CRIPTOGRAFÍA POSCUÁNTICA BASADA EN CÓDIGOS CORRECTORES DE ERRORES

SANDRA INÉS MANRIQUE GUERRERO

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE CIENCIAS
ESCUELA DE MATEMÁTICAS
BUCARAMANGA
2024

CRIPTOGRAFÍA POSCUÁNTICA BASADA EN CÓDIGOS CORRECTORES DE ERRORES

SANDRA INÉS MANRIQUE GUERRERO

Trabajo de grado para optar al título de
Matemática

Director:
Wilson Olaya León
Profesor
Escuela de Matemáticas

Codirector:
Jorge Eliecer Gómez Ríos
Profesor
Escuela de Matemáticas

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE CIENCIAS
ESCUELA DE MATEMÁTICAS
BUCARAMANGA
2024

DEDICATORIA

A mis padres, hermanos y amigos, quienes han sido mi inspiración y apoyo incondicional. Gracias a su constante respaldo, sus consejos y al apoyo que me han brindado, he podido alcanzar este logro, que es casi tan suyo como mío. Estoy profundamente agradecida por todo lo que han hecho por mí.

AGRADECIMIENTOS

En primer lugar, quiero expresar mi sincero agradecimiento a Dios por ser mi fortaleza y guiarme en cada etapa de mi vida. Agradezco profundamente a mis padres por su constante apoyo y motivación a lo largo de toda mi carrera, así como por su esfuerzo incansable para darme siempre lo mejor.

También quiero expresar mi gratitud a mis hermanos, en especial a mi hermana Lina, quien siempre estuvo ahí para escucharme en los momentos difíciles.

Gracias a mis amigos, especialmente a Stefanny, Wilmar, Jhonatan, Karen, Marlon y Daniel, por sus valiosos consejos, cariño y compañía; los llevo en el corazón.

Agradezco a mis compañeros de carrera por hacer este camino más ameno, y a mis profesores y a la universidad por formarme y transmitir tanto amor por mi carrera.

Quiero también agradecer a mi director de tesis, el profesor Wilson Olaya, por su orientación, apoyo, paciencia y motivación continua. Asimismo, agradezco al codirector de este proyecto, el profesor Jorge Eliecer Gomez Rios, por enriquecer este trabajo con su colaboración.

Finalmente, mi gratitud a todas las personas que, a lo largo de mi carrera, me impulsaron a seguir adelante.

Contenido

Índice de cuadros

Índice de figuras

RESUMEN

TÍTULO: CRIPTOGRAFÍA POSCUÁNTICA BASADA EN CÓDIGOS CORRECTORES DE ERRORES*

AUTOR:SANDRA INÉS MANRIQUE GUERRERO**
PALABRAS CLAVE: TEORÍA DE LA CODIFICACIÓN, CRIPTOGRAFÍA, CÓDIGOS DE GOPPA, CRIPTOGRAFÍA POSCUÁNTICA, CRIPTOSISTEMA MCELIECE, CORRECCIÓN DE ERRORES.

DESCRIPCIÓN:

La teoría de la codificación y la criptografía son dos áreas fundamentales para las formas de comunicación modernas. La teoría de la codificación se centra en diseñar sistemas que permitan la transmisión confiable de información a través de canales que puedan estar sujetos a interferencias o ruido. Por otro lado, la criptografía se ocupa de asegurar la confidencialidad e integridad de la información, protegiéndola contra terceros que no tienen acceso autorizado.

Dentro de la teoría de la codificación se destacan los códigos correctores de errores, los cuales permiten detectar y corregir errores que puedan ocurrir durante la transmisión de datos. Dentro de ellos se encuentran los códigos clásicos de Goppa, una familia de códigos lineales introducidos por Valery Denisovich Goppa en 1970[1], que ofrecen una buena alternativa para detectar y corregir errores durante la transmisión de datos, ya que se basan en polinomios sobre un cuerpo finito y aprovechan propiedades algebraicas avanzadas de estos polinomios para generar esquemas de corrección de errores eficientes.

En el área de la criptografía, se destaca el potencial de los códigos de Goppa en sistemas criptográficos como el criptosistema McEliece [2]. En esta tesis abordamos la criptografía poscuántica, que surgió dada la vulnerabilidad de la criptografía de clave pública frente a la computación cuántica. Nos centramos en el criptosistema McEliece, basado en la dificultad del problema de decodificación de códigos lineales aleatorios, como los códigos de Goppa, el cual, en 2022, fue uno de los finalistas del concurso para buscar un estándar en criptografía poscuántica organizado por el NIST (Instituto Nacional de Estándares y Tecnología de EE.UU.) en 2017, ya que pese a los numerosos ataques realizados, ha demostrado ser resistente a ataques utilizando computadoras cuánticas.

*Trabajo de grado

**Facultad de Ciencias. Escuela de Matematicas. Director: Wilson Olaya León, Doctor en ´ Ciencias Matematicas. Codirector: Jorge Eliecer Gómez Ríos, Matemático.

[1]Goppa, V. D. (1970). A new class of linear correcting codes [vid. p. 25]. Problemy Peredachi Informatsii, 6(3), 24-30.

[2]McEliece, R. J. (1978). A Public-Key System Based on Algebraic Coding Theory (inf. téc.) (vid. p. 115). Jet Propulsion Lab.

ABSTRACT

TITLE: POST-QUANTUM CRYPTOGRAPHY BASED ON ERROR-CORRECTING CODES*

AUTHOR: SANDRA INÉS MANRIQUE GUERRERO**

KEYWORDS: CODING THEORY, CRYPTOGRAPHY, GOPPA CODES, POST-QUANTUM CRYPTOGRAPHY, MCELIECE CRYPTOSYSTEM, ERROR CORRECTION.

DESCRIPTION:

Coding theory and cryptography are two fundamental areas for modern communication systems. Coding theory focuses on designing systems that allow reliable information transmission over channels that may be subject to interference or noise. On the other hand, cryptography deals with ensuring the confidentiality and integrity of information, protecting it from unauthorized third-party access.

In coding theory, error-correcting codes stand out as they allow detecting and correcting errors that may occur during data transmission. Among them, Goppa codes, a family of linear codes introduced by Valery Denisovich Goppa in 1970[1], provide an excellent alternative for detecting and correcting errors during data transmission. These codes are based on polynomials over a finite field and leverage advanced algebraic properties of these polynomials to create efficient error correction schemes.

In the field of cryptography, the potential of Goppa codes is prominent in cryptographic systems such as the McEliece cryptosystem[2]. This thesis addresses post-quantum cryptography, which arose from the vulnerability of public-key cryptography to quantum computing. We focus on the McEliece cryptosystem, which is based on the difficulty of decoding random linear codes, such as Goppa codes. In 2022, this cryptosystem was one of the finalists in the NIST (National Institute of Standards and Technology) post-quantum cryptography standardization contest initiated in 2017. Despite numerous attacks, it has demonstrated resistance to quantum computer-based attacks.

*Bachelor Thesis

**Facultad de Ciencias. Escuela de Matematicas. Director: Wilson Olaya León, Doctor en ´ Ciencias Matematicas. Codirector: Jorge Eliecer Gómez Ríos, Matemático.

[1]Goppa, V. D. (1970). A new class of linear correcting codes [vid. p. 25]. Problemy Peredachi Informatsii, 6(3), 24-30.

[2]McEliece, R. J. (1978). A Public-Key System Based on Algebraic Coding Theory (inf. téc.) (vid. p. 115). Jet Propulsion Lab.

Introducción

La teoría algebraica de números es una rama de las matemáticas encargada de estudiar las propiedades de los números, dentro de esta área, a finales del siglo XIX, surgió el estudio de los cuerpos finitos. Dentro de los avances en esta rama destacan importantes contribuciones de los matemáticos Évariste Galois y Richard Dedekind. Sin embargo, solo hasta el siglo XX se empezaron a estudiar realmente los cuerpos finitos.

En 1932, el matemático alemán Ernst Steinitz, creó un conjunto de reglas para los cuerpos finitos, haciendo más organizada la forma de estudiarlos. En la década de 1950, el matemático ruso Andréi Kolmogórov usó el concepto de cuerpos finitos en la teoría de la probabilidad, haciéndola más aplicable.

La teoría de los códigos correctores de errores, que forma parte importante del estudio de los cuerpos finitos, se creó durante la Segunda Guerra Mundial. Claude Shannon y Richard Hamming, junto con otros científicos, desarrollaron algoritmos utilizando cuerpos finitos para garantizar que los datos pudieran enviarse de forma precisa a través de canales de comunicación que tuvieran perturbaciones.

Durante las décadas siguientes, se utilizaron cuerpos finitos en criptografía, al usarse para garantizar la seguridad en las comunicaciones electrónicas y las transacciones financieras. Los cuerpos finitos se utilizan para realizar operaciones aritméticas que hacen que funcionen los algoritmos de cifrado asimétrico, como RSA y ECC (error correcting codes) y también para crear sistemas de detección y corrección de errores, así como para generar números pseudoaleatorios en algoritmos criptográficos.

El inicio histórico de la teoría de códigos correctores de errores se vincula con la necesidad de garantizar una comunicación segura en situaciones donde existen interferencias o perturbaciones en los canales de transmisión. Esta teoría inicia con el artículo clásico Shannon, 1948 "La teoría matemática de la comunicación", escrito por Claude Shannon en 1948 y posteriormente, las matemáticas sobre cuerpos finitos fueron utilizadas para la construcción de dichos códigos correctores de errores. Estos códigos se utilizan ampliamente en aplicaciones, por ejemplo, transmisión de imágenes en el espacio profundo, diseño de números en tarjetas financieras/de

crédito, ISBN de libros, generación de números de teléfono, criptografía y más. Es fascinante observar el importante papel de las matemáticas en la implementación exitosa de estos códigos.

La criptografía es la disciplina encargada de proteger la información para garantizar la confidencialidad de las comunicaciones privadas. Los protocolos de cifrado, descifrado y construcción de claves forman parte de lo que conocemos como criptosistemas. Estos criptosistemas se clasifican en criptografía simétrica y criptografía asimétrica, la primera está compuesta por criptosistemas que utilizan una misma clave secreta para cifrar y descifrar mensajes, mientras la segunda está compuesta por criptosistemas en los cuales, el proceso de cifrado y descifrado se lleva a cabo mediante una clave pública para el proceso de cifrado y una clave privada para descifrar.

A finales de la década de 1970, se propuso el RSA[***] como el primer criptosistema de clave pública Rivest et al., 1978. Actualmente, la seguridad de la mayoría de las comunicaciones utiliza este criptosistema que se basa en la dificultad de resolver dos problemas matemáticos: la factorización y el logaritmo discreto. Sin embargo, en 1994, Peter Shor Shor, 1994 publicó un algoritmo que, si se ejecutara en un ordenador cuántico suficientemente potente, podría resolver eficientemente estos problemas. Este avance marcó el inicio de la vulnerabilidad de la criptografía de clave pública frente a la computación cuántica. En 1978, Robert J. McEliece introdujo la criptografía de clave pública basada en los códigos correctores de errores, particularmente en la familia de códigos de Goppa, ver Goppa, 1970.

Los criptosistemas basados en códigos no ganaron gran popularidad debido a su ineficiencia respecto a su tamaño de clave. Actualmente, están siendo investigados, ya que estos no son afectados directamente por los algoritmos desarrollados por Peter Shor y Lowe Grover. Se ha demostrado que algunos de estos criptosistemas cuentan con una base criptográfica sólida y pueden ser adaptados para conseguir altos estándares de seguridad. Esto ofrece grandes oportunidades para mejoras en lo que respecta a la supercomputación, incluyendo las computadoras cuánticas. Problemas complicados basados en códigos correcctores de errores y decodificación de mensajes con errores aleatorios, son fundamentales en la seguridad

[***]RSA es un algoritmo de cifrado de clave pública ampliamente utilizado en la criptografía moderna, nombrado así por las iniciales de sus inventores: Ron Rivest, Adi Shamir y Leonard Adleman.

del criptosistema McEliece, esta no depende de la dificultad de factorización de números enteros o encontrar el logaritmo discreto de un número como en el caso de los criptosistemas conocidos RSA, ElGamal, etc. La seguridad del criptosistema de McEliece se basa en la capacidad de utilizar códigos ocultos de corrección de errores para recuperar texto sin formato a partir de texto cifrado que el remitente inicialmente pensó que era un error aleatorio. Los computadores cuánticos no parecen mostrar progresos significativos en sistemas de ataque a la criptografía basada en códigos, aparte de las búsquedas mediante ataque exhaustivo. Por lo tanto, el esquema de cifrado McEliece es uno de los candidatos para la criptografía poscuántica.

La criptografía poscuántica, busca sistemas resistentes a ataques tanto de computadores clásicos como cuánticos. En el 2017 el NIST (National Institute of Standards and Technology) organizó un concurso para buscar un estándar en criptografía poscuántica, dentro de las 69 propuestas que se presentaron de todo el mundo se encontró que exploraban diversas áreas como la teoría de códigos, retículos, funciones de hash y álgebra multivariada. En particular, Daniel J. Bernstein et al. presentaron Classic McEliece Bernstein et al., s.f., un criptosistema poscuántico basado en códigos de clave pública (PKC, por las iniciales de public key cryptography), especialmente en el criptosistema propuesto por McEliece en 1978 y sus variaciones, ya que, este esquema de cifrado se ha modificado varias veces debido al gran tamaño de clave y que a pesar de los numerosos ataques documentados a lo largo de 40 años, el nivel de seguridad del criptosistema McEliece se ha mantenido sorprendentemente estable. Logrando en 2022 ser uno de los tres finalistas y, por lo tanto, candidato a la estandarización global por parte del NIST.

En este trabajo estamos interesados en estudiar la criptografía poscuántica basada en la teoría de códigos correctores de errores. Introduciendo las bases matemáticas, la criptografía, la teoría de corrección de errores y la computación necesaria para entenderla. Se detallará el criptosistema de McEliece y su implementación usando el software matemático SageMath, además, la historia de todo el proceso que llevó a que este criptosistema fuera uno de los tres finalistas en la competencia del NIST para la estandarización de la criptografía poscuántica.

1. Preliminares

Los cuerpos finitos son objetos matemáticos básicos que tienen muchos usos en diferentes áreas, dentro de las que se encuentran el estudio de números, el cifrado de mensajes, el diseño de códigos y la computación. Los cuerpos finitos son muy importantes para la teoría de la información y la criptografía, ya que proporcionan una estructura matemática que permite realizar cálculos precisos y eficientes para el diseño de códigos correctores de errores y algoritmos criptográficos, que puedan mejorar la seguridad de las comunicaciones asegurando la confidencialidad y la integridad de la información.

En este capítulo, se presentan algunas propiedades fundamentales de los cuerpos finitos y relaciones con otras estructuras algebraicas útiles para diseñar y analizar sistemas resistentes en la teoría de la información y la seguridad computacional.

1.1. Cuerpos finitos

En criptografía, especialmente para el diseño de algoritmos de cifrado y firmas digitales, se utilizan estructuras matemáticas como cuerpos finitos y anillos, dado que estos son algunas de las estructuras algebraicas fundamentales que nos permiten realizar operaciones aritméticas de manera eficiente y segura en entornos computacionales. A continuación se presentan algunas estructuras y sus propiedades. Como referencia ver Gallian, 2006

Definición 1.1. *Un conjunto no vacío $\mathscr{R}$ se dice que es un* ***anillo*** *si en $\mathscr{R}$ están definidas dos operaciones, denotadas por + (suma) y $\cdot$ (multiplicación) respectivamente, tales que para todos a, b, c en $\mathscr{R}$:*

(i) $(\mathscr{R}, +)$ *es un grupo abeliano;*

(ii) $a \cdot (b \cdot c) = (a \cdot b) \cdot c$*;*

(iii) $a \cdot (b + c) = a \cdot b + a \cdot c$ *y* $(b + c) \cdot a = b \cdot a + c \cdot a$ *(las dos leyes distributivas)*

El axioma (iii) sirve para relacionar las dos operaciones de $\mathscr{R}$. Si la multiplicación de $\mathscr{R}$ es tal que $a \cdot b = b \cdot a$, para cada a, b en $\mathscr{R}$ entonces se dice que $\mathscr{R}$ es un $anillo\ conmutativo$. Además, si existe $1 \in \mathscr{R}$ tal que para cada $a \in \mathscr{R}$, $a \cdot 1 = 1 \cdot a = a$ se dice que $\mathscr{R}$ es un anillo con unidad o unitario.

De lo anterior se tiene la siguiente definición:

Definición 1.2. *Un **cuerpo** $(\mathbb{F}, +, \cdot)$ es un anillo conmutativo con elemento unidad, en el cual cada elemento distinto de cero tiene inverso multiplicativo ($\mathbb{F}$ tiene división).*

Ejemplo 1.3. *Un ejemplo común de cuerpo es $\mathbb{Z}_p$, el conjunto de enteros módulo p, con p un número primo (esto se prueba más adelante).*
Un caso particular es el conjunto de los enteros módulo 3, denotados por $\mathbb{Z}_3$, cuyos elementos son $\{0, 1, 2\}$ y las operaciones suma y multiplicación definidas de la siguiente manera.

$+$	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

$\times$	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

Note que no se puede afirmar en términos generales que para cualquier número natural n distinto de cero, el conjunto $\mathbb{Z}_n$ sea un cuerpo. De hecho, si tomamos $n = 9$, observamos que el número 3 en $\mathbb{Z}_9$ no tiene un inverso multiplicativo, más aún $3 \cdot 3 = 0$ en $\mathbb{Z}_9$.

El conjunto de irracionales no satisface el axioma (i), por lo tanto, no forma un cuerpo, y dado que en el conjunto de enteros $\mathbb{Z}$, los únicos elementos con inverso multiplicativo son -1 y 1, entonces $\mathbb{Z}$ tampoco es un cuerpo.

Definición 1.4. *Un **dominio** $(\mathscr{R}, +, \cdot)$ es un anillo conmutativo con unidad, en el cual no existen divisores de cero. Es decir, si $a, b \in R$ y $a \cdot b = 0$, entonces $a = 0$ o $b = 0$.*

Definición 1.5. *Sea $\mathscr{R}$ un anillo. La **característica** de $\mathscr{R}$ denotada por $car(\mathscr{R})$ es el menor entero positivo n tal que $n \cdot a = 0$, para todo $a \in \mathscr{R}$. En caso de que dicho entero no exista, se dice que la característica es cero.*

Es posible demostrar que si $\mathscr{R}$ es un anillo con unidad, entonces $car(\mathscr{R}) = n$ si y sólo si

$$n \cdot 1 = \underbrace{1 + 1 + 1 + \cdots + 1}_{n \text{ veces}} = 0$$

Además, si $\mathscr{R}$ es anillo conmutativo, con unidad y sin divisores de cero, entonces $car(\mathscr{R}) = 0$ o $car(\mathscr{R}) = p$, con p entero primo.

El teorema a continuación es fundamental, ya que proporciona información valiosa sobre la estructura de un cuerpo y ayuda a la comprensión del desarrollo de las operaciones aritméticas en el mismo.

Teorema 1.6. *La característica de un cuerpo $\mathbb{F}$ es o bien 0 o bien un número primo.*

Demostración. Suponga que $car(\mathscr{R}) \neq 0$, entonces existe p el menor entero positivo tal que $p \cdot a = 0$, para todo $a \in \mathbb{F}$, y supongamos que $p = r \cdot s$, con r y s enteros positivos mayores que 1. Entonces, como $1 \in \mathbb{F}$, tenemos que

$$0 = p \cdot 1 = (r \cdot s) \cdot 1 = (r \cdot 1)(s \cdot 1);$$

y como $\mathbb{F}$ es un dominio entero (no tiene divisores propios de cero), entonces debe ser que $r \cdot 1 = 0$ o $s \cdot 1 = 0$, y por lo tanto $r \cdot a = (r \cdot 1) \cdot a = 0$ o $s \cdot a = (s \cdot 1) \cdot a = 0$, para todo $a \in \mathbb{F}$, lo cual contradice el hecho de que p era el menor. Por lo tanto, p debe ser primo. □

En este trabajo nos centraremos en los cuerpos que son finitos. Como veremos más adelante, los cuerpos finitos tienen una estructura algebraica específica y tienen una cardinalidad que es una potencia de un número primo, es decir, $q = p^n$ con $p, n \in \mathbb{N}$ y p primo. Un cuerpo finito de orden $q = p^m$ se denota por $\mathbb{F}_q$ o $GF(q)$ (por las iniciales de Galois Field).

El siguiente resultado conecta los números primos con la naturaleza de las operaciones en el conjunto $\mathbb{Z}_m$ de enteros con operaciones definidas módulo m, también conocidas como aritmética modular. Este conjunto se comporta de manera interesante, y queremos entender en qué circunstancias se convierte en un cuerpo.

Teorema 1.7. *El conjunto $\mathbb{Z}_m$ de enteros módulo m forma un cuerpo si, y solo si, m es un número primo.*

Demostración. Dado que $\mathbb{Z}_m$ es un anillo, se sigue que $\mathbb{Z}_m$ es un cuerpo si y solo si cada elemento distinto de cero de $\mathbb{Z}_m$ es invertible.

Si m no es primo, entonces $m = ab$, donde $1 < a < m$ y $1 < b < m$. Entonces a no tiene inverso en $\mathbb{Z}_m$, pues si $ax \equiv 1 \mod m$, entonces $abx \equiv b \mod m$. Pero $ab \equiv 0 \mod m$, lo que implica $b \equiv 0 \mod m$, y esto es una contradicción.

Supongamos que m es un primo, y sea $1 \leq a < m$. Entonces $\gcd(a, m) = 1$, así que por el teorema de Bézout $ax + my = 1$, para algunos enteros x e y. Por lo tanto, $ax \equiv 1 \mod m$ y a es invertible módulo m. □

Teorema 1.8. *Todo cuerpo finito $\mathbb{F}$ tiene orden potencia de algún número primo, es decir, $|\mathbb{F}| = q = p^n$, para algún p entero primo y $n \in \mathbb{N}$. Además, dado un primo p y un entero positivo m, existe un único cuerpo (salvo isomorfismo) con p^m elementos.*

Demostración. Ver Gallian, 2006 teorema 22.1. □

Definición 1.9. *Sea $\mathscr{R}$ un anillo. Un subconjunto no vacío $I \subseteq \mathscr{R}$ se dice que es un **ideal** de $\mathscr{R}$ si cumple las siguientes condiciones:*

1. *I es un subanillo de $\mathscr{R}$. Es decir, I es cerrado bajo la suma y la resta: si $a, b \in I$, entonces $a + b \in I$ y $a - b \in I$.*

2. *Para todo $r \in \mathscr{R}$ y todo $a \in I$, tanto $r \cdot a \in I$ como $a \cdot r \in I$. (Esto garantiza que I es cerrado bajo la multiplicación por cualquier elemento de $\mathscr{R}$).*

Definición 1.10. *Sea $\mathscr{R}$ un anillo conmutativo y $P \subseteq \mathscr{R}$ un ideal propio de $\mathscr{R}$. Decimos que P es un **ideal primo** si, siempre que $a, b \in R$ satisfacen $a \cdot b \in P$, entonces $a \in P$ o $b \in P$.*

Definición 1.11. *Sea $\mathscr{R}$ un anillo conmutativo. Un ideal $M \subseteq \mathscr{R}$ se dice que es un **ideal maximal** si M es un ideal propio de $\mathscr{R}$ y no existe otro ideal propio $I \subseteq \mathscr{R}$ tal que $M \subsetneq I \subsetneq \mathscr{R}$.*

1.2. Anillos de polinomios sobre cuerpos finitos

Los anillos de polinomios sobre cuerpos finitos son estructuras algebraicas esenciales en matemáticas y teoría de códigos. Estos anillos permiten realizar operaciones con polinomios cuyos coeficientes son elementos en un cuerpo finito. La aritmética de polinomios en este contexto es muy relevante en aplicaciones criptográficas y códigos correctores de errores.

