se va a calcular.

SALIDA

El orden del automorfismo `sigma`.

EJEMPLOS

```
sage: F = GF(8, 'a')
sage: a = F.gen()
sage: R = PolynomialRing(F,'t')
sage: K.<t> = FractionField(R)
sage: aut = K.hom([(t+a)/t])
sage: order_of_automorphism(K,aut)
> 7
```

jth_norm(gamma, sigma, j)
Calcula la j-ésima norma del elemento `gamma` usando el automorfismo `sigma`.

ARGUMENTOS

`gamma` El elemento para el cual se va a calcular la norma.

`sigma` El automorfismo utilizado en el cálculo de la norma.

`j` El orden de la norma.

SALIDA

La j-ésima norma de `gamma`.

EJEMPLOS

```
sage: F = GF(8, 'a')
sage: a = F.gen()
sage: R = PolynomialRing(F,'t')
sage: K.<t> = FractionField(R)
sage: aut = K.hom([(t+a)/t])
sage: alpha = t
sage: jth_norm(alpha, aut, 10)
> (a + 1)*t + a
```

normal_basis(aut, c, F)

Determina si la matriz construida utilizando el automorfismo `aut` y el elemento `c` tiene determinante nulo. Esto es equivalente a decir que $\{c, \sigma(c), \ldots, \sigma^{n-1}(c)\}$ forma una base normal de la extensión $\mathbb{F}(t)^{\sigma} \subseteq \mathbb{F}(t)$.

ARGUMENTOS

`F` Cuerpo de fracciones empleado.

`automorphism` Un automorfismo del cuerpo de fracciones `F`.

`c` Un elemento del cuerpo `F`.

SALIDA

`bool`: Verdadero si el determinante es distinto de cero, Falso si es cero.

EJEMPLOS

```
sage: F = GF(8, 'a')
sage: a = F.gen()
sage: R = PolynomialRing(F,'t')
sage: K.<t> = FractionField(R)
sage: aut = K.hom([(t+a)/t])
sage: c = t
sage: normal_basis(aut,c,K)
> True
```

RightED(f, g, inverse_aut)

Realiza la división euclídea por la derecha en el anillo de polinomios sesgados definido por $\mathbb{F}(t)[x;\sigma]$.

ARGUMENTOS

`f` El polinomio sesgado dividendo.

`g` El polinomio sesgado divisor.

`inverse_aut` El inverso del automorfismo con el que se define la extensión de Ore.

SALIDA

c: El polinomio cociente resultante de la división de `f` por `g`.

r: El polinomio resto resultante de la división de `f` por `g`.