Definición 1.12. *Sea $\mathbb{F}$ un cuerpo; el conjunto $\mathbb{F}[x]$ está definido como:*

$$\mathbb{F}[x] := \left\{ \sum_{i=0}^{n} a_i x^i : a_i \in \mathbb{F},\ 0 \leq i \leq n,\ y\ n \geq 0 \right\};$$

junto con la adición y multiplicación de polinomios, forma un anillo y se llama el ***anillo de polinomios*** *sobre* $\mathbb{F}$. *Los elementos de* $\mathbb{F}[x]$ *se llaman polinomios sobre* $\mathbb{F}$ *y se denotan por* $f(x)$.

Definición 1.13. *El* ***grado de un polinomio*** $f(x)$ *no nulo, denotado por* $\deg(f(x))$, *es el mayor exponente con coeficiente no nulo en la expresión estándar de* $f(x)$.

Definición 1.14. *Un polinomio* $f(x)$ *de grado positivo se dice que es* ***reducible*** *sobre* $\mathbb{F}$ *si existen dos polinomios* $g(x)$ *y* $h(x)$ *sobre* $\mathbb{F}$ *tales que* $1 \leq \deg(g(x)), \deg(h(x)) < \deg(f(x))$ *y* $f(x) = g(x)h(x)$. *Si no existen tales polinomios en* $\mathbb{F}[x]$, *entonces* $f(x)$ *se dice que es irreducible sobre* $\mathbb{F}$.

Ejemplo 1.15. *El polinomio* $f(x) = x^4 + 2x^6 \in \mathbb{Z}_3[x]$ *tiene grado 6 y es reducible, ya que* $f(x) = x^4(1 + 2x^2)$. *A su vez, el polinomio* $g(x) = 1 + 2x^2 = 1 - x^2 = -(x^2 - 1) = 2(x^2 - 1) = 2(x + 1)(x - 1) = 2(x + 1)(x + 2)$, *por lo tanto, se puede escribir como* $g(x) = 2(x + 1)(x - 1)$.

Ejemplo 1.16. *El polinomio* $g(x) = 1 + x + x^2 \in \mathbb{Z}_2[x]$ *es irreducible, ya que no existe un polinomio con un grado aceptable (como en la definición 1.14 que sea su factor. Esto debido a que los polinomios de grado dos o tres son reducibles si tienen alguna raíz en el cuerpo correspondiente. Ver Gallian, 2006 teorema 17.1. Note que para todo* $a \in \mathbb{Z}_2 = \{0, 1\}$ *se tiene que* $g(a) = a^2 + a + 1 \neq 0$, *es decir,* $g(x)$ *es irreducible en* $\mathbb{Z}_2$.

En los anillos de polinomios tenemos el algoritmo de división, máximo común divisor, mínimo común múltiplo, etc. Dado que para cada $m > 1$ en $\mathbb{Z}$, el anillo $\mathbb{Z}_m = \mathbb{Z}/\langle m \rangle$ está construido, la relación similar se mantiene para los anillos de polinomios también.

Teorema 1.17. *Sea* $\mathscr{R}$ *un anillo y sea* A *un subanillo de* $\mathscr{R}$. *El conjunto de las clases laterales* $\mathscr{R}/A = \{r + A \mid r \in \mathscr{R}\}$ *es un anillo bajo las operaciones* $(s + A) + (t + A) = s + t + A$ *y* $(s + A)(t + A) = st + A$ *si y solo si* A *es un ideal de* $\mathscr{R}$. *El anillo* R/A *se llama anillo cociente.*

Demostración. Ver Gallian, 2006 teorema 14.2 □

Definición 1.18. *Sea* $\mathbb{F}$ *un cuerpo. Un* ***ideal principal*** *en* $\mathbb{F}[x]$ *es un ideal* $\langle f(x) \rangle$ *generado por algún polinomio* $f(x)$. *Es decir,*

$$\langle f(x)\rangle = \{f(x)q(x) : q(x) \in \mathbb{F}[x]\}.$$

Teorema 1.19. *Si $\mathbb{F}$ es un cuerpo, entonces todo ideal en $\mathbb{F}[x]$ es un ideal principal.*

Demostración. Sea I un ideal de $\mathbb{F}[x]$. Si I es el ideal trivial cero, no hay nada que probar. Supongamos que I es un ideal no trivial en $\mathbb{F}[x]$, y sea $f(x) \in I$ un elemento distinto de cero de grado minimal.

Si $\deg(f(x)) = 0$, entonces $f(x)$ es una constante no nula y 1 está en I. Como 1 genera todo $\mathbb{F}[x]$, se tiene que $\langle 1\rangle = I = \mathbb{F}[x]$ y, por lo tanto, I es un ideal principal.

Ahora, supongamos que $\deg(f(x)) \geq 1$ y sea $g(x)$ cualquier elemento en I. Por el algoritmo de la división, existen $q(x)$ y $r(x)$ en $\mathbb{F}[x]$ tales que

$$g(x) = f(x)q(x) + r(x) \quad \text{y} \quad \deg(r(x)) < \deg(f(x)).$$

Dado que $g(x), f(x) \in I$ y I es un ideal, se sigue que $r(x) = g(x) - f(x)q(x)$ también está en I.

Pero, como escogimos $f(x)$ con grado minimal, $r(x)$ debe ser el polinomio cero. Por lo tanto, podemos escribir cualquier elemento $g(x)$ en I como $g(x) = f(x)q(x)$ para algún $q(x) \in \mathbb{F}[x]$. Esto implica que $I = \langle f(x)\rangle$.

Por lo tanto, I es un ideal principal. □

Teorema 1.20. *Sea $f(x)$ un polinomio de grado n en $\mathbb{F}[x]$, entonces el conjunto*

$$\frac{\mathbb{F}[x]}{\langle f(x)\rangle} = \{p(x) + \langle f(x)\rangle : p(x) \in \mathbb{F}[x]\}.$$

es un anillo cociente.

Demostración. Dado que $\mathbb{F}$ es un cuerpo, entonces $\langle f(x)\rangle$ es un ideal de $\mathbb{F}[x]$, por tanto, $\frac{\mathbb{F}[x]}{\langle f(x)\rangle}$ es un anillo cociente. □

El próximo resultado conecta los polinomios irreducibles con la estructura cociente resultante. Consideremos un polinomio $f(x)$ con coeficientes en un cuerpo $\mathbb{F}$, donde el grado de $f(x)$ es mayor o igual a 1. La pregunta que nos planteamos es: ¿cuándo el anillo $(\mathbb{F}[x]/\langle f(x)\rangle\ )$ se convierte en un cuerpo? El resultado siguiente responde a este interrogante.

Teorema 1.21. *Sea $f(x)$ un polinomio sobre un cuerpo $\mathbb{F}$ con grado ≥ 1. Entonces, el anillo $\mathbb{F}[x]/\langle f(x)\rangle$ es un cuerpo si y solo si $f(x)$ es un polinomio irreducible sobre $\mathbb{F}$.*

Demostración. Supongamos que $F[x]/\langle f(x)\rangle$ es un cuerpo. Entonces $\langle f(x)\rangle$ es un ideal maximal. Los ideales maximales son ideales primos, por lo que $f(x)$ es primo. En un DIP (Dominio de Ideales Principales), como $F[x]$, los elementos primos son irreducibles, por lo que $f(x)$ es irreducible.

Otra opción es considerar que $f(x)$ es reducible. Entonces existen elementos no unidades $q(x)$ y $s(x)$ tales que $f(x) = q(x)s(x)$. Se sigue que $\langle f(x)\rangle \subset \langle q(x)\rangle$, lo que proporciona una prueba alternativa.

Recóprocamente si $f(x)$ es irreducible. Entonces es un elemento primo en $F[x]$. Como $F[x]$ es un DIP, los ideales primos son maximales. Dado que el cociente de un anillo conmutativo con identidad por un ideal maximal es un cuerpo, esto completa la prueba. □

Ejemplo 1.22. *El anillo $\mathbb{Z}_2[x]/\langle 1+x+x^2\rangle = \{p(x) + \langle 1+x+x^2\rangle : p(x) \in \mathbb{Z}_2[x]\}$.*
Note que $p(x) \in \mathbb{Z}_2[x]$ se puede escribir como $p(x) = (x^2+x+1)q(x) + r(x)$ donde $r(x) = 0$ o $0 \leq deg(r(x)) \leq 1$, así $r(x) = a_1x + a_0$ con $a_1, a_0 \in \mathbb{Z}_2$, luego
$\{p(x) + \langle 1+x+x^2\rangle : p(x) \in \mathbb{Z}_2[x]\} = \{a_0 + a_1x + \langle 1+x+x^2\rangle : a_0, a_1 \in \mathbb{Z}_2[x]\}$
Como $\overline{1+x+x^2}+\langle 1+x+x^2\rangle = \overline{0}+\langle 1+x+x^2\rangle$ entonces $\overline{x^2}+\langle 1+x+x^2\rangle = \overline{1+x}+\langle 1+x+x^2\rangle$,
Así $\mathbb{Z}_2[x]/\langle 1+x+x^2\rangle = \{\overline{0}, \overline{1}, \overline{x}, \overline{1+x}\}$ es un cuerpo de orden $2^2 = 4$.

Teorema 1.23. *Sea $f(x)$ un polinomio irreducible con grado m en $\mathbb{F}_p[x]$, entonces $\frac{\mathbb{F}_p[x]}{\langle f(x)\rangle}$ es un cuerpo finito con p^m elementos.*

Demostración. Sea $f(x) \in \mathbb{F}_p[x]$ irreducible de grado m. Entonces dado $g(x) \in \mathbb{F}_p[x]$, $g(x) = f(x)q(x) + r(x)$ con $0 \leq deg(r(x)) \leq m$, $r(x) = b_0 + b_1x + \cdots + b_{m-1}x^{m-1}$, como cada

b_i tiene p posibilidades sigue que $\left|\frac{\mathbb{F}_p[x]}{\langle f(x)\rangle}\right| = p^m$. Esto muestra que este es un cuerpo con p^m elementos. □

Un cuerpo finito de orden q, denotado por $\mathbb{F}_q$, donde $q = p^n$ para algún primo p y un número natural n, se describe como:

$$\mathbb{F}_q = \mathbb{Z}_p[x]/\langle f(x)\rangle = \left\{ \sum_{i=0}^{n-1} a_i x^i + \langle f(x)\rangle : 0 \leq i \leq n-1, a_i \in \mathbb{Z}_p \right\}.$$

donde $\mathbb{Z}_p[x]$ es el anillo de polinomios en la variable x con coeficientes de $\mathbb{Z}_p$, y $f(x)$ es un polinomio irreducible de grado n sobre $\mathbb{Z}_p$.

Definición 1.24. *Un elemento α en un cuerpo finito $\mathbb{F}_q$ es llamado **elemento primitivo** de $\mathbb{F}_q$ si $\mathbb{F}_q = \{0, \alpha, \alpha^2, \ldots, \alpha^{q-1}\}$, es decir, α es generador de $\mathbb{F}_q^*$, esto es, $\mathbb{F}_q^* = \mathbb{F}_q - \{0\} = \langle\alpha\rangle = \{a, a^2, \ldots, a^{q-1}\}$.*

Ejemplo 1.25. *Consideremos el cuerpo $\mathbb{F}_4 = \mathbb{Z}_2(\alpha)$, donde α es una raíz del polinomio irreducible $1 + x + x^2 \in \mathbb{Z}_2[x]$. Entonces tenemos:*

$\alpha^2 = -(1+\alpha) = 1 + \alpha, \quad \alpha^3 = \alpha(\alpha^2) = \alpha(1+\alpha) = \alpha + \alpha^2 = \alpha + 1 + \alpha = 1.$

Al realizar una tabla con las operaciones, vemos que $\mathbb{F}_4 = \{0, \alpha, 1+\alpha, 1\} = \{0, \alpha, \alpha^2, \alpha^3\}$, por lo que α es un elemento primitivo de $\mathbb{F}_4$.

Cuadro 1: Suma y producto en $\mathbb{F}_4$

$+$	0	α	$1+\alpha$	1
0	0	α	$1+\alpha$	1
α	α	0	1	$1+\alpha$
$1+\alpha$	$1+\alpha$	1	0	α
1	1	$1+\alpha$	α	0

$\cdot$	0	α	$1+\alpha$	1
0	0	0	0	0
α	0	$1+\alpha$	1	α
$1+\alpha$	0	1	α	$1+\alpha$
1	0	α	$1+\alpha$	1

El lema que se presenta a continuación muestra algunas propiedades importantes sobre el orden de los elementos y su relación con la cardinalidad del cuerpo. Este lema establece dos

ideas relevantes: primero, que el orden multiplicativo de cualquier elemento no nulo en el cuerpo es un divisor de $q-1$, y segundo, que para dos elementos no nulos cuyos órdenes son coprimos, el orden del producto de estos elementos es el producto de sus órdenes respectivos. Estas propiedades son fundamentales en numerosas aplicaciones matemáticas y criptográficas.

Lema 1.26. *El orden multiplicativo de cualquier elemento no nulo $\alpha \in \mathbb{F}_q$ divide a $q-1$. Además, para cualquier par de elementos no nulos $\alpha, \beta \in \mathbb{F}_q$, tales que mcd$(|\alpha|, |\beta|) = 1$, entonces $|\alpha\beta| = |\alpha||\beta|$.*

Demostración. Ver Gallian, 2006 teorema 7.1, corolario 2. □

Teorema 1.27. *Sea $c \in \mathbb{F}_{2^m}$ entonces existe un a tal que $a \cdot a = c$.*

Demostración. Como $\mathbb{F}_{2^m}$ es un cuerpo, entonces para cualquier elemento c en $\mathbb{F}_{2^m}$, existe un elemento d tal que $d \cdot c = 1$.

Además, como $\mathbb{F}_{2^m}$ es un cuerpo finito, entonces cada elemento d tiene un orden finito, es decir existe un entero positivo t tal que $d^t = 1$. Por el lema 1.26, sabemos que t divide a $2^m - 1$. Es decir, existe un entero k tal que $2^m - 1 = t \cdot k$.

Note que $2^m - 1$ es un número impar y, por lo tanto, t también debe ser impar. Esto implica que podemos expresar t de la forma $t = 2 \cdot s + 1$ para algún entero s.

Ahora, consideramos la expresión de d^t:

$$d^t = d^{2s+1}$$

Dado que $d^t = 1$, podemos escribir:

$$d^{2s+1} = 1$$

Multiplicamos ambos lados de la ecuación por c:

$$d^{2s+1} \cdot c = 1 \cdot c$$

$$d^{2s} \cdot d \cdot c = c$$

Dado que $d \cdot c = 1$, sustituimos:

$$d^{2s} \cdot 1 = c$$

$$d^{2s} = c$$

Finalmente, hemos demostrado que:

$$d^{2s} = c$$

□

Teorema 1.28. *Un elemento no nulo de $\mathbb{F}_q$ es un elemento primitivo si y solo si su orden multiplicativo es $q-1$; además, todo cuerpo finito tiene al menos un elemento primitivo.*

Demostración. Supongamos que α es un elemento primitivo de $\mathbb{F}_q$, lo cual implica que el conjunto $\{\alpha^0, \alpha^1, \alpha^2, \ldots, \alpha^{q-2}\}$ contiene todos los elementos no nulos de $\mathbb{F}_q$. Se sabe que el orden multiplicativo de α es el entero más pequeño k tal que $\alpha^k = 1$. Dado que α es primitivo, el orden multiplicativo de α debe ser $q-1$, si esto no ocurriera, α no generaría todos los elementos no nulos de $\mathbb{F}_q$.

Recíprocamente, consideremos que el orden multiplicativo de α es $q-1$, es decir, que $\alpha^{q-1} = 1$, pero $\alpha^k \neq 1$ para todo $k < q-1$. Dado que un cuerpo finito $\mathbb{F}_q$ tiene $q-1$ elementos no nulos, se sigue que α genera todos los elementos no nulos de $\mathbb{F}_q$, esto es $\mathbb{F}_q - \{0\} = \{\alpha^1, \alpha^2, \ldots, \alpha^{q-1}\}$, por lo tanto, α es elemento primitivo. □

En general, los elementos primitivos no son únicos para ningún cuerpo. Un polinomio minimal de un elemento $\alpha \in \mathbb{F}_{q^m}$ con respecto a $\mathbb{F}_q$ es el único polinomio mónico no nulo $f(x)$ de menor grado en $\mathbb{F}_q[x]$ tal que $f(\alpha) = 0$.

El polinomio minimal siempre es irreducible sobre el cuerpo base; y las raíces de este polinomio son todas elementos primitivos de $\mathbb{F}_{q^m}$.

Teorema 1.29. *Sea n un número natural, $\mathbb{F}_q^n$ define un conjunto $\{(a_1, a_2, ..., a_n) : a_i \in \mathbb{F}_q$ para cada $1 \leq i \leq n\}$. La suma en $\mathbb{F}_q^n$ se realiza coordenada por coordenada. Este conjunto forma un espacio vectorial n—dimensional sobre $\mathbb{F}_q$.*

1.3. Cuerpos finitos con Sagemath

A lo largo de este trabajo nos apoyaremos en el uso de un lenguaje de programación llamado Sagemath o Sage, para llevar a cabo cálculos necesarios para presentar ejemplos, facilitando la comprensión de las partes teóricas.

Usando Sage, podemos crear un cuerpo finito con el siguiente comando:

```
In: GF(order, name, modulus, impl)
```

donde el único parámetro obligatorio es el orden o cardinalidad del cuerpo finito. El parámetro `name` es una cadena que nos permite darle un nombre a la variable de los polinomios que forman parte del cuerpo. El parámetro `modulus` nos permite especificar el polinomio $p(x)$ con el que tomaremos cociente. Si no se especifica, se toma por defecto el polinomio de Conway. Por último, `impl` nos permite especificar si deseamos implementar los elementos de una forma distinta a la explicada anteriormente.

Además, Sage también nos permite obtener la representación de un cuerpo G como espacio vectorial sobre otro cuerpo F de la siguiente forma:

```
In: G.vector_space(F, map=true)
```

Este método retorna tres valores. En primer lugar, el espacio vectorial V sobre F al que G es isomorfo; en segundo lugar, el isomorfismo desde G a V; y en tercer lugar, su aplicación inversa. Gracias a esta forma, podremos alternar entre las diferentes implementaciones de G. Para usar las aplicaciones devueltas por la función, lo haremos usando paréntesis:

```
In: funcion(x)
```

donde x es el elemento del que queremos obtener la imagen por la aplicación funcion.

En Sage también se tienen funciones como:

- `G.order()`: Esta función proporciona el orden del grupo G, es decir, el número de elementos en el grupo.
- `G.characteristic()`: Esta función proporciona la característica del grupo.

- `a.multiplicative_order()`: proporciona el orden de un elemento.
- `G.random_element()`: proporciona un elemento aleatorio del grupo G.
- `G.is_prime_field()`: verifica si el orden de un cuerpo finito es un número primo.

  Además, para hallar un elemento primitivo o generador del cuerpo podemos usar las funciones:

- `G.gen()`
- `G.multiplicative_generator()`
- `G.primitive_element()`

Veamos un ejemplo en $\mathbb{F}_{25}$:

```
[3]: G1 = GF(25,x)
     G1
```

```
[3]: Finite Field in x of size 5^2
```

```
[4]: G1.is_field()
```

```
[4]: True
```

```
[5]: G1.order()
```

```
[5]: 25
```

```
[6]: G1.characteristic()
```

```
[6]: 5
```

```
[7]: F=GF(5)
```

```
[8]:  G1.vector_space(F, map=true)
```

```
[8]: (Vector space of dimension 2 over Finite Field of size 5,
      Isomorphism:
        From: Vector space of dimension 2 over Finite Field of␣
      ↪size 5
        To:   Finite Field in x of size 5^2,
      Isomorphism:
        From: Finite Field in x of size 5^2
        To:   Vector space of dimension 2 over Finite Field of␣
      ↪size 5)
```

```
[9]: G1.random_element()
```

```
[9]: 3*x + 2
```

```
[10]: K.<a> = G1
```

```
[11]: a.multiplicative_order()
```

```
[11]: 24
```

```
[12]: a = G1.gen()
```

```
[13]: print(a**4)
```

```
      2*x + 2
```

```
[14]: G1.is_prime_field()
```

```
[14]: False
```

```
[15]: F.is_prime_field()
```

```
[15]: True
```

```
[16]: GF(25).multiplicative_generator()
```

```
[16]: z2
```

```
[17]: GF(25).primitive_element()
```

```
[17]: z2
```

Para construir anillos de polinomios en Sage podemos usar varias formas, una de estas es

```
In: PolynomialRing(base_ring, t)
```

de esta forma creamos un anillo de polinomios en una variable, y pedimos que esta variable se muestre por pantalla como t. Sin embargo, de esta forma no se define t como variable simbólica en Sage, y no se puede usar este símbolo para escribir polinomios del anillo. Otra forma es

```
In: R.<t>=PolynomialRing(base_ring, t)
```

donde el operador $< t >$ asigna la variable t al anillo polinómico y se utiliza la notación de puntos para asignar la variable para asegurar que t sea tratado como un generador del anillo polinómico.

Por otra parte, la función `ideal()` se utiliza para crear un ideal en un anillo o cuerpo.

```
In: R.ideal(generators)
```

donde R es un anillo o cuerpo, y `generators` es una lista de generadores del ideal. Y la función `quotient()` se utiliza para crear un anillo cociente a partir de un anillo y un ideal.

```
In: R.quotient(I)
```

donde R es un anillo e I es un ideal en ese anillo.
Veamos un ejemplo:

```
[1]: R = PolynomialRing(QQ, 't')
     R
```

```
[1]: Univariate Polynomial Ring in t over Rational Field
```

[2]:
```
S= PolynomialRing(GF(2), 't')
S
```

[2]:
```
Univariate Polynomial Ring in t over Finite Field of size 2␣
↪(using GF2X)
```

[3]:
```
R.<t> = PolynomialRing(GF(25))
```

[4]:
```
R = PolynomialRing(GF(25), 't')
t = R.0
t in R
```

[4]:
```
True
```

[5]:
```
I1 = GF(25).ideal(1)
```

[6]:
```
I1
```

[6]:
```
Principal ideal (1) of Finite Field in z2 of size 5^2
```

[8]:
```
Q1 = GF(25).quotient(I1); Q1
```

[8]:
```
Quotient of Finite Field in z2 of size 5^2 by the ideal (1)
```

[9]:
```
M = R.ideal(t^2+4)
Q = R.quotient(M)
Q
```

[9]:
```
Univariate Quotient Polynomial Ring in tbar over Finite Field␣
↪in z2 of size 5^2
with modulus t^2 + 4
```

[10]:
```
Q.random_element()
```

[10]:
```
(2*z2 + 1)*tbar + 2*z2 + 1
```

[12]:
```
Q.<t> = R.quotient(M); Q
```

```
[12]: Univariate Quotient Polynomial Ring in t over Finite Field in␣
      ↪z2 of size 5^2
      with modulus t^2 + 4
```

Además existen en SageMath otras funciones que nos permiten realizar operaciones con los polinomios, de la siguiente manera:

- La comprobación de que un polinomio es irreducible (sobre el cuerpo de definición) se realiza con la función:
  `f.is_irreducible()`
- Para factorizar un polinomio (sobre el cuerpo de definición), se utiliza el método:
  `f.factor()`
- El máximo común divisor de dos polinomios, que es independiente del cuerpo base, se obtiene con la función:
  `f.gcd(g)`
- Para calcular el mínimo común múltiplo, se emplea:
  `f.lcm(g)`
- Si queremos hallar las raíces (con su multiplicidad) que existen en el cuerpo base, utilizamos el método:
  `f.roots()`
- El grado de un polinomio se obtiene utilizando:
  `f.degree()`
- Para obtener una lista con todos los coeficientes, ordenados de grado menor a grado mayor, se utiliza el método:
  `f.coefficients()`
- Para calcular el polinomio minimal se utiliza la función:
  `p.minpoly()`

Veamos el siguiente ejemplo:

```
[3]: R.<t> = PolynomialRing(GF(25))

[4]: f=R.random_element()
     print(f)

     (z2 + 1)*t^2 + (4*z2 + 4)*t + 4*z2 + 1

[5]: f.factor()

[5]: (z2 + 1) * (t + z2 + 3) * (t + 4*z2 + 1)

[6]: g=R.random_element()
     print(g)

     t + 4*z2 + 2

[7]: f.gcd(g)

[7]: 1

[8]: _.factor()

[8]: 1

[9]: f.lcm(g)

[9]: t^3 + (4*z2 + 1)*t^2 + (3*z2 + 3)*t + 2*z2 + 4

[10]: _.factor()

[10]: (t + z2 + 3) * (t + 4*z2 + 1) * (t + 4*z2 + 2)

[11]: (t^2 - 2).is_irreducible()

[11]: False
```

```
[12]: f.roots()

[12]: [(4*z2 + 2, 1), (z2 + 4, 1)]

[13]: f.degree()

[13]: 2

[14]: f.coefficients()

[14]: [4*z2 + 1, 4*z2 + 4, z2 + 1]

[17]: F.<j, a> = QQ[sqrt(-1), sqrt(7)]
      RF.<X> = PolynomialRing(F)

[18]: p=1/2 * ( exp(2*pi*i /7) + exp(-2*pi*i /7) )

[19]: p.minpoly()

[19]: x^3 + 1/2*x^2 - 1/2*x - 1/8
```

2. Teoría de códigos

La teoría de códigos es una rama de las matemáticas que trata de las normas para codificar información. A grandes rasgos, codificar es transformar una información en una señal conveniada para su comunicación, mientras que decodificar sería el proceso inverso. En esta rama son fundamentales la detección y corrección de errores para evitar la pérdida de información. En este capítulo abordaremos los códigos correctores de errores, los cuales permiten mitigar los efectos de diferentes tipos de fallos que se pueden producir en la transmisión de información, y además estudiaremos los procesos de codificación y decodificación para estos códigos.

2.1. Códigos correctores de errores

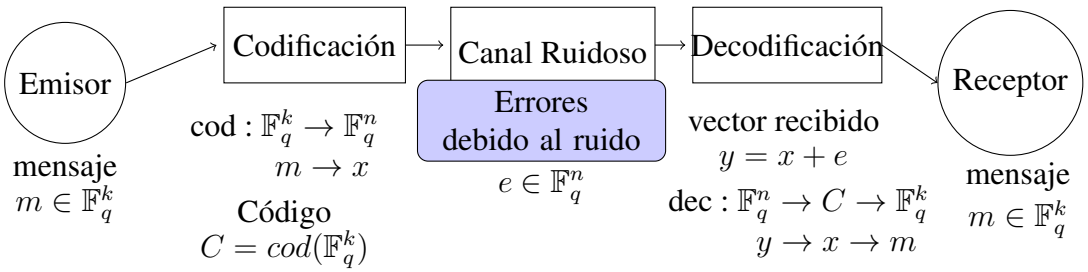
En teoría de la codificación, un código se debe entender como una combinación de signos (números, letras, etc.) que tiene un cierto valor en un sistema o que hace posible reformular y

comprender un mensaje secreto. Codificar caracteres consiste en transformar un carácter de un sistema en un símbolo perteneciente a otro sistema de representación.

Los códigos de corrección de errores permiten que al enviar información codificada a través de un canal sea posible, además de detectar un error, corregirlo sin necesidad de repetir la transmisión. De forma general, se puede decir que la técnica más empleada para la corrección consiste en identificar, como combinación correcta, a la palabra código que sea más parecida a la errónea recibida.

Los códigos correctores de errores son comunes en numerosas aplicaciones modernas donde es necesario almacenar, procesar y transmitir datos de manera confiable, como en medios audiovisuales, códigos de respuesta rápida (QR), sistemas informáticos tolerantes a fallas y telecomunicaciones en el espacio profundo, como por ejemplo el envío de imágenes desde Júpiter y Saturno a la Tierra con la nave espacial Voyager, entre otras. El objetivo de la teoría de la codificación es transmitir mensajes a través de canales ruidosos, donde el término ruido se refiere a cualquier factor que provoque alteraciones en el mensaje. La transmisión de mensajes se visualiza en el siguiente diagrama:

Figura 1: Esquema de transmisión de información



En el anterior esquema, observamos que cuando el mensaje m es transmitido y ocurre un error e durante la transmisión, el receptor recibe y, donde y se obtiene sumando el mensaje codificado x con el error e, lo que se expresa como $y = x + e$, donde $+$ es la suma usual de vectores en $\mathbb{F}_q^n$.

Los códigos correctores de errores buscan garantizar una comunicación fiable en situaciones donde existen interferencias o perturbaciones en los canales de transmisión. Otro propósito de la

teoría de la codificación diferente de la corrección de errores es asegurar la privacidad y la integridad de la información al enviar un mensaje, mediante técnicas de cifrado, un proceso a través del cual se transforma la información a fin de que solo aquellos autorizados puedan acceder a ella, lo cual se denomina criptografía y lo trataremos más adelante. El problema fundamental en la teoría de la codificación es entonces determinar qué mensaje se envió en función de lo que se recibe, es decir, la detección y corrección de errores al enviar información. Además, la teoría de la codificación es fundamental en la compresión de datos, donde se buscan métodos para reducir la cantidad de información necesaria para representar un conjunto de datos sin perder información importante. En este contexto, se definen varios elementos fundamentales:

Definición 2.1. *Un **código de bloque** C de longitud n es un subespacio de $\mathbb{F}_q^n$. También se acostumbra decir que C es un código de bloque q-ario de longitud n. Los elementos de C son llamados palabras código.*

Definición 2.2. *El **alfabeto** A es el conjunto de posibles elementos utilizados para construir los mensajes a enviar. En particular, en este trabajo asumiremos $A = \mathbb{F}_q^n$.*

Ejemplo 2.3. *Algunos ejemplos de códigos de bloque son:*

- *Un código de bloque de longitud n en el cual cada palabra de código es repetición de un solo símbolo se llama un código de repetición de longitud n. Así, el código ternario de repetición de longitud 5 es $C = \{(0,0,0,0,0), (1,1,1,1,1), (2,2,2,2,2)\}$*

- *Un código de bloque de longitud n en el cual se añade un dígito adicional (paridad) con el fin de asegurar la detección y corrección de errores se llama código de paridad de longitud n. Así, el código de paridad q-ario de longitud n es el conjunto*
  $C = \{(c_1, c_2, \cdots, c_n) : c_i \in \mathbb{F}_q \text{ y } \sum_{i=1}^{n} c_i = 0\}$

Definición 2.4. *La **distancia de Hamming** entre dos vectores de $\mathbb{F}_q^n$, denotada por $d(x, y)$, donde $x = (x_1, x_2, \ldots, x_n)$ y $y = (y_1, y_2, \ldots, y_n)$ se define como $d_H(x, y) = |\{i : x_i \neq y_i\}|$. La distancia mínima de un código C se define como*

$$d(C) := \min\{d(x, y) \mid \forall x, y \in C, x \neq y\}.$$

Definición 2.5. *El **peso de Hamming** $wt(x)$ de $x \in \mathbb{F}_q^n$ es el número de coordenadas no nulas en x. Es decir, para todo $x \in \mathbb{F}_q^n$, $wt(x) = d(x, 0) = |\{i : x_i \neq 0\}|$.*

A continuación se establecen las propiedades fundamentales de la distancia de Hamming como una métrica en el espacio $\mathbb{F}_q^n$ y se proporcionan relaciones específicas para el caso de $\mathbb{F}_2^n$.

Teorema 2.6. *La distancia de Hamming es una métrica en $\mathbb{F}_q^n$. Esto es, cumple las siguientes propiedades. Para todo $x, y, z \in \mathbb{F}_q^n$:*

1. $d(x, y) \geq 0$.
2. $d(x, y) = 0$ *si y solo si* $x = y$.
3. $d(x, y) = d(y, x)$.
4. $d(x, y) \leq d(x, z) + d(z, y)$.

Demostración. Los ítems 1., 2. y 3. se siguen de la definición de distancia de Hamming. Para la desigualdad triangular sean $x = (x_1, \ldots, x_n)$, $y = (y_1, \ldots, y_n)$ y $z = (z_1, \ldots, z_n) \in \mathbb{F}_q^n$. Considere los siguientes conjuntos,

$$A = \{i \,|\, x_i = y_i\}, \quad B = \{i \,|\, x_i = z_i\}, \quad C = \{i \,|\, z_i = y_i\},$$

Note que $B \cap C \subseteq A$, entonces $|B \cap C| \leq |A|$. Por otro lado, como $|B \cap C| = |B| + |C| - |B \cup C|$ y $|B \cup C| \leq n$, se tiene que,
$d(x, y) = n - |A| \leq n - |B| - |C| + |B \cup C| \leq n - |B| - |C| + n = d(x, z) + d(z, y)$. $\square$

Algunas caracterizaciones para el peso de Hamming y para el caso de $\mathbb{F}_2^n$.

- Si $x, y \in \mathbb{F}_q^n$, entonces $d(x, y) = wt(x - y)$. Como consecuencia de esto, si x y y son vectores en $\mathbb{F}_2^n$, $d_(x, y) = wt(x + y)$.
- Si q es par y $x, y \in \mathbb{F}_q^n$, entonces $d(x, y) = wt(x + y)$, esto porque el inverso aditivo de cada elemento en el cuerpo, es el mismo elemento.
- Si $x, y \in \mathbb{F}_2^n$, entonces $wt(x + y) = wt(x) + wt(y) - 2 \cdot wt(x \star y)$, donde $\star$ representa multiplicación componente a componente, esto se debe a que al sumar cada peso por separado estamos contando también las coordenadas donde $x_i = y_i = 1$.

Definición 2.7. *La **codificación** se define como la función*
$cod : \{$*Conjunto de Mensajes*$\} \to C \subseteq \mathbb{F}_q^n$*, es decir, para codificar un mensaje* $m \in \mathbb{F}_q^k$*, se calcula la palabra código* $c = cod(m)$.

No se tiene una forma común de codificar en general. Para ilustrar, consideremos dos mensajes simples: "Sí" y "No". Podríamos asignar la codificación "Sí" $\to$ 1 y "No" $\to$ 0. Sin embargo, si durante la transmisión, debido al ruido en el canal, el receptor recibe 0 en lugar de 1 (o viceversa), el receptor no tendría forma de saber cuál era el mensaje original.

Para abordar este problema, se puede utilizar una técnica de codificación basada en la decodificación por bits mayoritarios. Por ejemplo, en $\mathbb{F}_2^3$ podríamos asignar la codificación "Sí" $\to$ 111 y "No" $\to$ 000. En caso de que ocurra un error en la transmisión, incluso si un solo bit se altera en el canal, el receptor aún podría detectar y corregir fácilmente el error. Este caso específico utiliza el código binario de repetición de longitud 3, donde los bits mayoritarios determinan el mensaje decodificado.

Este enfoque plantea el problema central en la Teoría de Codificación: cómo detectar y corregir errores en la transmisión de datos.

El siguiente teorema aborda la capacidad de un código C para detectar y corregir errores en sus palabras código. Específicamente, el teorema establece que un código C puede detectar hasta s errores en cualquier palabra código si la distancia mínima $d(C)$ es mayor o igual que $s + 1$. Del mismo modo, el código C puede corregir hasta t errores en cualquier palabra código si la distancia mínima $d(C)$ es mayor o igual que $2t + 1$. Estas propiedades son esenciales en la teoría de códigos, ya que permiten evaluar la capacidad de un código para detectar y corregir errores durante la transmisión de información. Cuanto mayor sea la distancia mínima, mayor será su capacidad de detección y corrección de errores.

Teorema 2.8. *Un código* C *con distancia mínima* d *puede:*

1. *Detectar hasta* s *errores en cualquier palabra código si* $d(C) \geq s + 1$.

2. *Corregir hasta* t *errores en cualquier palabra código si* $d(C) \geq 2t + 1$.

Demostración. 1. Supongamos que C no puede detectar s errores en cualquier palabra código. Esto significa que existe una palabra código c_1 en C y una palabra código c_2 tales que $d(c_1, c_2) \leq s$. Sin embargo, esto contradice la hipótesis $d(C) \geq s + 1$.

2. Supongamos que C no puede corregir t errores en cualquier palabra código c. Esto significa que existe una palabra código c_1 en C y una palabra código c_2 tales que $d(c, c_1) \leq t$ y $d(c, c_2) \leq t$, por tanto, $d(c_1, c_2) \leq 2t$. Sin embargo, esto contradice la hipótesis de que $d(C) \geq 2t + 1$. □

El resultado a continuación establece que para cualquier vector u en un espacio vectorial $\mathbb{F}_q^n$ y un número entero r en el rango de 0 a n, la esfera de radio r, centrada en $u \in \mathbb{F}_q^n$

$$S(u, r) := \{v \in \mathbb{F}_q^n : d(u, v) \leq r\}$$

contiene exactamente $\sum_{i=0}^{r} \binom{n}{i}(q-1)^i$ elementos en $\mathbb{F}_q^n$. En otras palabras, esta esfera consiste en todos los vectores que pueden obtenerse al modificar hasta r componentes del vector u en $\mathbb{F}_q^n$. Este resultado proporciona una forma de contar el número de palabras código posibles alrededor de un punto dado en el espacio $\mathbb{F}_q^n$.

Teorema 2.9. *Para algún $u \in \mathbb{F}_q^n$ y $r \in \{0, 1, \ldots, n\}$, $S(u, r)$ contiene exactamente $\binom{n}{0} + \binom{n}{1}(q-1) + \binom{n}{2}(q-1)^2 + \ldots + \binom{n}{r}(q-1)^r$ elementos de $\mathbb{F}_q^n$.*

Demostración. Para cada i en el rango de 0 a r, hay $\binom{n}{i}$ maneras de elegir i componentes del vector u que se modificarán. Para cada componente elegida, hay $q - 1$ opciones posibles para su modificación, ya que estamos en un cuerpo finito $\mathbb{F}_q$ con q elementos en total, y excluimos el valor original.

Entonces, el número total de vectores que podemos obtener al modificar exactamente i componentes es $\binom{n}{i}(q-1)^i$. Sumando este valor para todas las posibles i de 0 a r, obtenemos

$$|S(u, r)| = \sum_{i=0}^{r} \binom{n}{i}(q-1)^i.$$

Por lo tanto, hemos demostrado que el conjunto $S(u, r)$ contiene exactamente $\sum_{i=0}^{r} \binom{n}{i}(q-$

$1)^i$ elementos en $\mathbb{F}_q^n$ cuando u está en $\mathbb{F}_q^n$ y r está en el rango de 0 a n. □

Teorema 2.10 (Empaquetado de esferas o límite de Hamming). *Sea M es el número de palabras en un código C de longitud n sobre $\mathbb{F}_q$ con $d(C) \geq 2t+1$, entonces tenemos:*

$$M\left(\binom{n}{0} + \binom{n}{1}(q-1) + \binom{n}{2}(q-1)^2 + \ldots + \binom{n}{t}(q-1)^t\right) \leq q^n.$$

Demostración. Cada palabra en el código C tiene una esfera alrededor de ella en el espacio $\mathbb{F}_q^n$ con radio t (ya que el código puede corregir hasta t errores). La cantidad de palabras distintas que podemos formar con t errores o menos es exactamente

$$\binom{n}{0} + \binom{n}{1}(q-1) + \binom{n}{2}(q-1)^2 + \ldots + \binom{n}{t}(q-1)^t.$$

Esto se debe a las posibles combinaciones de errores que se pueden introducir en una palabra de código de longitud n.

Ahora, dado que cada esfera tiene como máximo M palabras y estas esferas son disjuntas, es decir, dos esferas no pueden contener el mismo punto en $\mathbb{F}_q^n$, el número total de puntos en el espacio $\mathbb{F}_q^n$ que estas esferas pueden cubrir está limitado por q^n.

Por lo tanto, tenemos la desigualdad:

$$M\left(\binom{n}{0} + \binom{n}{1}(q-1) + \binom{n}{2}(q-1)^2 + \ldots + \binom{n}{t}(q-1)^t\right) \leq q^n.$$

□

Ejemplo 2.11. *En el código de repetición binario de longitud 5,*

$$C = \{(0,0,0,0,0), (1,1,1,1,1)\}$$

toda bola con centro en una palabra código y radio 2 tiene cardinal $|S(a,2)| = \binom{5}{0} + \binom{5}{1}(2-1) + \binom{5}{2}(2-1)^2 = 1+5+10 = 16$*, es decir, $S(a,2)$ tiene 16 palabras de $\mathbb{F}_2^5$ y por el empaquetamiento de esferas se tiene que $2 \cdot 16 \leq 2^5 = 32$ y, por lo tanto, las dos esferas cubren todo el espacio $\mathbb{F}_2^5$.*

Definición 2.12. *Un código C de longitud n sobre $\mathbb{F}_q$ es llamado* ***perfecto*** *si satisface la igualdad en el límite de empaquetado de esferas.*

Ejemplo 2.13. *Algunos ejemplos de códigos perfectos.*

1. *Código de repetición binario de longitud impar, un caso particular es* $Rep(5)$=$\{(x,x,x,x,x) : x \in \mathbb{F}_2\}$*, como se observó en el ejemplo anterior.*

2. *Código total que consiste en todos los elementos de* $\mathbb{F}_q^n$*.*

Ejemplo 2.14. *La yuxtaposición de los vectores* $u = (u_1, u_2, u_3, u_4)$ *y* $v = (v_1, v_2, v_3)$ *es* $(u||v) = (u_1, u_2, u_3, u_4, v_1, v_2, v_3)$

Teorema 2.15 (Construcción de Plotkin : $(u||u+v)$**).** *Si C_1 es algún código binario de longitud n, con M_1 palabras código, y $d(C_1) = d_1$, C_2 es otro código binario de longitud n, con M_2 palabras código, y $d(C_2) = d_2$, entonces el código binario $C_3 := \{(u||u+v) : u \in C_1, v \in C_2\}$, donde $(u||u+v)$ representa la yuxtaposición de las palabras u y $u+v$, es un código binario de longitud $2n$ con $M_1 \cdot M_2$ palabras código, y $d(C_3) = \min\{2d_1, d_2\}$.*

Demostración. Claramente, C_3 es un código de longitud $2n$ sobre $\mathbb{F}_2$, pues es la suma de las longitudes de los códigos C_1 y C_2. Consideremos ahora la función $\varphi : C_1 \oplus C_2 \to C_3$ definida por

$$\varphi(u, v) := (u, u + v).$$

Note que φ es isomorfismo entre espacios vectoriales. Entonces se tiene que

$$\dim_K C_3 = \dim_K C_1 \oplus C_2 = k_1 + k_2.$$

Así si C_1 y C_2 tienen $M_1 = 2^{k_1}$ y $M_2 = 2^{k_2}$ palabras código, respectivamente, entonces C_3 tendrá $2^{k_1+k_2} = M_1 \cdot M_2$ palabras código.

Resta demostrar la afirmación sobre la distancia mínima. Si C_3 es el espacio vectorial nulo, entonces $d_1 = d_2 = 0$. Supongamos entonces que $C \neq \{0\}$ y sea $0 \neq c = (u, u + v) \in C_3$.

Dado que $\text{wt}(u) \geq |\text{sop}(u) \cap \text{sop}(v)|$, donde $\text{sop}(v) = |\{i | v_i \neq 0\}|$, se sigue que

$$\begin{aligned}\text{wt}(c) &= \text{wt}(u) + \text{wt}(u+v) \\ &= \text{wt}(u) + \text{wt}(u) + \text{wt}(v) - 2|\text{sop}(u) \cap \text{sop}(v)| \\ &\geq \text{wt}(v).\end{aligned}$$

Si $v \neq 0$, entonces dado que $\text{wt}(v) \geq d_2$, se tiene que $\text{wt}(c) \geq d_2$. Si $v = 0$, entonces dado que $c \neq 0$, se tiene que $\text{wt}(c) = 2\text{wt}(u) \geq 2d_1$. En resumen se tiene
$d \geq \text{mín}\{2d_1, d_2\}$.
Con $u = 0$ y v con peso mínimo o lo contrario, se alcanza el mínimo. □

Si suponemos que para cualquier número natural n, E_n es el conjunto de vectores en $\mathbb{F}_2^n$ que tienen peso par. Esto, junto con la construcción de Plotkin, produce una interesante familia de códigos: por ejemplo, eligiendo C_1 como
$E_4 = \{(0,0,0,0), (1,1,0,0), (1,0,1,0), (0,1,0,1), (0,0,1,1), (0,1,1,0), (1,0,0,1), (1,1,1,1)\}$
y C_2 como el código de repetición binario de longitud 4, luego usando la construcción de Plotkin obtenemos C_3 y extendiendo este tipo de construcción con un código de repetición binario de longitud 8. Obtenemos que $|E_4| = 8 = 2^3$, $|C_3| = 2 \cdot 8 = 2^{3+1}$. Al hacer esta construcción una y otra vez, obtenemos códigos con longitud 2^m, teniendo 2^{m+1} palabras de código, y distancia $2^m - 1$ para $m \geq 2$. Estos códigos son conocidos como los códigos Reed-Muller de primer orden.

Teorema 2.16. *Los códigos Reed-Muller de primer orden son códigos perfectos, es decir, estos códigos alcanzan el límite de empaquetado de esferas.*

Demostración. Ver Huffman y Pless, 2010 teorema 1.10.1. □

Ejemplo 2.17 (Aplicación en ISBN)**.** *El ISBN (Número Internacional Normalizado del Libro), un número de 10 dígitos, por ejemplo $x_1x_2x_3 \ldots x_{10}$ está diseñado de tal manera que satisface $\sum_{i=1}^{10} i \cdot x_i \equiv 0 \mod 11$ o en algunos códigos más antiguos se satisface $\sum_{i=1}^{5}[3 \cdot x_{2i} + x_{2i-1}] \equiv 0 \mod 11$. Esto se llama suma ponderada. Así basta comprobar la congruencia, para detectar si hay un error o no en el ISBN.*

2.2. Códigos lineales

Los códigos lineales de longitud n sobre $\mathbb{F}_q$ son subespacios de $\mathbb{F}_q^n$. Por lo tanto, la dimensión de estos subespacios es finita. Dado que un código lineal es un espacio vectorial, todos sus elementos pueden describirse en términos de una base ya que $\mathbb{F}_q^n$ tiene dimensión finita. En teoría de codificación, una base para un código lineal se representa a menudo en forma de una matriz, llamada matriz generadora, mientras que una matriz que representa una base para el código dual se llama matriz de comprobación de paridad. Estas matrices desempeñan un papel importante en la teoría de codificación.