EJEMPLOS

```
sage: F = GF(8, 'a')
sage: a = F.gen()
sage: R = PolynomialRing(F,'t')
sage: K.<t> = FractionField(R)
sage: aut = K.hom([(t+a)/t])
sage: inverse_aut = K.hom([a/(t-1)])
sage: P.<x> = SkewPolynomialRing(K, aut)
sage: f = (t^4*a + t^2*a^2 + a)/(t^5 + a)*x^4 + x^2 + 1
sage: g = (t^4*a + t^2*a^3 + t)/(t^3 + t*a)*x^2 + x + t
sage: quotient,reminder = RightED(f,g,inverse_aut)
sage: quotient
> ((a*t^8 + a^2*t^6 + (a^2 + 1)*t^4 + a^2 + 1)/(t^8 + t^7 + (a^2 + a)*t
    ^6 + t^5 + (a^2 + a)*t^4 + t^3 + t^2 + a^2*t + a^2 + a))*x^2 + (((a
    ^2 + 1)*t^9 + (a^2 + a)*t^8 + t^7 + (a^2 + a)*t^6 + t^5 + (a^2 + 1)*
    t^4 + a*t^2)/(t^9 + a*t^8 + a*t^7 + t^6 + (a^2 + 1)*t^5 + (a + 1)*t
    ^4 + (a^2 + a)*t^3 + (a + 1)*t^2 + (a^2 + 1)*t + a + 1))*x + ((a^2 +
     a)*t^12 + a*t^11 + (a^2 + a)*t^10 + (a + 1)*t^9 + (a^2 + a)*t^8 + a
    ^2*t^7 + t^6 + (a^2 + a)*t^5 + (a^2 + a)*t^4 + a^2*t^3 + (a^2 + a +
    1)*t^2 + a*t + a^2 + a + 1)/(t^12 + (a^2 + a)*t^11 + t^10 + (a^2 + a
    )*t^9 + a^2*t^8 + (a^2 + a + 1)*t^7 + t^6 + a^2*t^5 + t^4 + a^2*t^3
    + t^2 + (a^2 + a + 1)*t)
sage: reminder
> (((a^2 + 1)*t^14 + a*t^13 + t^11 + (a^2 + 1)*t^10 + a^2*t^8 + (a^2 + a
    )*t^7 + a*t^6 + t^5 + a*t^4 + (a + 1)*t^3 + (a + 1)*t^2 + (a^2 + 1)*
    t + a)/(t^13 + (a^2 + a + 1)*t^12 + (a^2 + a + 1)*t^11 + (a^2 + a +
    1)*t^9 + a*t^8 + (a^2 + a + 1)*t^7 + t^6 + a^2*t^4 + a^2*t^3 + (a^2
    + 1)*t^2 + a^2*t + a^2 + a))*x + ((a^2 + a)*t^12 + (a + 1)*t^11 + a*
    t^9 + (a^2 + a)*t^6 + (a^2 + a + 1)*t^5 + a*t^4 + (a^2 + 1)*t^3 + (a
     + 1)*t^2 + (a + 1)*t)/(t^11 + (a^2 + a)*t^10 + t^9 + (a^2 + a)*t^8
    + a^2*t^7 + (a^2 + a + 1)*t^6 + t^5 + a^2*t^4 + t^3 + a^2*t^2 + t +
    a^2 + a + 1)
sage: f == g*quotient + reminder
> True
```

LeftED(f, g, aut)

Realiza la división euclídea por la izquierda en el anillo de polinomios sesgados definido por $\mathbb{F}(t)[x;\sigma]$.

ARGUMENTOS

`f` El polinomio sesgado dividendo.

`g` El polinomio sesgado divisor.

`aut` El automorfismo utilizado para la definición del anillo de polinomios sesgados.

SALIDA

c: El polinomio cociente resultante de la división de `f` por `g`.

r: El polinomio resto resultante de la división de `f` por `g`.

EJEMPLOS

```
sage: F = GF(8, 'a')
sage: a = F.gen()
sage: R = PolynomialRing(F,'t')
sage: K.<t> = FractionField(R)
sage: aut = K.hom([(t+a)/t])
sage: P.<x> = SkewPolynomialRing(K, aut)
sage: f = (t^4*a + t^2*a^2 + a)/(t^5 + a)*x^4 + x^2 + 1
sage: g = (t^4*a + t^2*a^3 + t)/(t^3 + t*a)*x^2 + x + t
sage: quotient,reminder = LeftED(f,g,aut)
sage: quotient
> ((a*t^7 + a^2*t^6 + (a^2 + a)*t^5 + (a^2 + a + 1)*t^4 + (a^2 + a)*t^3
    + (a^2 + a + 1)*t^2 + a*t + a^2)/(t^8 + (a^2 + 1)*t^7 + (a^2 + a +
    1)*t^6 + a*t^3 + t^2 + (a^2 + 1)*t))*x^2 + (((a + 1)*t^8 + (a + 1)*t
    ^7 + a^2*t^6 + (a^2 + 1)*t^5 + (a^2 + a)*t^4 + (a^2 + 1)*t^3 + (a +
    1)*t + 1)/(t^8 + a*t^7 + (a^2 + 1)*t^5 + (a^2 + a)*t^4 + (a + 1)*t^3
     + (a^2 + a + 1)*t^2 + (a^2 + 1)*t + a^2 + 1))*x + (t^12 + (a^2 + 1)
    *t^11 + t^10 + a^2*t^9 + a^2*t^8 + (a^2 + a)*t^7 + a*t^6 + (a^2 + 1)
    *t^5 + a^2*t^4 + a^2 + a + 1)/(t^13 + t^12 + a*t^11 + a^2*t^10 + a*t
    ^9 + (a + 1)*t^8 + (a^2 + a)*t^7 + (a^2 + 1)*t^6 + (a + 1)*t^5 + a
    ^2*t^4 + a*t^3 + (a + 1)*t^2 + a*t)
sage: reminder
> (((a + 1)*t^13 + t^12 + a^2*t^11 + (a^2 + 1)*t^10 + (a^2 + a + 1)*t^9
    + (a^2 + a + 1)*t^8 + a^2*t^7 + t^6 + (a^2 + a)*t^5 + a*t^4 + (a^2 +
     1)*t^3 + t^2 + (a^2 + 1)*t + a^2)/(t^13 + t^12 + a*t^11 + a^2*t^10
    + a*t^9 + (a + 1)*t^8 + (a^2 + a)*t^7 + (a^2 + 1)*t^6 + (a + 1)*t^5
    + a^2*t^4 + a*t^3 + (a + 1)*t^2 + a*t))*x + (a^2*t^11 + (a + 1)*t^10
     + (a^2 + a)*t^8 + (a^2 + 1)*t^7 + a^2*t^6 + (a^2 + a + 1)*t^4 + a
    ^2*t^3 + a*t^2 + (a + 1)*t + a^2 + 1)/(t^12 + t^11 + a*t^10 + a^2*t
    ^9 + a*t^8 + (a + 1)*t^7 + (a^2 + a)*t^6 + (a^2 + 1)*t^5 + (a + 1)*t
    ^4 + a^2*t^3 + a*t^2 + (a + 1)*t + a)
sage: f == quotient*g + reminder
> True
```