Definición 2.18 (Código lineal q-ario)**.** *Si C es un subespacio de $\mathbb{F}_q^n$ de dimensión k, entonces se dice que C es un código $[n, k]$ q-ario ; y si además $d(C) = d$, entonces C es un código $[n, k, d]$ q-ario.*

Lema 2.19. *Un código q-ario $[n, k, d]$ contiene q^k palabras código. Si x y $y \in \mathbb{F}_q^n$, entonces $d(x, y) = wt(x - y)$.*

Demostración. Sea $x \in C$ y sean $\{b_1, b_2, \ldots, b_k\}$, una base de C, entonces $x = \alpha_1 b_1 + \alpha_2 b_2 + \ldots + \alpha_k b_k$ con $\alpha_i \in \mathbb{F}_q$ así por la regla del producto existen q^k combinaciones posibles.

Además, las coordenadas no nulas de $x - y$ son las coordenadas en las que x y y difieren, es decir, $d(x, y) = wt(x - y)$. □

El código C es un subespacio vectorial de $\mathbb{F}_q^n$ con espacio dual definido como:

Definición 2.20 (Código dual)**.** *Sea C un código lineal $[n, k]$ sobre $\mathbb{F}_q$. El código dual es el código $[n, n - k]$ dado por $C^\perp := \{v \in \mathbb{F}_q^n | v \cdot w = 0 \, \forall w \in C\}$ donde $\cdot$ representa el producto interno usual entre vectores, es decir $v \cdot w = \sum_{i=1}^n v_i w_i$.*

Teorema 2.21. *Sea C un código lineal y $w(C)$ sea el menor de los pesos entre las palabras código no nulas de C, entonces $d(C) = w(C)$.*

Demostración. Mostremos que $d(C) \leq w(C)$. Sea $w(C)$ el peso mínimo de un código lineal C, donde $w(C) = w(x)$ para algún $x \in C$, como $w(x) = d(x, 0) \geq d(C)$, luego $w(C) \geq d(C)$. Por otro lado, veamos que, $d(C) \geq w(C)$. En efecto, sea $d(C)$ la distancia mínima del código

lineal, luego existen $x, y \in C$ tales que $d(C) = d(x, y) = w(x - y) \geq w(C)$, luego $d(C) \geq w(C)$ de donde se concluye que $d(C) = w(C)$. □

Calcular el peso mínimo en lugar de la distancia mínima en códigos lineales ofrece una ventaja importante en términos de eficiencia computacional. Determinar el peso mínimo, es decir, el menor número de símbolos no nulos en una palabra código, es computacionalmente menos exigente que encontrar la distancia mínima, que implica comparar todas las posibles combinaciones de dos palabras código en busca de la menor distancia. Al centrarse en el peso mínimo, se simplifica el proceso de evaluación y comparación de códigos, lo que facilita el diseño y análisis de sistemas de corrección y detección de errores, sin comprometer la precisión esencial de la evaluación de la calidad del código en términos de su capacidad para resistir errores en la transmisión de datos.

Definición 2.22. *La **matriz generadora** para un código C lineal q-ario $[n, k]$ es una matriz $k \times n$, cuyas filas forman una base del espacio vectorial C sobre $\mathbb{F}_q$. Es decir, $C = \{xG : x \in \mathbb{F}_q^k\}$ y, por lo tanto, es la imagen de la transformación lineal*

$$\mathbb{F}_q^k \longrightarrow \mathbb{F}_q^n$$

$$m \longrightarrow xG.$$

Definición 2.23. *Sea C un código lineal q-ario $[n, k]$ con matriz generadora G, entonces una matriz H de tamaño $(n - k) \times n$ se llama **matriz de control de paridad** para C si $GH^T = 0$, donde H^T es la traspuesta de la matriz H.*

Las dos matrices definidas anteriormente, representan las dos formas de definir un espacio vectorial, ya que

$$C = \{xG : x \in \mathbb{F}_q^k\} = c.$$

Teorema 2.24. *Sea C un código lineal $[n, k]$ y sea $G^\perp$ una matriz que genera $C^\perp$. Entonces $H = G^\perp$ es una matriz de control de paridad de C, además, tenemos que $C = ker(H)$, es decir, es el espacio nulo de H, esto es, $C = \{x \in \mathbb{F}_q^n : H \cdot x = 0\}$*

Demostración. Queremos ver que $G(G^\perp)^T = 0$. Si $\{v_1, v_2, \cdots, v_k\}$ son las filas de G y

$\{w_1, w_2, \cdots, w_{n-k}\}$ son las columnas de $(G^{\perp})^T$, es decir, las filas de $G^{\perp}$, entonces $v_i \cdot w_j = 0$ $\forall i, j$ y, por lo tanto $G_{k\times n}(G^{\perp})^T_{n\times(n-k)} = 0_{k\times(n-k)}$.

Ahora veamos que $C = \ker(H)$. Si $\{w_1, w_2, \cdots, w_{n-k}\}$ son las columnas de $(G^{\perp})^T$, es decir, las filas de $G^{\perp}$ y sea $x \in C$ entonces $w_i \cdot x = 0$ $\forall i = 1, 2, \cdots, n-k$ lo que implica que $H \cdot x = 0$. Por lo tanto $C \subseteq \ker(H)$, y como $dim(C) = k = dim(\ker(H))$, ya que $dim(\ker(H)) = dim(\mathbb{F}_q^n) - dim(Im(H)) = n - (n-k) = k$, entonces $C = \ker(H)$. □

Ejemplo 2.25. *Sea $C = \{000, 111\}$ el código de repetición binario $[3, 1]$, con su código dual $C^{\perp} = \langle 101, 011\rangle$. Por lo tanto, podemos tomar a la matriz generadora de $C^{\perp}$,*
$G^{\perp} = H = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$*, como la matriz de control de paridad de C.*

Note que las matrices generadoras no son únicas, puesto que las bases de un espacio vectorial no son únicas, más aún, incluso para la misma base está, una permutación (diferente de la identidad) de las filas de una matriz generadora también da lugar a una matriz generadora diferente.

Definición 2.26. *Cualquier par de códigos lineales son **equivalentes** si la matriz generadora de uno se puede obtener del otro mediante una o la combinación de operaciones elementales entre las filas y las columnas:*

- *Permutación de filas,*
- *Multiplicación de filas por un escalar no nulo,*
- *Adición de un múltiplo escalar de una fila a otra,*
- *Permutación de columnas,*
- *Multiplicación de cualquier columna por un escalar no nulo.*

Note que códigos equivalentes tienen los mismos parámetros fundamentales, es decir, misma longitud, dimensión y distancia mínima ya que:

- La longitud de un código es el número de coordenadas en cada palabra de código. Dado que la equivalencia se basa en una permutación de las coordenadas o multiplicaciones

escalares, la longitud debe ser la misma.

- La dimensión de un código es la cantidad de palabras de código linealmente independientes que forman una base para el código. Como la equivalencia no cambia la estructura lineal del espacio vectorial que forman los códigos, la dimensión también es la misma.

- La equivalencia solo reordena las coordenadas o multiplica por escalares, lo que no afecta la distancia de mínima entre las palabras de código.

Definición 2.27. *Una matriz generadora de la forma* $(I_k|X)$ *donde* I_K *es la matriz identidad y* X *es una matriz con las columnas adicionales que permiten generar el código, se dice que está en* ***forma estándar****. Una matriz de control de paridad en la forma* $(Y|I_{n-k})$ *se dice que está en forma estándar.*

Si $G = (I_k|X)$ es la matriz generadora en forma estándar de C, un código $[n, k]$, entonces una matriz de control de paridad para C es $H = (-X^T|I_{n-k})$, pues se comprueba que $G \cdot H^T = (I_k|X)\begin{pmatrix} -(X^T)^T \\ I_{n-k} \end{pmatrix} = -X + X = 0$. Note que todo código lineal es equivalente a un código con matriz generadora en forma estándar.

Ejemplo 2.28. *El código* $C = \{000, 001, 100, 101\}$ *es un código binario*$[3, 2, 1]$ *y su matriz generadora no tiene forma estándar. La matriz generadora es*

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

Pero este código es equivalente al código $C = \{000, 100, 010, 110\}$ *cuya matriz generadora tiene forma estándar*

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

.

Finalmente, veremos como podemos usar la matriz de control de paridad para obtener información sobre la distancia mínima de un código lineal.

Lema 2.29. *Sea* C *un* $[n, k, d]$ *código lineal con matriz de control de paridad* H*. Si en* H *hay* j *columnas linealmente dependientes, entonces* C *contiene unapalabra código con elementos*

distintos de cero en algunas de sus posiciones.

Demostración. Dado que j columnas de H son linealmente dependientes, entonces existe al menos una combinación lineal, no trivial, de estas columnas que suma cero, y dado que $C = \{y \in \mathbb{F}_q^n : Hy^T = 0\}$ el código contiene una palabra con elementos distintos de cero en algunas de sus posiciones. □

Teorema 2.30. *Sea C un código lineal y sea H una matriz de control de paridad para C. Entonces:*

1. *$d(C) \geq d$ si y solo si cualquier $d-1$ columnas de H son linealmente independientes.*

2. *$d(C) \leq d$ si y solo si H tiene d columnas que son linealmente dependientes.*

Demostración. 1. Consideremos $d-1$ columnas de H, denotadas por $H_{i_1}, H_{i_2}, \ldots, H_{i_{d-1}}$, donde $i_j \in \{1, 2, \ldots, n\}$. Supongamos que estas columnas son linealmente dependientes. Entonces existen elementos, no todos nulos, $v_{i_1}, v_{i_2}, \ldots, v_{i_{d-1}} \in \mathbb{F}_q$ tales que $H_{i_1}v_{i_1} + H_{i_2}v_{i_2} + \ldots + H_{i_{d-1}}v_{i_{d-1}} = 0$, donde 0 representa el vector cero. Consideremos el vector $v = (v_1, v_2, \ldots, v_n)$ en $\mathbb{F}_q^n$, definido por $v_i = v_{i_j}$ si $i \in \{i_1, i_2, \ldots, i_{d-1}\}$ y $v_i = 0$ si $i \notin \{i_1, i_2, \ldots, i_{d-1}\}$, entonces $Hv = 0$, con lo cual $v \in C$ y $wt(v) \leq d-1$, esto implica que $d(C) \leq d-1$.

Recíprocamente, si $d(C) < d$, entonces existe $v \in C$ tal que $wt(v) \leq d-1$, es decir, v tiene a lo sumo $d-1$ coordenadas no nulas. Por otro lado, $Hv = 0$ lo cual representa una combinación lineal de $d-1$ columnas de H, donde no todos los escalares son nulos, esto es, $d-1$ columnas de H linealmente dependientes.

2. Supongamos que $d(C) \leq d$. Esto implica que existe un vector $v \in C$ tal que $wt(v) \leq d$. Denotemos las coordenadas no nulas de v por $v_{i_1}, v_{i_2}, \ldots, v_{i_k}$, donde $k \leq d$.

Sin pérdida de generalidad, supongamos que $k = d$ (si $k < d$, podemos agregar ceros para que $k = d$). Entonces $Hv = 0$ donde las columnas de H correspondientes a las coordenadas $v_{i_1}, v_{i_2}, \ldots, v_{i_d}$ son $H_{i_1}, H_{i_2}, \ldots, H_{i_d}$; esto significa que $H_{i_1}v_{i_1} + H_{i_2}v_{i_2} + \cdots + H_{i_d}v_{i_d} = 0$

con $v_{i_1}, v_{i_2}, \ldots, v_{i_d}$ no son todos cero, lo que implica que $H_{i_1}, H_{i_2}, \ldots, H_{i_d}$ son linealmente dependientes.

Por otra parte, si H tiene d columnas que son linealmente dependientes. Es decir, existen d columnas $H_{i_1}, H_{i_2}, \ldots, H_{i_d}$ de H tales que: $H_{i_1}v_{i_1} + H_{i_2}v_{i_2} + \cdots + H_{i_d}v_{i_d} = 0$ para algún conjunto no trivial de coeficientes $v_{i_1}, v_{i_2}, \ldots, v_{i_d}$, no todos ceros. Considere el vector $v \in \mathbb{F}_q^n$, definido por $v_i = v_{i_j}$ si $i \in \{i_1, i_2, \ldots, i_d\}$ y $v_i = 0$ si $i \notin \{i_1, i_2, \ldots, i_d\}$. Entonces $Hv = 0$, con lo cual $v \in C$ y $wt(v) \leq d$. Por lo tanto, $d(C) \leq wt(v) \leq d$. □

Corolario 2.31. *Sea C un código lineal y sea H una matriz de control de paridad para C. Entonces las siguientes afirmaciones son equivalentes:*

1. *C tiene distancia mínima d.*

2. *Cualquier $d-1$ columnas de H son linealmente independientes y H tiene d columnas que son linealmente dependientes.*

2.2.1. Codigos lineales usando SageMath

En Sage existen tres formas distintas de construir un código lineal:

- A partir de la matriz generadora:

  ```
  In: codes.LinearCode(self, generator, d=None)
  ```

  En el código proporcionado, el parámetro generator hace referencia a la matriz generadora definida sobre un cuerpo finito. El parámetro d es opcional y se utiliza para indicar la distancia mínima del código, aunque normalmente no se emplea.

- Indicando el cuerpo sobre el que queremos construir el código, junto con su dimensión y longitud:

  ```
  In: codes.random_linear_code(F, length, dimension)
  ```

  Donde el símbolo F denota el cuerpo que estamos utilizando, mientras que length representa la longitud y dimension indica la dimensión deseada para el código lineal. La función mencionada construye el código generando matrices de tamaño dimension $\times$ length

con elementos del cuerpo F hasta encontrar una matriz cuyo rango sea máximo. Esta matriz se puede utilizar como matriz generadora del código lineal que estamos construyendo.

- A partir de la matriz de control de paridad:

  ```
  In: codes.from_parity_check_matrix(H)
  ```

  Donde H es la matriz de control de paridad del código lineal.

Además, en Sage una vez se ha definido el código lineal C se puede obtener su matriz generadora y su matriz de control de paridad usando respectivamente:

```
In: C.generator_matrix()
```

```
In: C.systematic_generator_matrix()
```

```
In: C.parity_check_matrix()
```

Por otro lado, para obtener el código dual Sage tiene la función:

```
In: C.dual_code()
```

Así mismo, en Sage existen funciones que permiten obtener la distancia mínima de un código C dado, esto para cuerpos finitos de menos de 2^8 elementos:

```
In: C.minimum_distance()
```

Y también nos proporciona una función para obtener el número de palabras de peso i, en un código C:

```
In: C.weight_distribution()
```

Veamos un ejemplo de como funcionan cada una de las funciones descritas

```
[1]:
```

```
C=codes.
 ↪LinearCode(Matrix(GF(2),[[1,1,1,0,0,0,0],[1,0,0,1,1,0,0],[0,1,0,1,0,1
 ↪
[ 1 ,1,0,1,0,0,1]]))
```

[2]:
```
C
```

[2]: [7, 4] linear code over GF(2)

[3]:
```
list(C)
```

[3]:
```
[(0, 0, 0, 0, 0, 0, 0),
 (1, 1, 1, 0, 0, 0, 0),
 (1, 0, 0, 1, 1, 0, 0),
 (0, 1, 1, 1, 1, 0, 0),
 (0, 1, 0, 1, 0, 1, 0),
 (1, 0, 1, 1, 0, 1, 0),
 (1, 1, 0, 0, 1, 1, 0),
 (0, 0, 1, 0, 1, 1, 0),
 (1, 1, 0, 1, 0, 0, 1),
 (0, 0, 1, 1, 0, 0, 1),
 (0, 1, 0, 0, 1, 0, 1),
 (1, 0, 1, 0, 1, 0, 1),
 (1, 0, 0, 0, 0, 1, 1),
 (0, 1, 1, 0, 0, 1, 1),
 (0, 0, 0, 1, 1, 1, 1),
 (1, 1, 1, 1, 1, 1, 1)]
```

[4]:
```
C1=codes.random_linear_code(GF(2),7,4)
```

[5]:
```
list(C1)
```

```
[5]: [(0, 0, 0, 0, 0, 0, 0),
      (0, 1, 1, 0, 1, 0, 1),
      (1, 1, 1, 0, 1, 0, 0),
      (1, 0, 0, 0, 0, 0, 1),
      (1, 1, 0, 1, 1, 1, 1),
      (1, 0, 1, 1, 0, 1, 0),
      (0, 0, 1, 1, 0, 1, 1),
      (0, 1, 0, 1, 1, 1, 0),
      (1, 1, 0, 0, 1, 0, 0),
      (1, 0, 1, 0, 0, 0, 1),
      (0, 0, 1, 0, 0, 0, 0),
      (0, 1, 0, 0, 1, 0, 1),
      (0, 0, 0, 1, 0, 1, 1),
      (0, 1, 1, 1, 1, 1, 0),
      (1, 1, 1, 1, 1, 1, 1),
      (1, 0, 0, 1, 0, 1, 0)]
```

```
[6]: C.parity_check_matrix()
```

```
[6]: [1 0 1 0 1 0 1]
     [0 1 1 0 0 1 1]
     [0 0 0 1 1 1 1]
```

```
[7]: G=C1.parity_check_matrix()
```

```
[8]: codes.from_parity_check_matrix(G)
```

```
[8]: [7, 4] linear code over GF(2)
```

```
[9]: C.generator_matrix()
```

```
[9]: [1 1 1 0 0 0 0]
     [1 0 0 1 1 0 0]
     [0 1 0 1 0 1 0]
```

```
[1 1 0 1 0 0 1]
```

```
[10]: C2=C.dual_code()
```

```
[11]: C2
```

```
[11]: [7, 3] linear code over GF(2)
```

```
[12]: C2.generator_matrix()
```

```
[12]: [1 0 1 0 1 0 1]
      [0 1 1 0 0 1 1]
      [0 0 0 1 1 1 1]
```

```
[13]: C2.minimum_distance()
```

```
[13]: 4
```

```
[14]: C2.weight_distribution()
```

```
[14]: [1, 0, 0, 0, 7, 0, 0, 0]
```

```
[15]: C.weight_distribution()
```

```
[15]: [1, 0, 0, 7, 7, 0, 0, 1]
```

```
[0]:
```

En Sage el código que nos permite comprobar si dos códigos son equivalentes es:

```
In: C1.is_permutation_equivalent(other, algorithm=None)
```

```
[3]: C1=codes.LinearCode(Matrix(GF(2),[[0,0,1],[1,0,0]]))
```

```
[4]: C2=codes.LinearCode(Matrix(GF(2),[[1,0,0],[0,1,0]]))
```

```
[5]: C3=codes.LinearCode(Matrix(GF(2),[[1,0,1,1],[0,1,1,0]]))
```

```
[6]: C1.is_permutation_equivalent(C2, algorithm=None)
```

```
[6]: True
```

```
[7]: C1.is_permutation_equivalent(C3, algorithm=None)
```

```
[7]: False
```

La teoría de codificación proporciona una amplia gama de códigos lineales que se emplean para codificar y decodificar mensajes. Estos códigos se pueden entender como transformaciones reversibles (se puede recuperar la entrada original) en espacios vectoriales. A lo largo de la siguiente sección, exploraremos en detalle los procesos de codificación y decodificación, analizando las complejidades detrás de la transferencia segura de información.

2.3. Codificación de códigos lineales

Sea C un código lineal $[n, k, d]$ sobre el cuerpo finito $\mathbb{F}_q$. Cada palabra código de C representa una pieza de información, por lo que C representa q^k piezas de información distintas. Supongamos que $m = (m_1, m_2, \ldots, m_k)$ es un mensaje en $\mathbb{F}_q^k$, entonces por codificación, nos referimos a una transformación lineal inyectiva cod : $\mathbb{F}_q^k \to \mathbb{F}_q^n$, si dicha transformación lineal está dada por una matriz en forma estándar, esto puede ser interpretado como añadir $n - k$ símbolos adicionales al mensaje m como

$$(m_1, m_2, \ldots, m_k) \mapsto (m_1, m_2, \ldots, m_k, \underbrace{m_{k+1}, \ldots, m_n}_{\text{redundancia}}).$$

Sea C un código [n, k] sobre $\mathbb{F}_q$ con matriz generadora G. Entonces, C contiene q^k palabras, código y, por lo tanto, C puede utilizarse para comunicar q^k mensajes simultáneamente. La codificación con un código lineal se realiza multiplicando el vector mensaje con la matriz generadora del código lineal, es decir, $m \mapsto m \cdot G$. El vector resultante $m \cdot G$ es una palabra código de C, ya que es una combinación lineal de las filas de la matriz generadora.

Supongamos que la palabra código $x = (x_1, x_2, \ldots, x_n)$ se envía a través del canal y que el vector $y = (y_1, y_2, \ldots, y_n)$ es recibido. Definimos el vector error e como:

$$e = y - x = (e_1, e_2, \ldots, e_n).$$

Antes de ver el proceso de decodificación examiemos primero la noción de una clase lateral, estas desempeñan un papel esencial en muchos esquemas de decodificación.

Definición 2.32. *Sea C un código lineal de longitud n sobre $\mathbb{F}_q$, y sea $u \in \mathbb{F}_q^n$ un vector; definimos la **clase** de u en C como el conjunto*

$$u + C = \{u + v : v \in C\}.$$

Este conjunto es la clase de equivalencia de u bajo la relación $a \sim b$ si $a - b \in C$ en $\mathbb{F}_q^n$

Teorema 2.33. *Sea C un código lineal $[n, k, d]$ sobre $\mathbb{F}_q$. Entonces,*

1. *Para cada vector $u \in \mathbb{F}_q^n$, se tiene que $u \in u + C$.*
2. *Para todo $u \in \mathbb{F}_q^n$, $|u + C| = |C| = q^k$.*
3. *Para todo $u, v \in \mathbb{F}_q^n$, $u \in v + C \longleftrightarrow u + C = v + C$.*
4. *Dos clases son idénticas o tienen intersección vacía.*
5. *Hay q^{n-k} clases diferentes de C.*
6. *Para todo $u, v \in \mathbb{F}_q^n$, $u - v \in C$ si y solo si $u + C = v + C$.*

Demostración.

1. Dado un vector $u \in \mathbb{F}_q^n$, como $0 \in C$ y $u = u + 0$, entonces $u \in u + C$.
2. Dado que C es un subespacio de $\mathbb{F}_q^n$ con dimensión k, tiene q^k elementos. $|u + C| = |\{u + v : v \in C\}| \leq |C|$. Por otro lado, note que $u + v \neq u + w$ si $v \neq w$, por lo tanto, $|u + C| = |C| = q^k$.
3. Supongamos que $u \in v + C$. Entonces, existe un $w \in C$ tal que $u = v + w$. Consideremos un vector arbitrario $x \in u + C$. Entonces, $x = u + c$ para algún $c \in C$. Sustituyendo $u = v + w$, obtenemos $x = v + w + c$. Como $w \in C$ y como C es un subespacio, $w + c \in C$. Por lo tanto, $x \in v + C$. Esto muestra que $u + C \subseteq v + C$. De manera similar, $v + C \subseteq u + C$, lo que implica que $u + C = v + C$.
4. Supongamos que $u + C$ y $v + C$ tienen una intersección no vacía. Esto significa que existe un vector $x \in \mathbb{F}_q^n$ tal que $x \in u + C$ y $x \in v + C$. Por lo tanto, podemos escribir

$x = u + c_1$ y $x = v + c_2$ para algunos $c_1, c_2 \in C$. Esto implica que $u + c_1 = v + c_2$, o $u - v = c_2 - c_1 \in C$. Por lo tanto, $u \in v + C$, lo que implica que $u + C = v + C$ por el punto anterior.

5. El espacio vectorial $\mathbb{F}_q^n$ tiene q^n elementos, y cada clase $u + C$ contiene q^k elementos. Dado que las clases son disjuntas o idénticas, el número total de clases es $q^n/q^k = q^{n-k}$.

6. ($\Rightarrow$) Supongamos que $u - v \in C$. Entonces, $u = (u - v) + v$ donde $u - v \in C$. Esto implica que $u \in v + C$ y, por lo tanto, $u + C = v + C$.

   ($\Leftarrow$) Supongamos que $u + C = v + C$, entonces $u \in v + C$, lo que significa que $u = v + c$ para algún $c \in C$. Esto implica que $u - v = c \in C$. □

Definición 2.34. *Un vector en una clase se llama **líder** de la clase si tiene el mínimo peso de Hamming.*

Como veremos más adelante, el líder de la clase va a ser importante para establecer un proceso de decodificación de códigos lineales en general, utilizando clases laterales.

2.3.1. Codificación usando SageMath

Sage también nos proporciona una serie de funciones relacionadas con la codificación:

- Para codificar un vector $word$ según un codificador específico, utilizamos la función: `In: C.encode(word, encoder_name=None, *args)`

  donde el parámetro $word$ hace referencia al vector que deseamos codificar. Además, los parámetros opcionales $encoder\_name$ y $*args$ indican el nombre del codificador y el conjunto de parámetros que se desean pasar al codificador, respectivamente.

- Podemos consultar los codificadores disponibles para el código lineal C mediante la función:
  `In: C.encoders_available()`

- Para obtener un codificador específico, utilizamos la función:
  `In: C.encoder(encoder_name, *args)`

que tiene dos parámetros opcionales: $encoder\_name$, donde introducimos el nombre del codificador que queremos obtener, y $*args$, donde introducimos los argumentos que deseamos pasar al codificador indicado.

- Finalmente, para incorporar otro codificador que hayamos construido para C, utilizamos la función:

  ```
  In: C.add_encoder(name, decoder)
  ```

  donde el parámetro $name$ indica el nombre del nuevo codificador y en el parámetro $decoder$ introducimos el nombre de la función que implementa el nuevo codificador.

Continuando con el ejemplo anterior en Sage, veamos como se pueden codificar las palabras usando las funciones mencionadas:

[3]:
```
C=codes.
↪LinearCode(Matrix(GF(2),[[1,1,1,0,0,0,0],[1,0,0,1,1,0,0],
[0,1,0,1,0,1,0],  [1,1,0,1,0,0,1]]))
```

[4]:
```
encoders = C.encoders_available()
print(encoders)
```

```
['GeneratorMatrix', 'Systematic']
```

[5]:
```
encoder1 = C.encoder('GeneratorMatrix')
encoder1
```

[5]:
```
Generator matrix-based encoder for [7, 4] linear code over␣
↪GF(2)
```

[6]:
```
m=vector(GF(2),[1, 0, 1, 1])
m
```

[6]:
```
(1, 0, 1, 1)
```

[7]:
```
c=encoder1(m)
```

```
[8]: c

[8]: (0, 1, 1, 0, 0, 1, 1)

[9]: v=vector(GF(2),[1,1,0,1])

[10]: def encoder2(vector):
          # Implementación de tu propio codificador
          return vector+v


      C.add_encoder("encoder2", encoder2)

[11]: encoder2(m)

[11]: (0, 1, 1, 0)

[12]: codeword = C.encode(m, encoder_name='Systematic')

[13]: codeword

[13]: (1, 0, 1, 1, 0, 1, 0)
```

2.4. Decodificación de códigos lineales

En general, el problema de la decodificación de un código lineal arbitrario es un problema difícil, ya que no existe un algoritmo único que sea eficiente para cualquier código; sin embargo, existen familias de códigos lineales para los cuales se pueden establecer algoritmos eficientes que garantizan una decodificación óptima.

El decodificador debe decidir, a partir del vector y recibido, qué palabra código c se transmitió, o equivalentemente, determinar cuál es el correspondiente vector error e en la transmisión. Este proceso se llama decodificación con un código lineal y se hace bajo el principio de que el vector error cometido es pequeño, es decir, el vector recibido se decodifica como la palabra código más cercana, o con menor distancia de Hamming y se conoce como decodificación del vecino próximo.

Sea C un $[n, k, d]$ código lineal sobre $\mathbb{F}_q$. Supongamos que se transmite la palabra código c y se recibe el vector $y = c + e$, donde e es el vector error. La decodificación puede verse como la función que primero corrige el error y después devuelve los símbolos de información $m \in \mathbb{F}_q^k$ que se codificaron con la palabra código c. Es decir,

$$\begin{aligned} \text{dec}: \quad \mathbb{F}_q^n &\longrightarrow C \longrightarrow \mathbb{F}_q^k \\ y &\longrightarrow c \longrightarrow m \end{aligned}$$

Note que el vector error $e = y - c \in y + C$. Por lo tanto, el vector error e y el vector recibido y están en la misma clase. Dado que los vectores error con peso Hamming pequeño son los más probables de ocurrir, entonces el vector error es el líder de la clase en la que aparece el vector y y se obtiene la palabra código transmitida $c = y - e$. Esto sugiere tener pre computado las clases laterales y sus líderes para el código C. Este método de decodificación es conocido como arreglo estandar.

Ejemplo 2.35. *Sea* $C = \{000000, 110100, 011010, 101110, 101001, 011101, 110011, 000111\}$ *un código linear sobre* $\mathbb{F}_2^6$ *entonces el arreglo estandar de* C *es:*

Cuadro 2: Ejemplo arreglo estándar

$000000 + C$	000000	110100	011010	101110	101001	011101	110011	000111
$000001 + C$	000001	110101	011011	101111	101000	011100	110010	000110
$000010 + C$	000010	110110	011000	101100	101011	011111	110001	000101
$000100 + C$	000100	110000	011110	101010	101101	011001	110111	000011
$001000 + C$	001000	111100	010010	100110	100001	010101	111011	001111
$010000 + C$	010000	100100	001010	111110	111001	001101	100011	010111
$100000 + C$	100000	010100	111010	001110	001001	111101	010011	100111
$010001 + C$	010001	100101	001011	111111	111000	001100	100010	100110

Si se recibe 101100 *el mensaje decodificado es* 101110 *con vector error* 000010.

Observe que si el código tiene parámetros grandes, se requerirían más cálculos y almacenamiento del arreglo estándar, lo que aumenta la complejidad del algoritmo. Por esta razón tenemos la decodificación Syndrome.

2.4.1. Decodificación por síndrome

El esquema de decodificación basado en el arreglo estándar funciona razonablemente bien cuando la longitud n del código lineal es pequeña, pero aún computacionalmente puede llevar bastante tiempo cuando n es grande. Se puede ahorrar tiempo utilizando el síndrome para identificar la clase a la que pertenece el vector recibido.

Definición 2.36. *Sea C un código lineal $[n, k, d]$ sobre $\mathbb{F}_q$ y sea H una matriz de control de paridad para C. Para cualquier $w \in \mathbb{F}_q^n$, el **síndrome** de w es vector $S(w) = Hw \in \mathbb{F}_q^{n-k}$.*

Notación: Como el síndrome depende de la elección de la matriz de control de paridad H, es más apropiado denotar el síndrome de w como $S_H(w)$ para enfatizar esta dependencia. Sin embargo, por simplicidad de la notación, se omite el subíndice H cuando no hay riesgo de ambigüedad.

Lema 2.37. *Sea C un código lineal $[n, k, d]$ y sea H una matriz de control de paridad para C. Para $u, v \in \mathbb{F}_q^n$, tenemos:*
(i) $S(u + v) = S(u) + S(v)$;
(ii) $S(c) = 0$ si y solo si $c \in C$;
(iii) $S(u) = S(v)$ si y solo si u y v están en la misma clase de C.

Demostración. (i) Se sigue de la definición.

(ii) $S(c) = 0$ si y solo si $Hc = 0$, como H es la matriz de control de paridad, esto equivale a que $c \in C$.

(iii) $S(u) = S(v)$, equivale a decir que $S(u - v) = 0$ esto es, si y solo si $u - v \in C$, es decir, u y v están en la misma clase lateral. □

Teorema 2.38. *Sea C un código lineal $[n, k]$, y sea $y = c + e$ el vector recibido, con $c \in C$ y e el vector error, entonces, $S(y) = S(e)$.*

Demostración. Del lema anterior, se tiene que:

$$S(y) = S(c) + S(e) = 0 + S(e) = S(e).$$

□

Una tabla de búsqueda de síndromes relaciona a cada representante de clase con su síndrome se llama.

Pasos para construir una tabla de búsqueda de síndromes asumiendo decodificación completa del arreglo estándar:

- Se determinan todas las clases para el código y se elige el líder de cada clase, este es el vector de menor peso de la clase.
- Utilizando una matriz de control de paridad H para el código C, se calcula el síndrome $S(u) = Hu$ para cada líder u de las clases.
- se almacena una tabla con los distintos síndromes y sus respectivos líderes de clase.

La decodificación mediante síndrome funciona de la misma manera que la decodificación mediante el arreglo estándar.

1. Para el vector recibido y, se calcula el síndrome $S(y)$.
2. Se busca en la tabla de síndromes el lider u de la clase con ese síndrome.
3. Se decodifica y como $c = y - u$.

Note que el paso 2. reduce el tiempo de búsqueda en el arreglo estándar. Así como el almacenamiento de las tablas pre computadas.

Ejemplo 2.39. *Sea* $C = \langle 100011, 010101, 001110 \rangle$*, sobre*$\mathbb{F}_2$*, un código cuya matriz generadora es*

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

y la matriz de control de paridad es

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

Como $d(C) = 3$*, podemos corregir cualquier error con peso máximo de 1. Y calculando la tabla de síndromes se tiene*

Cuadro 3: Ejemplo tabla de síndromes.

e	$S(e)$
000000	000
100000	011
010000	101
001000	110
000100	100
000010	010
000001	001

Supongamos que recibimos el vector $y = (111101)$*, entonces calculamos* $S(y) = Hy = (101)$*, y buscando en la tabla de sindromes se tiene que* $e = (010000)$ *es el vector de error cometido. Por lo tanto,* $c = y - e = 111101 - 010000 = 101101$.

2.4.2. Decodificación usando SageMath

En SageMath existen una serie de funciones relacionadas con la corrección de errores y la decodificación.

Para realizar la corrección de errores de una palabra del código C tenemos la función:

```
In: C.decode_to_code(word, decoder_name=None, *args)
```

donde el parámetro $word$ hace referencia a la palabra sobre la que deseamos corregir los errores que contiene. Los parámetros $decoder\_name$ y $args$ son ambos opcionales, para los cuales, el primero indica el nombre del decodificador que queremos usar y el segundo son el conjunto de

parámetros que se desean pasar a dicho decodificador.

Para realizar la corrección de errores de la palabra y además la decodificación se emplea:

```
In: C.decode_to_message(word, decoder_name=None, *args)
```

cuyos parámetros son idénticos a los de la función $decode\_to\_code()$. Hay que destacar que si en la decodificación realizada por dicha función tenemos que obtener un vector cuyas últimas coordenadas se corresponden con elementos nulos, la función nos devuelve el vector obtenido a partir de la eliminación de dichas últimas coordenadas nulas.

Además, tenemos una función que permite ver los distintos descodificadores disponibles para el código C:

```
In: C.decoders_available()
```

Una función que nos proporciona un decodificador de los posibles para C:

```
In: C.decoder(decoder_name, *args)
```

donde debemos introducir dos parámetros, $decoder\_name$ donde indicamos el nombre de un decodificador de los todos los posibles para C, que queremos obtener y $args$ que hace referencia a los argumentos que queremos pasar al decodificador seleccionado. Por último, podemos añadir un nuevo descodificador para el código C que hayamos implementado:

```
In: C.add_decoder(name, decoder)
```

donde el parámetro name indica el nombre con el que se reconocerá el decodificador y en el parámetro decoder debemos introducir el nombre de la función en la que hemos implementado nuestro decodificador.

Veamos el funcionamiento en sage de las funciones ya mencionadas para la decodificación:

[3]:

```
C=codes.
 ↪LinearCode(Matrix(GF(2),[[1,1,1,0,0,0,0],[1,0,0,1,1,0,0],[0,1,0,1,0,1
 ↪
 [1,1,0,1,0,0,1]]))
```

[4]:
```
C.decoders_available()
```

[4]:
```
['InformationSet', 'NearestNeighbor', 'Syndrome']
```

[5]:
```
encoder1 = C.encoder('GeneratorMatrix')
encoder1
c=encoder1(vector(GF(2),[1, 0, 1, 1]))
```

[6]:
```
c
```

[6]:
```
(0, 1, 1, 0, 0, 1, 1)
```

[7]:
```
C.decoder('NearestNeighbor')
```

[7]:
```
Nearest neighbor decoder for [7, 4] linear code over GF(2)
```

[8]:
```
y=C.decode_to_message(c,'Syndrome',1)
y
```

[8]:
```
(1, 0, 1, 1)
```

[9]:
```
e=vector(GF(2),[0,0,0,0,0,0,1])#vector de error de peso de␣
 ↪Hamming 1
cc=c+e
cc
```

[9]:
```
(0, 1, 1, 0, 0, 1, 0)
```

[10]:
```
y= C.decode_to_code(cc,'Syndrome',1)
y
```

```
[10]: (0, 1, 1, 0, 0, 1, 1)
```

```
[11]: y= C.decode_to_message(cc,'Syndrome',1)
      y
```

```
[11]: (1, 0, 1, 1)
```

```
[12]: v=vector(GF(2),[1,1,0,1,0,0,1])
```

```
[13]: def decoder1(vector):
          # Implementación de tu propio decodificador
          return vector-v


      C.add_encoder("encoder2", decoder1)
```

```
[14]: decoder1(c)
```

```
[14]: (1, 0, 1, 1, 0, 1, 0)
```

2.5. Códigos clásicos de Goppa

El matemático soviético y ruso Valery Denisovich Goppa, nacido en 1939, encontró en 1970 la conexión entre la geometría algebraica y los códigos. Este hallazgo condujo a la creación de los códigos de Goppa, Goppa, 1981, 1997. Resultó que los códigos de Goppa constituyen posiblemente la subclase más llamativa de los códigos alternantes, los cuales fueron introducidos por H. J. Helgert en 1974. Estos códigos cuentan con un eficiente algoritmo de decodificación desarrollado por N. Patterson en 1975.

Definición 2.40 (Código de Goppa). *Sea* $g(z) = g_0 + g_1 z + g_2 z^2 + \ldots + g_\tau z^\tau \in \mathbb{F}_{q^m}[z]$, *y sea* $L = (\alpha_1, \alpha_2, \ldots, \alpha_n) \in \mathbb{F}_{q^m}^n$ *con* $\alpha_i \neq \alpha_j \ \ \forall i \neq j$, *tal que* $g(\alpha_i) \neq 0$ *para todo* $\alpha_i \in L$, *el código de Goppa con parámetros* $g(z)$ *y* L *es*

$$\Gamma(L, g(z)) := \left\{ c = (c_1, c_2, \ldots, c_n) \in \mathbb{F}_q^n : R_c(x) = \sum_{i=1}^{n} c_i \frac{1}{z - \alpha_i} \equiv 0 \mod g(z) \right\}.$$

Note que para cada $1 \leq i \leq n$, como $g(\alpha_i) \neq 0$, se tiene que $\text{mcd}(z - \alpha_i, g(z)) = 1$.

Si el polinomio de Goppa, $g(z)$, es irreducible, entonces el código de Goppa $\Gamma(L, g)$ se dice que es irreducible y se verifica de forma trivial que $g(\alpha_i) \neq 0$ ya que si α_i fuera una raíz de $g(z)$, entonces $g(z)$ sería reducible.

Observe que el término $\dfrac{1}{z - \alpha_i}$ se puede ver como un polinomio $p_i(z)$ módulo $g(z)$:

$$\frac{1}{z - \alpha_i} \equiv p_i(z) = p_{i1} + p_{i2}z + \ldots + p_{i\tau}z^{\tau-1} \pmod{g(z)}.$$

Esto se debe a la forma que tienen los elementos en el anillo $\mathbb{F}_{q^m}[z]/\langle g(z)\rangle$. Por lo tanto, podemos reescribir la ecuación de la definición como:

$$\sum_{i=0}^{n-1} c_i p_i(z) \equiv 0 \pmod{g(z)}.$$

Si $g(z)$ y $f(z)$ son polinomios de Goppa de cierto grado con coeficientes en $\mathbb{F}_{q^m}$ de los códigos $\Gamma(L, g)$ y $\Gamma(L, f)$ respectivamente, se verifica que si $f(z) \mid g(z)$, entonces $\Gamma(L, g) \subseteq \Gamma(L, f)$, ya que si $c \in \Gamma(L, g)$ se tiene que $R_c(z) \equiv 0 \pmod{g(z)}$ y como $f(z) \mid g(z)$ entonces $R_c(z) \equiv 0 \pmod{f(z)}$. Por lo tanto $c \in \Gamma(L, f)$.

Teorema 2.41. *El inverso multiplicativo de $(z - \alpha_i)$ existe en el anillo cociente $\mathbb{F}_{q^m}[z]/\langle g(z)\rangle$. Más exactamente,*

$$\frac{1}{z - \alpha_i} = -\frac{g(z) - g(\alpha_i)}{z - \alpha_i}\frac{1}{g(\alpha_i)}.$$

Demostración. Queremos demostrar que el inverso multiplicativo de $(z - \alpha_i)$ existe en el anillo cociente $\mathbb{F}_{q^m}[z]/\langle g(z)\rangle$. Primero definamos $u(z)$ de la siguiente manera:

$$u(z) = -\frac{g(z)}{g(\alpha_i)} + 1 = -\frac{g(z) - g(\alpha_i)}{g(\alpha_i)}.$$

Podemos ver que $u(z) \equiv 1 \pmod{g(z)}$. Entonces,

$$\frac{1}{z - \alpha_i} \equiv \frac{u(z)}{z - \alpha_i} \equiv -\frac{g(z) - g(\alpha_i)}{(z - \alpha_i) \cdot g(\alpha_i)} \pmod{g(z)},$$

por tanto, el inverso multiplicativo de $(z - \alpha_i)$ es

$$-\frac{g(z) - g(\alpha_i)}{(z - \alpha_i) \cdot g(\alpha_i)}$$

ya que

$$\frac{1}{z - \alpha_i} \cdot (z - \alpha_i) \equiv -\frac{g(z) - g(\alpha_i)}{g(\alpha_i)} \equiv 1 \qquad (\text{mód } g(z)).$$

□

Como consecuencia de este resultado, se deriva la siguiente definición alternativa:

Corolario 2.42. *Sea $\Gamma(L, g)$ el código de Goppa definido como en 2.40*

$$c = (c_1, c_2, \ldots, c_n) \in \Gamma(L, g) \textit{ si y solo si } \sum_i c_i \left(-\frac{g(z) - g(\alpha_i)}{z - \alpha_i} \frac{1}{g(\alpha_i)} \right) \equiv 0 \mod g(z).$$

Recordemos que $g(z) = g_0 + g_1 z + \ldots + g_\tau z^\tau$. Reemplazando en la ecuación anterior $g(z)$, encontramos:

$$p_i(z) = -\frac{g_\tau(z^\tau - \alpha_i^\tau) + \cdots + g_1(z - \alpha_i)}{z - \alpha_i} \frac{1}{g(\alpha_i)}.$$

simplificando podemos escribir la primera fracción como:

$$g_\tau(z^{\tau-1} + z^{\tau-2}\alpha_i + \ldots + \alpha_i^{\tau-1}) + g_{\tau-1}(z^{\tau-2} + z^{\tau-3}\alpha_i + \ldots + \alpha_i^{\tau-2}) + \ldots + g_2(z - \alpha_i) + g_1.$$

Si sustituimos $p_i(z) = p_{i1} + p_{i2} z + \ldots + p_{i\tau} z^{\tau-1}$, podemos encontrar la siguiente expresión para los coeficientes p_{ij}. Estos son:

$$
\begin{aligned}
p_{i1} &= -(g_\tau \alpha_i^{\tau-1} + g_{\tau-1}\alpha_i^{\tau-2} + \ldots + g_2\alpha_i + g_1) \cdot \frac{1}{g(\alpha_i)}, \\
p_{i2} &= -(g_\tau \alpha_i^{\tau-2} + g_{\tau-1}\alpha_i^{\tau-3} + \ldots + g_2) \cdot \frac{1}{g(\alpha_i)}, \\
&\vdots \\
p_{i(\tau-1)} &= -(g_\tau \alpha_i + g_{\tau-1}) \cdot \frac{1}{g(\alpha_i)}, \\
p_{i\tau} &= -(g_\tau) \cdot \frac{1}{g(\alpha_i)}.
\end{aligned}
$$

De esta forma, hallamos la matriz de control de paridad sobre $\mathbb{F}_{q^m}$ para los códigos de Goppa como sigue:

Corolario 2.43. *Para un código de Goppa $\Gamma(L, g(z))$ definido como en 2.40 , la matriz de control de paridad sobre $\mathbb{F}_{q^m}$ es*

$$
H = \begin{bmatrix}
\frac{g_\tau}{g(\alpha_1)} & \frac{g_\tau}{g(\alpha_2)} & \cdots & \frac{g_\tau}{g(\alpha_n)} \\
\frac{g_\tau \alpha_1 + g_{\tau-1}}{g(\alpha_1)} & \frac{g_\tau \alpha_2 + g_{\tau-1}}{g(\alpha_2)} & \cdots & \frac{g_\tau \alpha_n + g_{\tau-1}}{g(\alpha_n)} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{g_\tau \alpha_1^{\tau-1} + \cdots + g_1}{g(\alpha_1)} & \frac{g_\tau \alpha_2^{\tau-1} + \cdots + g_1}{g(\alpha_2)} & \cdots & \frac{g_\tau \alpha_n^{\tau-1} + \cdots + g_1}{g(\alpha_n)}
\end{bmatrix}.
$$

Esta matriz se factoriza como:

$$
H = ZXY = \begin{bmatrix}
\frac{g_\tau}{g(\alpha_1)} & 0 & \cdots & 0 \\
\frac{g_{\tau-1}}{g(\alpha_1)} & \frac{g_\tau}{g(\alpha_2)} & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
\frac{g_1}{g(\alpha_1)} & \frac{g_2}{g(\alpha_2)} & \cdots & \frac{g_\tau}{g(\alpha_n)}
\end{bmatrix}
\begin{bmatrix}
1 & 1 & \cdots & 1 \\
\alpha_1 & \alpha_2 & \cdots & \alpha_n \\
\vdots & \vdots & \ddots & \vdots \\
\alpha_1^{\tau-1} & \alpha_2^{\tau-1} & \cdots & \alpha_n^{\tau-1}
\end{bmatrix}
\begin{bmatrix}
\frac{1}{g(\alpha_1)} & 0 & \cdots & 0 \\
0 & \frac{1}{g(\alpha_2)} & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & \frac{1}{g(\alpha_n)}
\end{bmatrix}.
$$

Ahora, dado que $c \in \Gamma(L, g)$ se tiene que $Hc = 0$, lo que implica $(ZXY)c = 0$, o lo que es equivalente a $(XY)c = 0$ (ya que la matriz Z es invertible).