`REEA(f, g, inverse_sigma)`

Realiza el Algoritmo de Euclides extendido por la derecha (REEA) en el anillo de polinomios sesgados definido por $\mathbb{F}(t)[x;\sigma]$.

ARGUMENTOS

`f` El primer polinomio sesgado.

`g` El segundo polinomio sesgado.

`inverse_sigma` El inverso del automorfismo utilizado para definir la extensión de Ore.

SALIDA

u: Lista de coeficientes para la identidad de Bézout tal que `f * u[i] + g * v[i] = r[i]`.

v: Lista de coeficientes para la identidad de Bézout tal que `u[i] * f + v[i] * g = r[i]`.

r: Lista de restos.

EJEMPLOS

```
sage: F = GF(8, 'a')
sage: a = F.gen()
sage: R = PolynomialRing(F,'t')
sage: K.<t> = FractionField(R)
sage: aut = K.hom([(t+a)/t])
sage: inverse_aut = K.hom([a/(t-1)])
sage: P.<x> = SkewPolynomialRing(K, aut)
sage: f = (t^4*a + t^2*a^2 + a)/(t^5 + a)*x^4 + x^2 + 1
sage: g = (t^4*a + t^2*a^3 + t)/(t^3 + t*a)*x^2 + x + t
sage: u,v,r = REEA(f,g,inverse_aut)
sage: f*u[2] + g*v[2] == r[2]
> True
```

A.6. Desarrollo de la implementación

Tanto el código desarrollado en este trabajo, como todos los ejemplos que se han expuesto previamente, pueden encontrarse en:

https://github.com/alejandroccb/TFG/tree/main/src

A.6.1. Organización del código

El árbol de directorios de todo el código y las pruebas realizadas es el siguiente:

`jupyter`

`ejemplos_codigo.ipynb`: Cuaderno de Jupyter donde se encuentran los ejemplos realizados durante la descripción del código.

`ejemplo_sugiyama.ipynb`: Cuaderno de Jupyter donde se explica paso por paso el Ejemplo 5.5.

`src`

`bch_sugiyama.sage`: Contiene la clase `BCHSugiyamaDecoder` para la decodificación de códigos BCH con el algoritmo de Sugiyama.

`sccc_sugiyama.sage`: Contiene las clases `SkewRSConvolutionalCode`: para la construcción de códigos convolucionales sesgados RS, la clase `SkewRSConvolutionalEncoder` para su codificación y por último la clase `SkewRSConvolutionalDecoder` para la decodificación con el algoritmo de Sugiyama.

`tests`

`test_bch_sugiyama.py`: Script de Pytest para pruebas.

`test_sccc_sugiyama.py`: Script de Pytest para pruebas.

`bch_sugiyama.py`: Traducción de `bch_sugiyama.sage` en Python.

`sccc_sugiyama.py`: Traducción de `sccc_sugiyama.sage` en Python.