La matriz XY puede considerarse como la matriz de control de paridad para $\Gamma(L, g)$ sobre

$\mathbb{F}_{q^m}$. La matriz XY está dada por:

$$XY = \begin{bmatrix} g(\alpha_1)^{-1} & g(\alpha_2)^{-1} & \cdots & g(\alpha_n)^{-1} \\ \alpha_1 g(\alpha_1)^{-1} & \alpha_2 g(\alpha_2)^{-1} & \cdots & \alpha_n g(\alpha_n)^{-1} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_1^{\tau-1} g(\alpha_1)^{-1} & \alpha_2^{\tau-1} g(\alpha_2)^{-1} & \cdots & \alpha_n^{\tau-1} g(\alpha_n)^{-1} \end{bmatrix}.$$

Considerando los elementos de $\mathbb{F}_{q^m}$ como vectores de longitud m sobre $\mathbb{F}_q$ mediante un isomorfismo de espacios vectoriales, tenemos que una matriz de control de paridad para $\Gamma(L, g)$ sobre $\mathbb{F}_q$ es una matriz de $m\tau \times n$, con al menos τ columnas linealmente independientes sobre $\mathbb{F}_q$. Por lo tanto, la distancia de Hamming del código de Goppa, $d(\Gamma(L, g)) \geq \tau + 1$. Dado que, para la matriz XY sobre $\mathbb{F}_q$, como máximo $m\tau$ filas son linealmente independientes, se tiene que rango$(XY) \leq m\tau$, lo que implica que la nulidad $(XY) \geq n - m\tau$. Por lo tanto, la dimensión del código de Goppa, $\dim_{\mathbb{F}_q} \Gamma(L, g) \geq n - m\tau$.

Esto prueba el siguiente teorema que resume los paramétros fundamentales del código de Goppa.

Teorema 2.44. *El código $\Gamma(L, g)$ es un $[n, k, d]$ código lineal q^m-ario con parametros:*

1. *(longitud) $n = |L|$,*
2. *(dimensión) $k \geq n - m\tau$,*
3. *(distancia mínima) $d \geq \tau + 1$.*

$\mathbb{F}_2$ podemos garantizar una distancia mínima de al menos $2\tau + 1$. Por tanto, para estos casos doblamos la capacidad correctora de los códigos de Goppa. Esta es la principal causa por la que usualmente se utilizan los códigos de Goppa sobre el cuerpo $\mathbb{F}_2$.

Demostración.

Teorema 2.45. *Si el código de Goppa $\Gamma(L, g)$ se define sobre $\mathbb{F}_2$, con $L = (\alpha_0, \ldots, \alpha_{n-1}) \in \mathbb{F}_{2^m}^n$ y el polinomio de Goppa $g(x) \in \mathbb{F}_{2^m}[x]$ de grado τ es separable sobre $\mathbb{F}_{2^m}$, es decir, el polinomio no tiene raíces multiples, entonces la distancia mínima del código $d \geq 2\tau + 1$.*

c Si $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1}) \in \mathbb{F}_2^n$ es una palabra código con peso $w > 0$ en el código de Goppa $\Gamma(L, g)$, entonces $c_{i_1} = c_{i_2} = \cdots = c_{i_w} = 1$ con $0 \leq i_1 < i_2 < \cdots < i_w \leq n-1$ con todos los demás $c_i = 0$. Si $L = \{\alpha_0, \alpha_1, \ldots, \alpha_{n-1}\} \subseteq \mathbb{F}_{2^m}$, definimos

$$f(z) = \prod_{j=1}^{w}(z - \alpha_{i_j}) \in \mathbb{F}_{2^m}[z].$$

$$0 = f(z)\sum_{i=0}^{n-1} c_i \frac{g(z) - g(\alpha_i)}{z - \alpha_i}\frac{1}{g(\alpha_i)} = f(z)\sum_{j=1}^{w}\frac{g(z) - g(\alpha_{i_j})}{z - \alpha_{i_j}}\frac{1}{g(\alpha_{i_j})}$$
$$= \sum_{j=1}^{w}(g(z) - g(\alpha_{i_j}))\frac{1}{g(\alpha_{i_j})}\prod_{\substack{h=1\\h\neq j}}^{w}(z - \alpha_{i_h}).$$

Tomando el último polinomio módulo g obtenemos

$$0 \equiv -\sum_{j=1}^{w}\prod_{\substack{h=1\\h\neq j}}^{w}(z - \alpha_{i_h}) \equiv -f'(z) \pmod{g(z)}.$$

Por lo tanto, g divide a $f'(z)$. Dado que estamos en característica 2, $f'(z)$ contiene potencias pares y, por lo tanto, es el cuadrado de un polinomio en $\mathbb{F}_{2^m}[z]$. Como g es separable, no contiene raíces múltiples y así $g^2 \mid f'(z)$. Como consecuencia,

$$w - 1 \geq \deg(f'(z)) \geq 2\tau.$$

Es decir, cualquier palabra de código no nula tiene un peso de al menos $2\tau + 1$.

□

Ejemplo 2.46. *Sea $\mathbb{F}_{2^4} = \mathbb{F}_2(\alpha)$, donde α una raíz de $p(x) = x^4 + x + 1$. Por lo tanto, α tiene orden multiplicativo 15, es decir, $\mathbb{F}_{2^4}^* = \langle\alpha\rangle = \{1, \alpha, \ldots, \alpha^{14}\}$. En consecuencia, podemos representar los elementos de $\mathbb{F}_{2^4}$ como:*

$$
\begin{array}{rcccccl}
0 = & & & & & = (0,0,0,0), & \alpha^7 = 1 + \alpha + \alpha^3 = (1,1,0,1),\\
1 = & 1 & & & & = (1,0,0,0), & \alpha^8 = 1 + \alpha^2 = (1,0,1,0),\\
\alpha = & & \alpha & & & = (0,1,0,0), & \alpha^9 = \alpha + \alpha^3 = (0,1,0,1),\\
\alpha^2 = & & & \alpha^2 & & = (0,0,1,0), & \alpha^{10} = 1 + \alpha + \alpha^2 = (1,1,1,0),\\
\alpha^3 = & & & & \alpha^3 & = (0,0,0,1), & \alpha^{11} = \alpha + \alpha^2 + \alpha^3 = (0,1,1,1),\\
\alpha^4 = & 1 & + \alpha & & & = (1,1,0,0), & \alpha^{12} = 1 + \alpha + \alpha^2 + \alpha^3 = (1,1,1,1),\\
\alpha^5 = & & \alpha & + \alpha^2 & & = (0,1,1,0), & \alpha^{13} = 1 + \alpha^2 + \alpha^3 = (1,0,1,1),\\
\alpha^6 = & & & \alpha^2 & + \alpha^3 & = (0,0,1,1), & \alpha^{14} = 1 + \alpha^3 = (1,0,0,1).
\end{array}
$$

Consideremos el código de Goppa $\Gamma(L, g(z))$ *definido por el polinomio* $g(z) = (z+\alpha)(z+\alpha^{14}) = z^2 + \alpha^7 z + 1$ *y* $L = \{\alpha^i | 2 \leq i \leq 13\}$. *Ahora, para encontrar la matriz de control de paridad de* $\Gamma(L, g(z))$, *necesitamos calcular* $g(\alpha^2)^{-1} = (\alpha^4 + \alpha^9 + 1)^{-1} = ((1+\alpha) + (\alpha + \alpha^3) + 1)^{-1} = (\alpha^3)^{-1} = \alpha^{12}$, *y de manera similar, otros elementos para calcular* H *como se describió anteriormente.*

$$
H = \begin{pmatrix}
\alpha^{12} & \alpha^6 & \alpha^6 & \alpha & \alpha^{11} & 1 & \alpha^{14} & \alpha^8 & \alpha^{11} & \alpha^{14} & \alpha^{12} & \alpha \\
\alpha^{14} & \alpha^9 & \alpha^{10} & \alpha^6 & \alpha^2 & \alpha^7 & \alpha^7 & \alpha^2 & \alpha^6 & \alpha^{10} & \alpha^9 & \alpha^{14}
\end{pmatrix}.
$$

Las entradas pueden observarse como vectores binarios de longitud 4 utilizando la tabla descrita anteriormente. Esto se representa como:

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

Entonces, el núcleo de H *produce la matriz generadora* G *de* $\Gamma(L, g(z))$. *Realizando el*

cálculo se tiene que:

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

Ejemplo 2.47. *Sea $\mathbb{F}_{2^3}$ y tome $g(x) = \alpha^3 + x + x^2$, donde α es un elemento primitivo de $\mathbb{F}_8$, que satisface $\alpha^3 + \alpha + 1 = 0$. Sea $L = \mathbb{F}_8$, entonces definimos el código*

$$\Gamma(L, g) = \{c \in \mathbb{F}_2^8 : Hc = 0\}$$

. Los elementos de $\mathbb{F}_8$ son

$$\begin{aligned} 0 &= && && = (0,0,0), \\ 1 &= 1 && && = (1,0,0), \\ \alpha &= && \alpha && = (0,1,0), \\ \alpha^2 &= && && \alpha^2 = (0,0,1), \\ \alpha^3 &= 1 && +\alpha && = (1,1,0), \\ \alpha^4 &= && \alpha && +\alpha^2 = (0,1,1), \\ \alpha^5 &= 1 && +\alpha && +\alpha^2 = (1,1,1), \\ \alpha^6 &= 1 && && +\alpha^2 = (1,0,1), \end{aligned}$$

Así, la matriz de control de paridad del código es

$$H = \begin{pmatrix} \alpha^4 & \alpha^4 & \alpha & 1 & \alpha & \alpha^2 & \alpha^2 & 1 \\ 0 & \alpha^4 & \alpha^2 & \alpha^2 & \alpha^4 & \alpha^6 & 1 & \alpha^6 \end{pmatrix}.$$

Reemplazando cada entrada en H por el vector columna en $\mathbb{F}_2^3$ obtenemos:

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

Aplicando la reducción por filas, obtenemos la siguiente matriz de control de paridad en su forma estándar:

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

Así, la matriz generadora del código $[8, 2, 5]$ de Goppa es igual a:

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

2.5.1.1. Codificación de códigos de Goppa usando SageMath. Para realizar la construcción de un código de Goppa en SageMath tenemos la función:

```
In: codes.GoppaCode(g, L)
```

Dónde g corresponde al polinomio generador de Goppa sobre $\mathbb{F}_{q^m}$, y L es un conjunto de elementos de $\mathbb{F}_{q^m}$ que no son raices del polinomio y cuya cardinalidad es la longitud del código.

Para codificar un mensaje con códigos de Goppa en $Sage$ se usa la funcion

```
In: E=codes.encoders.GoppaCodeEncoder(C)
```

para crear el codificador para el código C de Goppa y la función

```
In: E.encode(m)
```

para codificar el mensaje m usando el codificador E creado con la función anterior.

Adicionalmente para estos códigos podemos usar las funciones para hallar la distancia mínima, matriz de contol de paridad y matriz generadora que se han mencionado previamente para códigos lineales en general. Además, la función,

```
In: C.distance_bound()
```

calcula la cota inferior para la distancia mínima del código de Goppa dada en el item 3 del Teorema 2.44.

Veamos el funcionamiento en $Sage$ de estas funciones usando los ejemplos 2.46 y 2.47 :

[3]:
```
F.<alpha> = GF(2^4, modulus=x^4 + x + 1)
```

[4]:
```
R.<z>=F[]
g= (z + alpha)*(z + alpha^14)
L=[a for a in F.list() if g(a)!=0 and a!=0 and a!=1]
```

[5]:
```
L
```

[5]:
```
[alpha^2,
 alpha^3,
 alpha + 1,
 alpha^2 + alpha,
 alpha^3 + alpha^2,
 alpha^3 + alpha + 1,
 alpha^2 + 1,
 alpha^3 + alpha,
 alpha^2 + alpha + 1,
```

```
       alpha^3 + alpha^2 + alpha,
       alpha^3 + alpha^2 + alpha + 1,
       alpha^3 + alpha^2 + 1]
```

[6]:
```
C=codes.GoppaCode(g,L)
```

[7]:
```
C
```

[7]:
```
[12, 4] Goppa code over GF(2)
```

[8]:
```
C.distance_bound()
```

[8]:
```
3
```

[9]:
```
C.minimum_distance()
```

[9]:
```
5
```

[10]:
```
C.parity_check_matrix()
```

[10]:
```
[1 0 0 0 0 1 1 1 0 1 1 0]
[1 0 0 1 1 0 0 0 1 0 1 1]
[1 1 1 0 1 0 0 1 1 0 1 0]
[1 1 1 0 1 0 1 0 1 1 1 0]
[1 0 1 0 0 1 1 0 0 1 0 1]
[0 1 1 0 0 1 1 0 0 1 1 0]
[0 0 1 1 1 0 0 1 1 1 0 0]
[1 1 0 1 0 1 1 0 1 0 1 1]
```

[11]:
```
C.generator_matrix()
```

[11]:
```
[1 0 0 0 1 1 0 1 1 1 0 1]
[0 1 0 0 1 1 1 1 0 0 1 0]
[0 0 1 0 1 0 1 1 1 0 0 0]
[0 0 0 1 1 1 0 0 0 0 1 1]
```

[12]:
```
F2.<alpha> = GF(2^3, modulus=x^3 + x + 1)
```

[13]:
```
R.<z>=F2[]
g2= alpha^3 + z + z^2
L2=[a for a in F2.list() if g2(a)!=0]
```

[14]:
```
L2
```

[14]:
```
[0,
 alpha,
 alpha^2,
 alpha + 1,
 alpha^2 + alpha,
 alpha^2 + alpha + 1,
 alpha^2 + 1,
 1]
```

[15]:
```
C2=codes.GoppaCode(g2,L2)
```

[16]:
```
C2
```

[16]:
```
[8, 2] Goppa code over GF(2)
```

[17]:
```
E=codes.encoders.GoppaCodeEncoder(C2)
```

[18]:
```
E
```

[18]:
```
Encoder for [8, 2] Goppa code over GF(2)
```

[19]:
```
palabra = vector(GF(2), (0, 1))
c = E.encode(palabra)
c
```

[19]:
```
(0, 0, 1, 1, 0, 1, 1, 1)
```

[20]:
```
c in C2
```

```
[20]: True
```

```
[21]: C2.minimum_distance()
```

```
[21]: 5
```

```
[22]: C2.parity_check_matrix()
```

```
[22]: [0 0 1 0 0 0 1 0]
      [1 1 0 1 0 0 0 1]
      [1 0 0 0 1 1 0 1]
      [0 0 0 0 1 1 1 0]
      [0 0 0 1 0 0 0 1]
      [0 1 1 1 1 0 1 1]
```

```
[23]: C2.generator_matrix()
```

```
[23]: [1 1 0 1 1 1 0 1]
      [0 0 1 1 0 1 1 1]
```

Veamos un ejemplo con párametros mas grandes.

```
[4]: F.<alpha> = GF(3^3, modulus=2*x^3 + x +1)
```

```
[5]: R.<z>=F[]
     g= (z + alpha)*(z + alpha^14)
     L=[a for a in F.list() if g(a)!=0]
```

```
[6]: C=codes.GoppaCode(g,L)
```

```
[7]: C
```

```
[7]: [26, 20] Goppa code over GF(3)
```

```
[8]: H=C.parity_check_matrix()
```

```
[9]: H
```

```
[9]: [1 1 2 0 0 1 2 2 0 1 1 0 2 0 1 2 1 1 1 1 0 0 1 2 0 1]
     [1 1 0 0 2 2 1 0 1 1 1 1 0 1 0 0 1 1 0 2 1 0 2 1 2 2]
     [2 0 2 1 1 2 0 1 1 1 1 0 2 0 0 1 0 2 0 1 1 1 1 0 1 2]
     [0 2 1 2 1 1 2 1 2 1 0 1 1 2 2 0 1 2 1 0 2 2 1 1 0 1]
     [0 2 1 0 1 1 0 1 2 2 0 2 0 1 2 2 2 0 2 2 2 1 0 2 0 2]
     [0 1 2 0 0 0 0 2 1 2 2 1 1 0 0 1 0 1 0 1 0 0 1 1 2 2]
```

```
[10]: C.generator_matrix()
```

```
[10]: 20 x 26 dense matrix over Finite Field of size 3 (use the '.
      ↪str()' method to see
      the entries)
```

```
[12]: E=codes.encoders.GoppaCodeEncoder(C)
```

```
[14]: palabra = vector(GF(3),␣
      ↪(0,1,1,0,2,2,0,0,0,1,0,1,1,1,1,1,0,0,2,2))
      c = E.encode(palabra)
      c
```

```
[14]: (0, 1, 1, 0, 2, 2, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 2, 1,␣
      ↪2, 0, 2, 0, 0, 2)
```

2.5.1. Decodificación por síndromes para códigos de Goppa

Sean $c = (c_1, c_2, \ldots, c_n) \in \Gamma(L, g)$ la palabra código enviada y $y = (y_1, y_2, \ldots, y_n)$ el vector recibido con t errores, donde $2t + 1 \leq d$ (es decir, esta dentro de la capacidad correctora del código), por lo tanto,

$$y = c + e = (c_1, c_2, \ldots, c_n) + (e_1, e_2, \ldots, e_n)$$

con $e_i \neq 0$ en t coordenadas, esto es $w(e) = t$.

Para nuestro algoritmo de decodificación necesitamos:

- Localizar las posiciones de error, esto es, encontrar $B = \{i : 1 \leq i \leq n \text{ y } e_i \neq 0\}$.
- Encontrar los valores correspondientes de los errores (valores de e_i para $i \in B$).

Para encontrar estos, se tiene la siguiente definición.

Definición 2.48. *Sea $B = \{i : 1 \leq i \leq n \, y \, e_i \neq 0\}$. Definimos los polinomios **localizador de errores** $L(z)$ y **evaluador de errores** $E(z)$:*

$$L(z) = \prod_{i \in B}(z - \alpha_i),$$
$$E(z) = \sum_{i \in B} e_i \prod_{j \in B, j \neq i} (z - \alpha_j).$$

Estos polinomios cumplen las siguientes propiedades:

Lema 2.49. *Sean $L(z)$ y $E(z)$ como en la definición. Entonces*

1. *deg$(L(z)) = t$.*
2. *deg$(E(z)) \leq t - 1$.*
3. *mcd$(L(z), E(z)) = 1$.*
4. *Si $f_i(z) = \prod_{j \in B, j \neq i}(z - \alpha_j)$, entonces*
   a) *$L'(z) = \sum_{i \in B} \prod_{j \in B, j \neq i}(z - \alpha_j) = \sum_{i \in B} f_i(z)$, donde $L'(z)$ es la derivada de $L(z)$.*
   b) *$E(z) = \sum_{i \in B} e_i f_i(z)$.*
   c) *$f_i(\alpha_k) = \begin{cases} 0 & \text{si } k \neq i \\ \prod_{j \in B, j \neq i}(\alpha_k - \alpha_j) & \text{si } k = i \end{cases}$.*
5. *Si el código es binario, entonces $E(z) = L'(z)$.*

Demostración. 1., 2. y 4. se siguen de la definición de $L(z)$ y $E(z)$.

3. Basta comprobar que $L(z)$ y $E(z)$ no tienen las mismas raíces. Es claro que $L(\alpha_i) = 0$ si y

solo si $i \in B$. Por otro lado,

$$E(\alpha_i) = \sum_{j \in E} e_j \prod_{j' \in E, j' \neq j} (\alpha_i - \alpha_{j'}) = e_i \prod_{j' \in E, j' \neq i} (\alpha_i - \alpha_{j'}) \neq 0.$$

Por lo que podemos afirmar que mcd$(L(z), E(z)) = 1$.

5. Como el código es binario, entonces $e_i \neq 0$ implica $e_i = 1$ y por tanto $E(z) = \sum_{i \in B} e_i f_i(z) = \sum_{i \in B} f_i(z) = L'(z)$. □

El siguiente teorema muestra la utilidad de los polinomios $L(z)$ y $E(z)$.

Teorema 2.50. *Si conocemos $L(z)$ y $E(z)$ entonces podemos deducir las posiciones del vector error y sus respectivos valores. Esto es,*

1. $e_i \neq 0$ *si y solo si* $i \in B$ *si y solo si* α_i *es una raíz de* $L(z)$.
2. $e_i = \dfrac{E(\alpha_i)}{L'(\alpha_i)}$ *si* $i \in B$.

Demostración. 1. Se sigue de la definición de $L(z)$.

2. $\dfrac{E(\alpha_i)}{L'(\alpha_i)} = \dfrac{\sum_{k \in B} e_k f_k(\alpha_i)}{\sum_{k \in B} f_k(\alpha_i)} = \dfrac{e_i \prod_{j \in B, j \neq i} (\alpha_i - \alpha_j)}{\prod_{j \in B, j \neq i} (\alpha_i - \alpha_j)} = e_i.$ □

Observe que el problema de la decodificación de códigos de Goppa ahora está enfocado en encontrar los polinomios $L(z)$ y $E(z)$. Para esto introducimos la siguiente definición.

Definición 2.51. *El Síndrome del vector recibido $y = (y_1, y_2, \ldots, y_n)$ para el código de Goppa $\Gamma(L, g)$, denotado por $S(z)$, como:*

$$S(z) = \sum_{i=1}^{n} \frac{y_i}{z - \alpha_i} = \sum_{i=1}^{n} \frac{c_i}{z - \alpha_i} + \sum_{i \in B} \frac{e_i}{z - \alpha_i} \equiv \sum_{i \in B} \frac{e_i}{z - \alpha_i} \pmod{g(z)}$$

ya que $\sum_{i=1}^{n} \frac{c_i}{z - \alpha_i} \equiv 0 \pmod{g(z)}$.

El siguiente resultado nos proporciona una relación entre $L(z)$, $E(z)$ y $S(z)$.

Teorema 2.52. *Sean $L(z)$, $E(z)$ y $S(z)$ como antes. Entonces*

$$L(z)S(z) \equiv E(z) \pmod{g(z)}.$$

Demostración.

$$\begin{aligned} L(z)S(z) &\equiv \prod_{i\in B}(z-\alpha_i)\sum_{i\in B}\frac{e_i}{z-\alpha_i} \pmod{g(z)} \\ &\equiv \sum_{i\in B} e_i \prod_{j\in B}(z-\alpha_j)\frac{1}{z-\alpha_i} \pmod{g(z)} \\ &\equiv \sum_{i\in B} e_i \prod_{j\in B, j\neq i}(z-\alpha_j) \pmod{g(z)} \\ &= E(z) \end{aligned}$$

□

Algoritmo del sindrome.

Para corregir t errores con $2t+1 \leq d$ en un código de Goppa $\Gamma(L,g)$, donde $deg(g(z)) = \tau$, el algoritmo para corrección de errores usando síndrome sigue los siguientes pasos:

1. Calcular el síndrome del vector recibido $y = (y_1, y_2, \ldots, y_n)$:

$$S(z) = \sum_{i=1}^{n} \frac{y_i}{z-\alpha_i}.$$

2. Resolver el sistema:

$$L(z)S(z) \equiv E(z) \pmod{g(z)},$$

escribiendo el polinomio localizador de errores como $L(z) = L_0 + L_1 z + \ldots + L_{t-1}z^{t-1} + z^t$ y el polinomio evaluador de errores como $E(z) = E_0 + E_1 z + \ldots + E_{t-1}z^{t-1}$, necesitamos resolver el sistema de τ ecuaciones y $2t$ incógnitas.

3. Determinar el conjunto de ubicaciones de errores:

$$B = \{i : 1 \leq i \leq n \text{ y } L(\alpha_i) = 0\}.$$

4. Calcular los valores $e_i \neq 0$ en el vector error $e = (e_1, e_2, \ldots, e_n)$:

$$e_i = \frac{E(\alpha_i)}{L'(\alpha_i)} \quad \text{para todo } i \in B.$$

5. Calcular la palabra código enviada:

$$c = y - e.$$

Ejemplo 2.53. *Sea $\mathbb{F}_{3^2}$ el cuerpo correspondiente al polinomio primitivo $x^2 - x - 1$ sobre el cuerpo base $\mathbb{F}_3$, si α es una de sus raíces, entonces $\alpha^2 = 1 + \alpha$ y, por tanto, podemos escribir $\mathbb{F}_9 = \{a + b\alpha : a, b \in \mathbb{F}_3 = \{0, 1, 2\}\}$. De esta forma tenemos:*

$$\begin{aligned}
0 &= 0 \\
1 &= 1 \\
\alpha &= \alpha \\
\alpha^2 &= 1 + \alpha \\
\alpha^3 &= 1 + 2\alpha \\
\alpha^4 &= 2 \\
\alpha^5 &= 2\alpha \\
\alpha^6 &= 2 + 2\alpha \\
\alpha^7 &= 2 + \alpha
\end{aligned}$$

Consideremos el código de Goppa $\Gamma(L, g)$ con polinomio generador $g(z) = z(z - \alpha^7) = z^2 + \alpha^3 z$ y $L = (\alpha^i)_{i=0}^6 = (1, \alpha^1, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6)$.

Supongamos que el vector $y = (0, 0, 0, 0, 2, 2, 0) = (y_0, y_1, \ldots, y_6)$ es recibido. Dado que $d = 3$, entonces se sabe que este código puede corregir como máximo un error. Así, si suponemos que un error se ha cometido en la transmisión, entonces el grado de $L(z)$ es 1 y el grado de $E(z)$ es 0. Por lo tanto, si aplicamos el algoritmo para corrección de errores usando el síndrome.

1. Calculamos el síndrome

$$S(z) = \sum_{i=0}^{6} \frac{y_i}{z - \alpha^i} = \frac{2}{z - \alpha^4} + \frac{2}{z - \alpha^5}$$

Nuestro objetivo es encontrar dos polinomios b_1 *y* $b_2 \in \mathbb{F}_{3^2}[z]$ *tales que:*

$$\frac{2}{z - \alpha^4} \equiv b_1 \pmod{g(z)} \qquad y \qquad \frac{2}{z - \alpha^5} \equiv b_2 \pmod{g(z)}$$

Por el algoritmo de la división, encontramos que

$$z^2 + \alpha^3 z = (z - \alpha^4)(z + \alpha^5) + \alpha$$

Multiplicando ambos lados por $(\alpha)^{-1} = \alpha^7$ *obtenemos*

$$\alpha^7 g(z) = (z - \alpha^4)(\alpha^7 z - 1) + 1$$

Por lo tanto,

$$\frac{2}{z - \alpha^4} \equiv \alpha^7 z - 1 \pmod{g(z)}$$

.

De manera similar, calculamos

$$z^2 + \alpha^3 z = (z - \alpha^5)(z + \alpha^2) + \alpha^7$$

$$\alpha g(z) = (z - \alpha^4)(\alpha z + \alpha^3) + 1$$

Entonces,

$$\frac{2}{z - \alpha^5} \equiv \alpha z + \alpha^3 \pmod{g(z)},$$

en consecuencia,

$$\begin{aligned} S(z) &= \frac{2}{z-\alpha^4} + \frac{2}{z-\alpha^5} \\ &\equiv (\alpha z + \alpha^3) + (\alpha^7 z - 1) \pmod{g(z)} \\ &\equiv z(\alpha + \alpha^7) + \alpha^3 - 1 \pmod{g(z)} \\ &\equiv z(2\alpha - 1) - \alpha \pmod{g(z)} \\ &\equiv \alpha^6 z - \alpha \pmod{g(z)} \end{aligned}$$

2. Sustituyendo $L(z) = L_0 + z$, obtenemos:

$$L(z)S(z) = (L_0 + z)(\alpha^6 z - \alpha) = -L_0\alpha + (L_0\alpha^6 - \alpha)z + \alpha^6 z^2$$

$$\equiv -L_0\alpha + (L_0\alpha^6 - 2\alpha)z \pmod{g(z)}$$

$$\equiv L_0\alpha + (L_0\alpha^6 + \alpha)z \pmod{g(z)}$$

Por otro lado, como $E(z) = E_0$, obtenemos el sistema 2×2, comparando coeficientes de $L(z)S(z) \equiv E(z) \pmod{g(z)}$. Esto es, $L_0\alpha + (L_0\alpha^6 + \alpha)z \equiv E_0 \pmod{g(z)}$, luego el sistema es:

$$\begin{cases} E_0 = -L_0\alpha \\ 0 = L_0\alpha^6 + \alpha \end{cases}.$$

Por lo tanto, $L_0 = -\alpha^3$. Sustituyendo este valor en la primera ecuación obtenemos

$$E_0 = -\alpha(-\alpha^3) = \alpha^4.$$

En consecuencia, la solución única a este sistema es $L_0 = -\alpha^3$, $E_0 = \alpha^4$,
así $L(z) = z - \alpha^3$ y $E(z) = \alpha^4$.

3. La única raíz del polinomio $L(z)$ es α^3. Con lo cual, el conjunto de posiciones de error es $B = \{i | L(\alpha_i) = 0\} = \{3\}$.

4. El valor de la coordenada e_3 en el vector error es

$$e_3 = \frac{E(\alpha^3)}{L'(\alpha^3)} = \frac{\alpha^4}{1} = 2$$

5. La palabra de código enviada debe haber sido

$$c = y - e = (0,0,0,0,2,2,0) - (0,0,0,2,0,0,0) = (0,0,0,1,2,2,0).$$

2.5.2.1. Decodificación de códigos de Goppa usando SageMath. En SageMath podemos realizar la decodificación de un mensaje, codificado usando un codigo Goppa, usando el decodificador syndrome en la función

```
In: C.decoder(decoder_name, *args)
```

mencionada anteriormente, de la siguiente manera:

```
[16]: E=codes.encoders.GoppaCodeEncoder(C)
```

```
[17]: palabra = vector(GF(2), (0, 1))
      c = E.encode(palabra)
      c
```

```
[17]: (0, 1, 1, 1, 1, 1, 1, 0)
```

```
[18]: C.decoders_available()
```

```
[18]: ['InformationSet', 'NearestNeighbor', 'Syndrome']
```

```
[19]: y=C.decode_to_message(c,'Syndrome',1)
      y
```

```
[19]: (0, 1)
```

```
[3]: F.<alpha> = GF(3^2, modulus=x^2 - x -1)
```

```
[4]: R.<z>=F[]
     g= z^2 + alpha^3*z
     L=[a for a in F.list() if g(a)!=0 and a!=0]

[5]: C=codes.GoppaCode(g,L)

[6]: E=codes.encoders.GoppaCodeEncoder(C)

[7]: palabra = vector(GF(3), (0, 0,1))
     c = E.encode(palabra)
     c

[7]: (0, 0, 1, 2, 2, 0, 0)

[8]: e=vector(GF(3),(0, 0, 2, 0, 0, 0, 0))

[9]: y=c+e

[10]: y

[10]: (0, 0, 0, 2, 2, 0, 0)

[11]: m=C.decode_to_message(y,'Syndrome',1)
      m

[11]: (0, 0, 1)
```

2.5.2. Algoritmo de Patterson para corrección de errores

El algoritmo de Patterson es un algoritmo de decodificación, propuesto en 1975 por Nicholsa J. Patterson Patterson, 1975, que solo funciona para códigos binarios de Goppa. Es decir, un código de Goppa $\Gamma(L, g)$, donde $g(z)$ es un polinomio con coeficientes en $\mathbb{F}_{2^m}$ y $L = (\alpha_1, \alpha_2, \ldots, \alpha_n) \in \mathbb{F}_{2^m}^n$. Por el Teorema 2.45, la distancia minima d del código de Goppa binario $\Gamma(L, g)$ satisface que $d \geq 2\tau + 1$, donde $\tau = \deg(g(z))$, con lo cual el código puede corregir al menos τ errorres.

El algoritmo usa el síndrome del vector recibido $y = (y_1, y_2, \ldots, y_n)$, dado en la Definición 2.51 y usando el hecho que el código de Goppa es binario:

$$S(z) = \sum_{i=1}^{n} \frac{y_i}{z + \alpha_i} = \sum_{i=1}^{n} \frac{e_i}{z + \alpha_i} = \frac{\displaystyle\sum_{i=1}^{n} e_i \prod_{j \neq i} (z + \alpha_j)^{e_i}}{\displaystyle\prod_{j=1}^{n} (z + \alpha_j)^{e_i}}.$$

Note que el denominador es el polinomio localizador $L(z)$ y el númerador es su derivada $L'(z)$. En consecuencia,

$$S(z) = \frac{L'(z)}{L(z)}.$$

Ahora, utilizando esta igualdad, veremos como calcular el polinomio localizador de errores $L(z)$. Como el cuerpo tiene característica 2, entonces $(x + y)^2 = x^2 + y^2$ y si a y b están en $\mathbb{F}_{2^m}$, entonces existen $\sqrt{a}$ y $\sqrt{b}$ tal que $ax^2 + by^2 = (\sqrt{a}x + \sqrt{b}y)^2$ (ver Teorema 1.27). Entonces el polinomio localizador de errores puede dividirse en potencias pares e impares de z y representarse de la forma $L(z) = u(z)^2 + zv(z)^2$ y, por lo tanto, $L'(z) = v(z)^2$. Así

$$S(z) = \frac{L'(z)}{L(z)} = \frac{v(z)^2}{u(z)^2 + zv(z)^2}$$

$$\begin{aligned}
&\Rightarrow (u(z)^2 + zv(z)^2)S(z) \equiv v(z)^2 \pmod{g(z)} \\
&\Rightarrow (u(z)^2 + zv(z)^2) \equiv \frac{v(z)^2}{S(z)} \pmod{g(z)} \\
&\Rightarrow u(z)^2 \equiv \frac{v(z)^2}{S(z)} + zv(z)^2 \pmod{g(z)} \\
&\Rightarrow u(z)^2 \equiv \left(\frac{1}{S(z)} + z\right) v(z)^2 \pmod{g(z)} \\
&\Rightarrow u(z) = v(z)\sqrt{\frac{1}{S(z)} + z} \pmod{g(z)}
\end{aligned}$$

Así, si denotamos por $T(z) = \dfrac{1}{S(z)}$, es decir el inverso de $S(z)$, calculamos el polinomio $P(z)$ tal que $P^2(z) \equiv \sqrt{T(z)+z} \pmod{g(z)}$, y calculamos los polinomios $u(z)$ y $v(z)$, de grado mínimo, tal que $P(z)v(z) \equiv u(z) \pmod{g(z)}$, entonces $L(z) = u(z)^2 + zv(z)^2$.

Ahora, como las raíces de $L(z)$ determinan las posiciones error y el código de Goppa es binario, tenemos que las posiciones error son 1 en dichas coordenadas del vector error. Esto se debe a que el polinomio evaluador de errores es $E(z) = L'(z)$, visto en el item 5 del Lema 2.49, con lo cual al calcular las posiciones error reemplazamos estas coordendas por 1 en el vector error y las demás son 0, ya que $e_i = \frac{E(\alpha_i)}{L'(\alpha_i)} = \frac{L'(\alpha_i)}{L'(\alpha_i)} = 1 \quad$ para todo $i \in B$.

El algoritmo de Patterson

Para corregir t errores con $t \leq \tau$ en un código de Goppa binario $\Gamma(L, g)$, donde $deg(g(z)) = \tau$, el algoritmo de Patterson sigue los siguientes pasos:

1. Calcular el síndrome del vector recibido $y = (y_1, y_2, \ldots, y_n)$:

$$S(z) = \sum_{i=1}^{n} \frac{y_i}{z - \alpha_i}.$$

2. Hallar $T(z)$ tal que que $S(z)T(z) \equiv 1 \pmod{g(z)}$.

3. Calcular el polinomio $P(z)$ tal que $P^2(z) \equiv T(z) + z \pmod{g(z)}$.

4. Calcular $u(z)$ y $v(z)$ de grado mínimo tal que $P(z)v(z) \equiv u(z) \pmod{g(z)}$.
   Así, $L(z) = u(z)^2 + zv(z)^2$.

5. Encontrar las posiciones de error, es decir, las raíces de $L(z)$.

$$B = \{i : 1 \leq i \leq n \text{ y } L(\alpha_i) = 0\}.$$

6. Encontrar el vector de error $e = (e_1, e_2, \ldots, e_n)$ donde

$$e_i = \begin{cases} 0 & \text{si } i \notin B \\ 1 & \text{si } i \in B. \end{cases}$$

7. Calcular la palabra código enviada: $c = y - e$.

Después de corregir posibles errores en una palabra código, se puede encontrar el mensaje enviado ya que $(m_1, m_2, \ldots, m_k)G = (c_1, c_2, \ldots, c_n)$, lo que equivale a $G^T \cdot \begin{bmatrix} m_1 \\ \vdots \\ m_k \end{bmatrix} = \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix}$, para encontrar el vector de mensaje $(m_1, m_2, \ldots, m_k)$, se reduce el sistema anterior a

$$\left[\begin{array}{c|c} G^T & \begin{matrix} c_1 \\ \vdots \\ c_n \end{matrix} \end{array}\right] \sim \cdots \sim \left[\begin{array}{c|c} \mathrm{I}_k & \begin{matrix} \mathrm{m}_1 \\ \vdots \\ \mathrm{m}_k \end{matrix} \\ \hline P & \end{array}\right].$$

Ejemplo 2.54. *Sea $\mathbb{F}_{2^4} = \mathbb{F}_2(\alpha)$, donde α una raíz de $x^4 + x + 1$. Por lo tanto, α tiene orden multiplicativo 15, es decir, $\mathbb{F}_{2^4}^* = \langle\alpha\rangle = \{1, \alpha, \ldots, \alpha^{14}\}$. En consecuencia, podemos representar los elementos de $\mathbb{F}_{2^4}$ como:*

$$\begin{array}{rcccl|rcccl}
0 = & & & & = (0,0,0,0), & \alpha^7 = & 1 & +\alpha & +\alpha^3 & = (1,1,0,1), \\
1 = & 1 & & & = (1,0,0,0), & \alpha^8 = & 1 & +\alpha^2 & & = (1,0,1,0), \\
\alpha = & & \alpha & & = (0,1,0,0), & \alpha^9 = & & \alpha & +\alpha^3 & = (0,1,0,1), \\
\alpha^2 = & & & \alpha^2 & = (0,0,1,0), & \alpha^{10} = & 1 & +\alpha & +\alpha^2 & = (1,1,1,0), \\
\alpha^3 = & & & & \alpha^3 = (0,0,0,1), & \alpha^{11} = & & \alpha & +\alpha^2 +\alpha^3 & = (0,1,1,1), \\
\alpha^4 = & 1 & +\alpha & & = (1,1,0,0), & \alpha^{12} = & 1 & +\alpha & +\alpha^2 +\alpha^3 & = (1,1,1,1), \\
\alpha^5 = & & \alpha & +\alpha^2 & = (0,1,1,0), & \alpha^{13} = & 1 & & +\alpha^2 +\alpha^3 & = (1,0,1,1), \\
\alpha^6 = & & & \alpha^2 & +\alpha^3 = (0,0,1,1), & \alpha^{14} = & 1 & & +\alpha^3 & = (1,0,0,1).
\end{array}$$

Consideremos el código de Goppa $\Gamma(L, g(z))$ *definido por el polinomio* $g(z) = z^3 + \alpha^2 z + \alpha$ *y* $L = (0, \alpha, \alpha^2, \ldots, \alpha^{14}, 1)$. *Ahora, para encontrar la matriz de control de paridad de* $\Gamma(L, g(z))$, *necesitamos calcular* $g(\alpha^i)^{-1}$ *para cada* α_i, *como lo mostramos en la siguiente tabla:*

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
α_i	0	α	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}	1
$g(\alpha^i)$	α	α	α^{13}	α^{11}	1	α^3	α^{12}	α^2	α^{12}	α^4	α^6	α^{10}	α^{10}	α^{14}	α^{12}	α^{10}
$g(\alpha^i)^{-1}$	α^{14}	α^{14}	α^2	α^4	1	α^{12}	α^3	α^{13}	α^3	α^{11}	α^9	α^5	α^5	α	α^3	α^5

$$H = \begin{pmatrix} \alpha^{14} & \alpha^{14} & \alpha^2 & \alpha^4 & 1 & \alpha^{12} & \alpha^3 & \alpha^{13} & \alpha^3 & \alpha^{11} & \alpha^9 & \alpha^5 & \alpha^5 & \alpha & \alpha^3 & \alpha^5 \\ 0 & 1 & \alpha^4 & \alpha^7 & \alpha^4 & \alpha^2 & \alpha^9 & \alpha^5 & \alpha^{11} & \alpha^5 & \alpha^4 & \alpha & \alpha^2 & \alpha^{14} & \alpha^2 & \alpha^5 \\ 0 & \alpha & \alpha^6 & \alpha^{10} & \alpha^8 & \alpha^7 & 1 & \alpha^{12} & \alpha^4 & \alpha^{14} & \alpha^{14} & \alpha^{12} & \alpha^{14} & \alpha^{12} & \alpha & \alpha^5 \end{pmatrix}.$$

Las entradas pueden representarse como vectores binarios de longitud 4 utilizando la tabla descrita anteriormente, es decir,

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

Entonces, el núcleo de H produce la matriz generadora G de $\Gamma(L, g(z))$. Realizando el cálculo se tiene que:

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

Supongamos que vamos a enviar el mensaje $m = (0, 1, 0, 0)$ utilizando el código $\Gamma(L, g(z))$ anterior, así el mensaje codificado es $c = (0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0)$. Suponiendo que se han cometido $t = 3$ errores, de forma que se recibe el vector $y = (0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1)$, utilicemos el algoritmo de Patterson para corregir estos errores.

1. Calculamos el síndrome del vector recibido:

$$S(z) = \sum_{i=1}^{16} \frac{y_i}{z + \alpha_i} = \frac{1}{z+\alpha} + \frac{1}{z+\alpha^4} + \frac{1}{z+\alpha^6} + \frac{1}{z+\alpha^9} + \frac{1}{z+\alpha^{12}} + \frac{1}{z+1}.$$

calculando estos sumandos tenemos que:

$$S(z) = (\alpha^{14}z^2 + z) + (z^2 + \alpha^4 z + 1) + (\alpha^3 z^2 + \alpha^9 z + \alpha^{10}) + (\alpha^{11}z^2 + \alpha^5 z + \alpha^2)$$

$$+(\alpha^5 z^2 + \alpha^2 z + \alpha) + (\alpha^5 z^2 + \alpha^5 z + \alpha^{13}).$$

$$S(z) = \alpha^{11}z^2 + \alpha^6 z + \alpha^{13}$$

2. Calculamos $T(z)$ tal que que $S(z)T(z) \equiv 1 \pmod{g(z)}$

$$T(z) = \alpha^3 z^2 + \alpha^{11} z + \alpha^6.$$

3.Calculamos el polinomio $P(z)$ tal que $P^2(z) \equiv T(z) + z = \alpha^3 z^2 + \alpha^{12} z + \alpha^6 \pmod{g(z)}$

$$P(z) = \sqrt{\alpha^3}z + \sqrt{\alpha^{12}}\sqrt{z} + \sqrt{\alpha^6}$$

Como a partir del polinomio $g(z)$ *se tiene que* $\sqrt{z} = \alpha^8(\alpha^{14}z^2 + z)$*, entonces*

$$P(z) = \alpha^{13}z^2 + \alpha^4 z + \alpha^3$$

4. Calculamos los polinomios $u(z)$ *y* $v(z)$ *de grado mínimo tal que* $P(z)v(z) \equiv u(z) \pmod{g(z)}$.

Por lo tanto, $L(z) = u(z)^2 + zv(z)^2 = (\alpha^{13}z + \alpha^6)^2 + z(\alpha^2 z + \alpha^8)^2 = \alpha^4 z^3 + \alpha^{11}z^2 + \alpha z + \alpha^{12}$.

5. Encontramos las posiciones de error, es decir, las raíces de $L(z)$*. Para esto, note que* $L(z) = (z + \alpha^{10})(z + \alpha^{13})(z + 1)$

$$B = \{i : 1 \leq i \leq n \text{ y } L(\alpha_i) = 0\} = \{11, 14, 16\}.$$

6. Encontramos el vector de error $e = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1)$.

En consecuencia, la palabra enviada es $C = y + e = (0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1) + (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1) = (0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0)$.

2.5.3. El algoritmo de Patterson usando SageMath

Para ver el funcionamiento del algoritmo de Patterson usando Sage nos basaremos en el código mostrado en Rambaut Lemus, 2021. En el cúal se define la función `decodePatterson`(y), que hace uso de las siguientes funciones: Primero, la función `descomponer_polinomio(p)` que recibe por parámetros un polinomio, y devuelve su descomposición. Esta se utiliza para definir el polinomio localizador de errores dentro del algoritmo de Patterson, ya que descompone o separa un polinomio dado en dos partes permitiendo definir el polinomio localizador de errores como $L(z) = u(z)^2 + zv(z)^2$.

Al aplicar la función `descomponer_polinomio`$(self, p)$ sobre el polinomio de Goppa $g(z)$, se obtiene $g(z) = g_0(z)^2 + zg_1(z)^2$. A partir de ello, tenemos que $\sqrt{z} = g_0(z)g_1^{-1}(z)$, ya que podemos ver que $g(z) - g_0(z)^2 = zg_1(z)^2$ y aplicando ahora módulo $g(z)$ y obteniendo el polinomio inverso de $g_1(z)^2$ tenemos que $z \equiv g_0(z)^2 g_1(z)^{-2}$.

Después, la función `algoritmo_euclides_extendido(self, other)`, la cual calcula el máximo común divisor y lo expresa como la mínima combinación. Usando esta función se define `inversa_`(p, g) la cual devuelve el polinomio de grado menor, para de esta forma calcular el $P(z)$ en el tercer paso del algoritmo de Patterson.

Para el proceso de cifrado, se crea un mensaje y se cifra mediante la multiplicación por la matriz generadora G.