A.6.2. Tests

Se han desarrollado distintos tests utilizando la herramienta Pytest [21]. Estos tests se pueden encontrar en los siguientes ficheros:

- `test_bch_sugiyama.py`
- `test_sccc_sugiyama.py`

Para utilizar Pytest es necesario traducir los ficheros de Sage a Python, por eso, también han de incluirse las traducciones en el mismo fichero.

La función principal de estos test es verificar mediante un gran número de pruebas, que las clases implementadas funcionan como se espera. Estos ficheros tienen predefinidos unos cuantos tipos de códigos. El funcionamiento consiste en generar varios mensajes aleatorios para cada código, cuyo tamaño viene determinado por la dimensión del código. Después, los mensajes se codifican y se calcula un vector de errores aleatorio que simula los fallos que se han producido durante la transmisión de las palabras código. Por último, la palabra código con errores se decodifica y se comprueba que coincide con la palabra código original.

B. Planificación y costes del proyecto

Para realizar este trabajo, se ha seguido una planificación guiada a través de un diagrama de Gantt, el cual se puede ver en la Figura B.1. En este diagrama, exponemos cada una de las fases del proyecto y un tiempo aproximado de cuánto ha durado cada una de ellas. Se empezó con un estudio de los fundamentos de la teoría de códigos, se siguió con la redacción de la memoria y por último, se implementó el código del trabajo.

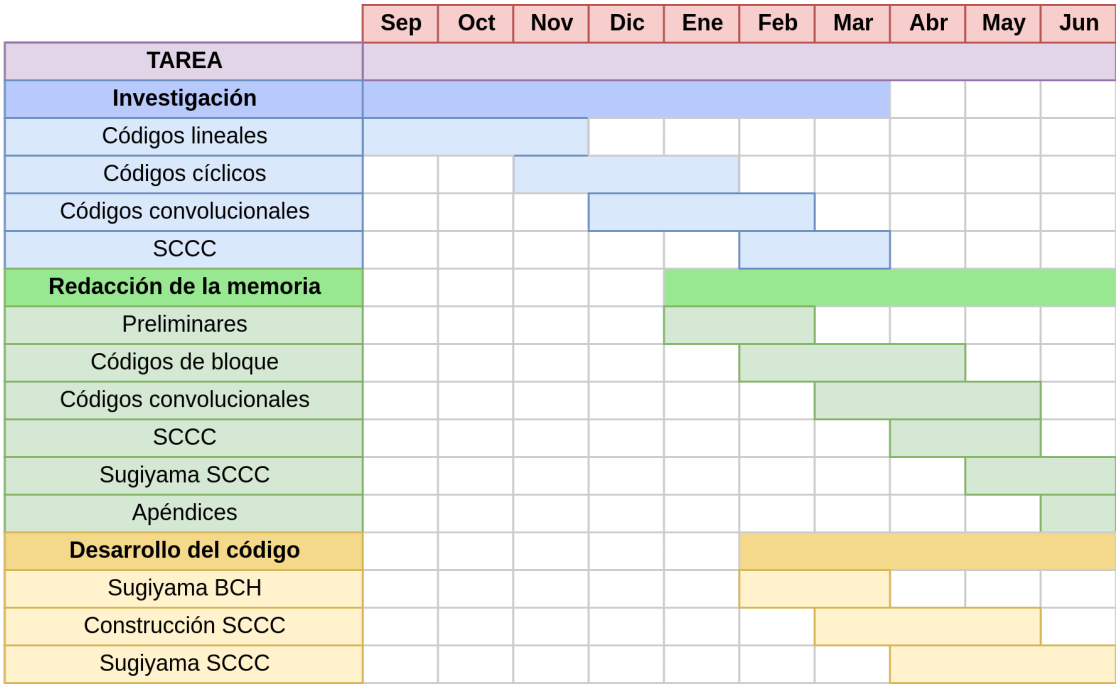

Figura B.1.: Planificación temporal de las etapas del proyecto.

Vamos a realizar también una aproximación del coste monetario de la creación de este proyecto, considerando su duración de 9 meses. Estos cálculos se realizarán teniendo en cuenta que se han trabajado 450 horas de acuerdo a los 18 créditos ECTS correspondientes asignados al TFG y con un gasto aproximado de 20 €/h.

Tipo	**Tareas**	**Cantidad (h)**	**Coste (€)**
Investigación	Códigos lineales	30	600
Investigación	Códigos cíclicos	30	600
Investigación	Códigos convolucionales	50	1000
Investigación	Códigos convolucionales cíclicos sesgados	40	800
Implementación del código	Sugiyama para BCH	30	600
Implementación del código	Construcción de códigos SCCC	40	800
Implementación del código	Sugiyama para SCCC	50	1000
Redacción de la memoria	Preliminares	20	400
Redacción de la memoria	Códigos de bloque	50	1000
Redacción de la memoria	Códigos convolucionales	50	1000
Redacción de la memoria	SCCC y Sugiyama para SCCC	60	1200
Total		450	9000

Tabla B.1.: Distribución de tareas y costos

La Tabla B.1 se puede observar cuánto tiempo ha requerido cada tarea junto al coste total calculado.

Bibliografía

Las referencias se listan por orden alfabético. Aquellas referencias con más de un autor están ordenadas de acuerdo con el primer autor.

[1] Anton Betten, Michael Braun, Harald Fripertinger, Adalbert Kerber, Axel Kohnert, and Alfred Wassermann. *Error-Correcting Linear Codes: Classification by Isometry and Applications*. Springer Berlin Heidelberg New York, 2006. [Citado en págs. 18 and 33]

[2] D. Boucher, W. Geiselmann, and F. Ulmer. Skew-cyclic codes. *Applicable Algebra in Engineering, Communication and Computing*, 18(4):379–389, 2007. [Citado en pág. 85]

[3] Matej Brešar. The wedderburn-artin theorem. 2024. [Citado en pág. 84]

[4] Lionel Chaussade, Pierre Loidreau, and Felix Ulmer. Skew codes of prescribed distance or rank. *Designs Codes and Cryptography*, 50, 03 2009. [Citado en pág. 85]

[5] P. Elias. Coding for noisy channels. *IRE Convention Records*, 3(4):37–46, 1955. [Citado en pág. 61]

[6] R. Fano. A heuristic discussion of probabilistic decoding. *IEEE Transactions on Information Theory*, 9(2):64–74, 1963. [Citado en pág. 61]

[7] G. Forney. Convolutional codes i: Algebraic structure. *IEEE Transactions on Information Theory*, 16(6):720–738, 1970. [Citado en págs. 18, 62, and 69]

[8] Heide Gluesing-Luerssen. Skew-polynomial rings and skew-cyclic codes, 2019. [Citado en págs. 18 and 73]

[9] Heide Gluesing-Luerssen and Wiland Schmale. On cyclic convolutional codes. *Acta Applicandae Mathematicae*, 82:183–237, 06 2004. [Citado en págs. 18, 30, 32, 62, 73, and 82]

[10] Heide Gluesing-Luerssen and Wiland Schmale. On cyclic convolutional codes. *Acta Applicandae Mathematica*, 82(2):183–237, 06 2004. [Citado en pág. 73]

[11] José Gómez-Torrecillas, Francisco Javier Lobillo, and Gabriel Navarro. A sugiyama-like decoding algorithm for convolutional codes. *IEEE Transactions on Information Theory*, 63(10):6216–6226, 2017. [Citado en págs. 18, 69, 73, 93, 97, and 98]

[12] Jaime Gutierrez and David Sevilla. Building counterexamples to generalizations for rational functions of ritt's decomposition theorem. *Journal of Symbolic Computation*, 2006. Received 26 April 2005; Available online 13 July 2006. Communicated by Bruno Salvy. [Citado en pág. 86]

[13] José Gómez-Torrecillas, F. J. Lobillo, and Gabriel Navarro. A new perspective of cyclicity in convolutional codes. *IEEE Transactions on Information Theory*, 62(5):2702–2706, 2016. [Citado en págs. 18 and 73]

[14] R. W. Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160, 1950. [Citado en pág. 15]

[15] W. Cary Huffman and Vera Pless. *Fundamentals of Error-Correcting Codes*. Cambridge University Press, 2010. [Citado en págs. 18, 19, 33, 43, 56, and 62]

[16] N. Jacobson. *Basic Algebra I: Second Edition*. Dover Books on Mathematics. Dover Publications, 2012. [Citado en pág. 86]

[17] N. Jacobson and A.M. Society. *The Theory of Rings*. Mathematical surveys and monographs. American Mathematical Society, 1943. [Citado en pág. 84]

[18] Nathan Jacobson and Paul Moritz Cohn. *Finite-dimensional division algebras over fields*. Springer-Verlag, Berlin Heidelberg, 2nd corrected printing edition, 1996. [Citado en págs. 18, 73, 74, 76, and 85]

[19] R. Johannesson and K.S. Zigangirov. *Fundamentals of Convolutional Coding: Second Edition*. 07 2015. [Citado en págs. 18 and 62]

[20] A.W. Knapp. *Basic Algebra*. Cornerstones. Birkhäuser Boston, 2006. [Citado en págs. 18 and 19]

[21] Holger Krekel, Bruno Oliveira, Ronny Pfannschmidt, Floris Bruynooghe, Brianna Laugher, and Florian Bruhin. pytest 7.4.0, 2004. [Citado en págs. 18 and 119]

[22] T. Y. Lam and A. Leroy. Vandermonde and wronskian matrices over division rings. *Journal of Algebra*, 119:308–336, Dec 1988. [Citado en pág. 86]

[23] T.Y Lam and A Leroy. Vandermonde and wronskian matrices over division rings. *Journal of Algebra*, 119(2):308–336, 1988. [Citado en pág. 79]

[24] André Leroy. Pseudolinear transformations and evaluation in ore extensions. *Bulletin of the Belgian Mathematical Society - Simon Stevin*, 2, 01 1995. [Citado en pág. 85]

[25] Julia Lieb, Raquel Pinto, and Joachim Rosenthal. Convolutional codes. *CoRR*, abs/2001.08281, 2020. [Citado en págs. 18 and 62]

[26] J.C. McConnell, J.C. Robson, and L.W. Small. *Noncommutative Noetherian Rings*. Graduate studies in mathematics. American Mathematical Society, 2001. [Citado en pág. 95]

[27] R. J. McEliece. The algebraic theory of convolutional codes. In V. S. Pless and W. C. Huffman, editors, *Handbook of Coding Theory*, pages 1065–1138. Elsevier, Amsterdam, 1998. [Citado en pág. 66]

[28] Øystein Ore. Theory of non-commutative polynomials. *Annals of Mathematics*, 34(3):480–508, 1933. [Citado en pág. 73]

[29] P. Piret. Structure and constructions of cyclic convolutional codes. *IEEE Transactions on Information Theory*, 22(2):147–155, 1976. [Citado en págs. 73 and 82]

[30] Ricardo A. Podestá. Introducción a la teoría de códigos autocorrectores, 2006. Disponible en: https://www.famaf.unc.edu.ar/documents/940/CMat35-3.pdf. [Citado en pág. 16]

[31] E. Prange. *Cyclic Error-correcting Codes in Two Symbols*. AFCRC-TN. Air Force Cambridge Research Center, 1957. [Citado en pág. 42]

[32] C. Roos. On the structure of convolutional and cyclic convolutional codes. *IEEE Transactions on Information Theory*, 25(6):676–683, 1979. [Citado en págs. 73, 82, and 93]

[33] Joachim Rosenthal and Roxana Smarandache. Maximum distance separable convolutional codes. *Applicable Algebra in Engineering, Communication and Computing*, 10(1):15–32, 1999. [Citado en pág. 71]

[34] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948. [Citado en pág. 15]

[35] Y. Sugiyama, M. Kasahara, S. Hirasawa, and T. Namekawa. A method for solving a key equation for decoding goppa codes. *Information and Control*, 27, 1975. [Citado en pág. 52]

[36] The Sage Developers. SageMath, The Sage Mathematics Software System (Version 9.5), 2022. Release Date: 2022-01-30, [Online; accessed 2024-03-08]. [Citado en págs. 18 and 58]

[37] B.L. van der Waerden. *Modern Algebra*. Number v. 1. Ungar, 1949. [Citado en pág. 83]

[38] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, 1967. [Citado en pág. 62]

[39] Robert Wisbauer. *Foundations of Module and Ring Theory: A Handbook for Study and Research*. 05 2018. [Citado en pág. 30]

[40] John M. Wozencraft. Sequential decoding for reliable communication. 1957. [Citado en pág. 61]