Después se genera un vector de errores e y al sumar este vector con el vector recibido se obtiene el mensaje final, que fue el enviado. Y este se decodifica con `decodePatterson`(y).

Veamos un ejemplo implementando el algoritmo:

[4]:
```
# Definimos el cuerpo donde vamos a trabajar
r = 4;rango = 3;N = 2^r
K_.<a> = GF(2)
F.<a> = GF(2^r)
```

[5]:
```
# Creamos el anillo de polinomios
PR = PolynomialRing(F,'Z')
Z = PR.gen()
g = Z^3+Z+1                          # Polinomio de Goppa
L = [a^i for i in range(N)]          # Soporte del codigo
```

[6]:
```
def descomponer_polinomio(p):
    # El siguiente metodo permite descomponer un polinomio p␣
 ↪en factores irreducibles p(z) = p0 (z) + z p1 (z)
    # Entrada: Polinomio p
    Phi1 = p.parent()
```

```
    p0 = Phi1([sqrt(c) for c in p.list()[0::2]])
    p1 = Phi1([sqrt(c) for c in p.list()[1::2]])
    return (p0,p1)
```

[7]:
```
# Algoritmo de euclides extendido: Obtener MCD y los s,t que lo generan.
# -------------------
def algoritmo_euclides_extendido(self, other):
    delta = self.degree() #grado de polinomio 1
    if other.is_zero(): # si el polinomio introducido es
        ring = self.parent() #comprobamos el cuerpo en el que trabajamos
        return self, R.one(), R.zero() #mcd = mismo polinomio y devuelve un uno (s) y un cero (t) en el cuerpo que trabajamos.

    # mcd (a,b) = as+bt

    ring = self.parent() #comprobamos el cuerpo en el que trabajamos
    a = self # guardamos una copia del primer polinomio 1 (self)
    b = other # guardamos una copia del segundo polinomio (other)

    s = ring.one() # guardamos en s el uno del anillo
    t = ring.zero() # guardamos en t el cero del anillo

    resto0 = a
    resto1 = b
```

```
    while true:
        cociente,resto_auxiliar = resto0.quo_rem(resto1) # La␣
↪funcion quo_rem de Sage devuelve el cociente y el resto.␣
↪Que guardamos en Q y ring.
        resto0 = resto1
        resto1 = resto_auxiliar

        s = t
        t = s - t*cociente

        if resto1.degree() <= floor((delta-1)/2) and resto0.
↪degree() <= floor((delta)/2):
            break

    V = (resto0-a*s)//b
    coeficiente_lider = resto0.leading_coefficient() #␣
↪guardamos el coeficiente lider del resto 0

    return resto0/coeficiente_lider, s/coeficiente_lider, V/
↪coeficiente_lider
```

[8]:
```
# Funcion que calcula la inversa de un polinomio utilizando␣
↪el algoritmo de euclides de Sage
# ------------------
def inversa_g(p,g):
    # Input: Polinomios p y g
    # Output:retornar polinomio P modulo g
    (d,A,B) = xgcd(p,g)
    return A.mod(g)
```

```
# -------------------
# Funcion de decodificacion de Patterson
# -------------------
def decodePatterson(y):
    # Calculamos primero el vector alpha con los elementos primitivos.
    alpha = vector(H*y)

    # Consideramos nuestras matrices T,Y,Z definidas así como nuestro polinomio irreducible g

    polinomioS = PR(0) # Inicializa como el polinomio 0 del anillo
    for i in range(len(alpha)):
        polinomioS = polinomioS + alpha[i]*(Z^(len(alpha)-i-1)) # Lo vamos rellenando con los alpha

    vector_g = descomponer_polinomio(g) # Guardamos en vector_g el par de polinomios irreducibles
    w = ((vector_g[0])*inversa_g(vector_g[1],g)).mod(g)
    vector_t = descomponer_polinomio(inversa_g(polinomioS,g) + Z)

    R = (vector_t[0]+(w)*(vector_t[1])).mod(g)

    (u,aux,v) = algoritmo_euclides_extendido(g,R)

    # Definimos el polinomio sigma
```

```
    sigma = u^2+Z*(v^2)

    # Vamos comprobando uno a uno los coeficientes de sigma
    # para asi determinar el conjunto de posiciones de error␣
↪E - {i tal que e_i es distinto de 0}
    for i in range(N):
        if (sigma(a^i)==0): # an error occured
            print ("Error encontrado en la posición: " +␣
↪str(i))
            y[i] = y[i] + 1
    return y
```

[9]:
```
# Creando la matriz A de la matriz H.
C = matrix(F,rango,rango)
for i in range(rango):
    count = rango - i
    for j in range(rango):
        if i > j:
            C[i,j]=g.list()[count]
            count = count + 1
        if i < j:
            C[i,j] = 0
        if i == j:
            C[i,j] = 1
```

[10]:
```
print ("Matriz C: "); show(C)
print ("Cuerpo ");show(F)
print ("coeficientes ");show(g.list())
print ("polinomio Goppa ");show(g)
```

```
Matriz C:
```

[10]:
$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$

Cuerpo

[10]:
$$\mathbf{F}_{2^4}$$

coeficientes

[10]:
$$[1, 1, 0, 1]$$

polinomio Goppa

[10]:
$$Z^3 + Z + 1$$

[11]:
```
X = matrix([[L[j]^i for j in range(N)] for i in range(rango)])
Y = diagonal_matrix([1/g(L[i]) for i in range(N)])
print('Matriz X:')
show(X)
print('Matriz Y: ')
show(Y)
```

Matriz X:

[11]:
$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & a & a^2 & a^3 & a+1 & a^2+a & a^3+a^2 & a^3+a+1 & a^2+1 & a^3+a & a^2+a+1 & a^3+a^2+a & a^3+a^2+a+1 & a^3+a^2+1 & a^3+1 & 1 \\ 1 & a^2 & a+1 & a^3+a^2 & a^2+1 & a^2+a+1 & a^3+a^2+a+1 & a^3+1 & a & a^3 & a^2+a & a^3+a+1 & a^3+a & a^3+a^2+a & a^3+a^2+1 & 1 \end{pmatrix}$$

Matriz Y:

[11]:
$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & a^2+1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & a & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & a^3+a^2+a & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & a^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & a^2+a+1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & a^3+a+1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & a^2+a+1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a+1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a^3+a^2+1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a^2+a & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a^2+a & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a^3+1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a^2+a+1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a^2+a & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

[12]:
```
H = C*X*Y
H_Goppa_K = matrix(K_, r*H.nrows(),H.ncols())
for i in range(H.nrows()):
    for j in range(H.ncols()):
        be = bin(eval(H[i,j]._int_repr()))[2:];
        be = '0'*(r-len(be))+be; be = list(be);
        H_Goppa_K[r*i:r*(i+1),j]=vector(map(int,be));
show(H_Goppa_K)
```

[12]:

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

[13]:
```
Krnl = H_Goppa_K.right_kernel();
G = Krnl.basis_matrix();
print('G')
show(G)
m = vector(K_,[randint(0,1) for _ in range(G.nrows())])
c = m*G
print ('Vector m')
show(m)
print ('Vector c')
```

```
show(c)
e = vector(K_,N)
e[8] = 1
e[9] = 1
print ('Vector de errores e')
show(e)
y = c + e
print ("Vector codificado y")
show(y)
```

G

[13]:
$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Vector m

[13]:
$$(1, 1, 1, 0)$$

Vector c

[13]:
$$(1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0)$$

Vector de errores e

[13]:
$$(0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0)$$

Vector codificado y

[13]:
$$(1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0)$$

[14]:
```
sol = decodePatterson(y)
show(sol)
```

```
Error encontrado en la posición: 8
Error encontrado en la posición: 9
```

[14]:

$(1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0)$.

3. Criptografía poscuántica

La criptografía es un conjunto de técnicas que tiene como abjetivo asegurar la transmisión confidencial y segura de información en un entorno donde la seguridad no puede ser garantizada por el canal. Con el tiempo, los métodos criptográficos han evolucionado al presentarse nuevas amenazas tecnológicas. Con el avance acelerado de la computación cuántica, la criptografía debe afrontar la necesidad de desarrollar criptosistemas que sean resistentes a ataques cuánticos, a esta rama centrada en el estudio de dichos algoritmos se le conoce como criptografía poscuántica, actualmente un enfoque en la criptografía poscuántica es el uso de códigos correctores de errores para la creación de criptosistemas resistentes. La teoría de códigos ofrece una base matemática para desarrollar criptosistemas que puedan resistir ataques mediante computadores cuánticos, dentro de los cuales destaca el criptosistema McEliece, propuesto por Robert McEliece en 1978.

En este capítulo se detallará el criptosistema McEliece, destacando su funcionamiento y las características que lo hacen óptimo para enfrentar ataques tanto de computadores clásicos como computadores cuánticos.

3.1. Criptografía

La criptografía busca el intercambio seguro de información a través de un canal inseguro, para ello se usan la codificación y la decodificación del mensaje utilizando algún problema dificil de resolver para alguien que no tenga acceso, pero no para el emisor y receptor, estos procesos ahora se llamaran encriptación y descencriptación. En este sentido, la criptografía se ocupa de asegurar la confidencialidad e integridad de la información, protegiéndola contra terceros que no tienen acceso autorizado a esta. Los algoritmos de cifrado, descifrado y construcción de claves conforman lo que conocemos como criptosistemas. Actualmente existen

dos tipos de sistemas criptográficos, estos son los criptosistemas simétricos y los criptosistemas asimétricos.

Criptosistemas simétricos: Un sistema de cifrado simétrico emplea una única clave para cifrar y descifrar el mensaje. El emisor y receptor deben estar de acuerdo previamente sobre esta clave. Luego de esto, el emisor cifra la información usando la clave, la envía al receptor, quien al recibirla, la descifra empleando la misma clave. Un ejemplo de criptosistema simétrico es «Enigma», un sistema que fue usado por Alemania, en este, las claves eran asignadas a diario como libros de códigos. Diariamente, un operador, receptor o transmisor revisaba su copia del libro para encontrar la clave del día.

Criptosistemas asimétricos: Este método de cifrado utiliza dos claves diferentes, una pública y otra privada, vinculadas entre sí matemáticamente. Por ejemplo, las claves pueden ser dos números primos grandes relacionados mediante el producto, pero no idénticos, de ahí el término asimétrico. La clave pública se puede compartir libremente, mientras que la clave privada debe mantenerse en secreto.

El surgimiento de la criptografía basada en códigos fue inspirado por el trabajo de Robert J. McEliece en 1978. Él fue el primero en implementar el uso de códigos binarios de Goppa para desarrollar un criptosistema de clave pública basado en códigos correctores de errores. Como vimos al final del capítulo anterior los códigos de Goppa binarios tienen un algoritmo eficiente para su decodificación y se pueden construir códigos binarios de Goppa para corregir cualquier número de errores. La idea general de la seguridad detrás del criptosistema McElice está en la dificultad de decodificar un código lineal aleatorio, que es un problema dificil computacionalmente (NP completo, ver Berlekamp et al., 1978).

Básicamente, el criptosistema de McEliece utiliza como clave publica la matriz generadora del código de Goppa oculta mediante la mezcla y permutación de las entradas de esa matriz. El texto cifrado se genera codificando el mensaje con la matriz disponible en la clave pública y sumando a este un error aleatorio que pueda ser corregido por el código de Goppa.

Dado que el tamaño de la clave pública es usualmente grande, esto se considera una desventaja en el criptosistema McElice y lo hace menos popular que otros criptosistemas. Ha habido muchos intentos de reducir el tamaño de la clave. Ejemplos de esto incluyen los códigos cuasicíclicos (QC) y los códigos de comprobación de paridad de baja densidad (QC-LDPC).

Actualmente, la criptografía de clave pública más utilizada se basa en problemas dificiles de teoría de números, como la factorización de un número como producto de dos primos o el cálculo de logaritmos discretos. Sin embargo, uno de los principales inconvenientes es que serán vulnerables una vez que estén disponibles computadoras cuánticas de un tamaño apropiado. Existe entonces una fuerte necesidad de sistemas alternativos que puedan resistir a los atacantes equipados con tecnología cuántica.

Como el desarrollo de las computadoras cuánticas a avanzando a pasos acelarados en los últimos años, el riesgo para la criptografía actual está aumentando. Por lo tanto, el mundo moderno requiere tener sistemas criptográficos resistentes a la computación cuántica, los cuales se conocen como criptosistemas post-cuánticos. Los sistemas de encriptación basados en teoría de códigos son un tipo de criptosistema capaz de resistir la computación cuántica, lo que proporciona un área en la criptografía poscuántica.

Cuando el algoritmo de Shor apareció para impactar a los criptosistemas basados en teoría de números y el desarrollo de la computación cuántica evolucionó con el tiempo, el valor de los criptosistemas basados en códigos aumentó y el criptosistema de McEliece, siendo el más antiguo en ese grupo, cobró gran importancia al no ser afectado por el algoritmo de Shor, y al ser finalista en el concurso mundial de estandarización poscuántica del NIST[****] (National Institute of Standards and Technology) Instituto Nacional de Estándares y Tecnología de EE.UU.

La criptografía poscuántica, busca sistemas resistentes a ataques tanto de computadores clásicos como cuánticos. En el 2017 el NIST organizó un concurso para buscar un estándar en criptografía poscuántica, en este se presentaron 67 propuestas de todo el mundo que exploraban diversas áreas como la teoría de códigos, retículos, funciones de hash y álgebra multivariada. En particular, Daniel J. Bernstein et al. presentaron Classic McEliece, un criptosistema poscuántico basado en el criptosistema propuesto por McEliece en 1978 y sus variaciones, ya que,

[****] https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization

este esquema de cifrado se ha modificado varias veces debido al gran tamaño de clave y que apesar de los numerosos ataques documentados a lo largo de 40 años, el nivel de seguridad del criptosistema McEliece se ha mantenido sorprendentemente estable. Logrando en 2022 ser uno de los tres finalistas y por lo tanto candidato a la estandarización global por parte del NIST. En este capítulo presentaremos en detalle el criptosistema de McEliece y su implementación usando SageMath. Aunque el artículo original de McElice utiliza códigos de Goppa binarios, este criptosistemma asimetrico funciona para cualquier familia de códigos que tengan un algoritmo de decodificación eficiente.

3.2. Criptosistema McEliece

La siguiente grafica presenta la estructura del criptosistema de McElice para una familia de códigos correctores de errores en general. Tomando C un código lineal corrector de t errores con un algoritmo eficiente de decodificación Dec_C, $G \in \mathbb{F}_q^{k\times n}$ una matriz generadora para C, matrices aleatorias $S \in \mathbb{F}_q^{k\times k}$ invertible y $P \in \mathbb{F}_q^{n\times n}$ de permutación, y e es el vector error aleatorio introducido.

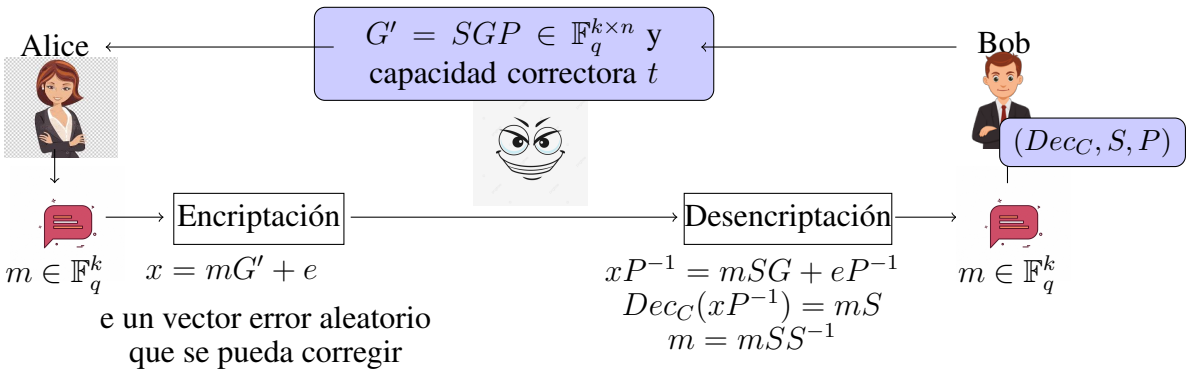


Figura 2: Esquema del Criptosistema McEliece

El criptosistema McEliece, desarrollado por Robert McEliece en 1978, ha ganado importancia por ser una alternativa resistente a los ataques de computación cuántica. Se basa en la teoría de codificación y su seguridad radica en la apariencia aleatoria de la matriz generadora del código y la dificultad de descifrar un código lineal cuya estructura no se conoce, el problema de descifrar el código es computacionalmente difícil para tamaños grandes; algunos algoritmos de descifrado, como los de los códigos de Reed-Solomon y los códigos de Goppa, permiten

corregir errores de manera eficiente. Para el criptosistema McEliece, se prefieren los códigos de Goppa clásicos binarios debido a su seguridad, simplicidad y mejores propiedades en comparación con otros códigos. Además, se pueden usar códigos Reed-Solomon y Reed-Solomon generalizados, aunque pueden ser vulnerables a ciertos ataques. Además, el criptosistema McEliece se puede construir con otros códigos lineales y códigos de producto de matrices para mayor versatilidad y seguridad.

En esta sección describimos en detalle el criptosistema McEliece, cubriendo sus debilidades y aplicaciones. Aunque, la versión original del criptosistema McEliece, dada por Robert J. McEliece en 1978, es basada en códigos binarios de Goppa, se puede extender de manera natural a cualquier código corrector de errores en general.

Sea C un $[n, k, d]$ código lineal sobre $\mathbb{F}_q$ capaz de corregir t errores con un algoritmo eficiente de decodificación Dec_C y $G \in \mathbb{F}_q^{k \times n}$ una matriz generadora para C

Entonces, el criptosistema McElice consta de los siguientes pasos:

(i) **Generación de claves**: el receptor genera una clave pública y una clave privada en función de los valores acordados anteriormente.

1. El receptor selecciona una matriz no singular aleatoria S de tamaño $k \times k$ y una matriz de permutación P de tamaño $n \times n$;
2. Calcula la matriz $\hat{G} = S \cdot G \cdot P$ de tamaño $k \times n$;
3. Publica su **clave pública**: $\{\hat{G}, t\}$;
4. Guarda su **clave privada**: (S, G, P).

(ii) **Encriptación**: El emisor quiere enviar un mensaje cifrado a este receptor:

1. El texto plano es un mensaje $m \in \mathbb{F}_q^k$;
2. Consulta la clave pública del receptor (en el directorio de claves): $\{\hat{G}, t\}$;
3. Genera un vector aleatorio e de n bits con peso de Hamming t;

4. Calcula el texto cifrado $c = m \cdot \hat{G} + e$ y lo envía.

(iii) **Desencriptación**: Supongamos que se recibe el texto cifrado $c \in \mathbb{F}_q^n$. El receptor lo decifra de la siguiente manera:

1. Calcula P^{-1} usando su clave privada;

2. Multiplica por P^{-1} el vector recibido, es decir calcula $c \cdot P^{-1} = m \cdot S \cdot G + e \cdot P^{-1}$, donde $e \cdot P^{-1}$ tiene peso t;

3. Utiliza el algoritmo de descodificación para corregir el error introducido obteniendo $m \cdot S \cdot G$ y con esto determina el mensaje codificado mS.

4. Finalmente, multiplicando por S^{-1} encuentra el mensaje enviado m.

Acontinuación presentamos un ejemplo "de juguete" del criptosistema McElice.

Ejemplo 3.1. *Texto plano:* $[0, 1, 0, 1]$

Tomando matriz generadora G del código de Hamming [7, 4, 3] y generamos S y P matrices aleatorias

$$G\text{:} \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}, \quad S: \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad P: \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$\hat{G}: \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Cifrado: generamos el vector error $e = [0, 0, 0, 0, 1, 0, 0]$ y ciframos el texto plano como

$$m\hat{G} + e = [0, 1, 0, 1]\hat{G} + e = [1, 0, 0, 1, 0, 1, 0] + [0, 0, 0, 0, 1, 0, 0] = [1, 0, 0, 1, 1, 1, 0]$$

Descifrado: multiplicamos por P^{-1} y usamos el algoritmo de decodificación de C.

$$P^{-1}: \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad y' = xP^{-1} = [0,1,0,1,0,1,1],$$

$$S^{-1}: \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Así el mensaje recibido fue : $[0,1,0,1]$

3.2.1. Criptosistema McEliece usando SageMath

Para implementar el criptosistema McEliece en SageMath para códigos binarios de Goppa, calculamos primero la matriz $\hat{G} = SGP$. Luego, generamos S, una matriz aleatoria invertible de orden $k \times k$, generando matrices aleatorias hasta que su determinante sea diferente de cero. También generamos la matriz P de permutaciones, es decir, que contenga solo un uno por fila y columna.

Después se genera un vector de errores e y se cifra el mensaje como $c = m \cdot \hat{G} + e$. Para descifrar el mensaje es necesario calcular la inversa de P, aplicar un algoritmo de decodificación y calcular la inversa de S. Tras aplicar el algoritmo de decodificación de Patterson, este elimina los errores devolviendo el vector mS, a partir del cual se obtiene el mensaje m. Veamos el siguiente ejemplo de implementación apoyandonos en el código mostrado en **Rambaut2021** .

[15]:
```
Krnl = H_Goppa_K.right_kernel();
G = Krnl.basis_matrix(); #matriz generadora para el codigo de␣
 ↪Goppa
S = random_matrix(GF(2), N-r*rango) # matriz binaria no␣
 ↪singular.

while (S.determinant()==0):
    S = random_matrix(GF(2), N-r*rango)

rng = range(N)

P = matrix(GF(2),N); #matriz de permutacion.

for i in range(N):
    p = floor(len(rng)*random());
    P[i,rng[p]]=1;
    rng=list(rng[:p])+list(rng[p+1:]);

G_gorro = S*G*P
show(G_gorro)
```

[15]:

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

[16]:
```
m = vector(K_,[randint(0,1) for _ in range(G_gorro.nrows())])
c = m*G_gorro
print ('Vector m');
show(m)
print ('Vector c');
```

```
show(c)
e = vector(K_,N)
e[8] = 1
e[9] = 1
print ('Vector de errores e') ; show(e)
y = c + e
print ("Vector codificado y") ; show(y)
```

```
Vector m
```

[16]:

$$(0, 1, 1, 0)$$

```
Vector c
```

[16]:

$$(0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1)$$

```
Vector de errores e
```

[16]:

$$(0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0)$$

```
Vector codificado y
```

[16]:

$$(0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1)$$

[17]:
```
yP = y*(P.inverse())
yd = decodePatterson(yP)
corregido = (G.transpose()\yd)*S.inverse()
show(corregido)
```

```
Error encontrado en la posición: 2
Error encontrado en la posición: 10
```

[17]:

$$(0, 1, 1, 0)$$

3.3. Ataques al criptosistema de McEliece

En esta sección se mostrará qué tan seguro es el criptosistema McElice, es decir, que tan difícil puede ser para un espía determinar el mensaje x, conociendo la clave publica $\{\hat{G}, t\}$. Uno de los ataques que puede intentar el espía es tratar de recuperar a G, la matriz generadora del codigo corrector de errores, apartir de $\hat{G}$ y así intuir que código se esta utilizando y, luego usar el algoritmo de decodificación. Otro tipo de ataque consiste en intentar recuperar el vector m de la palabra código c sin conocer G. Este estudio fue iniciado por McEliece McEliece, 1978 , donde mostró que para parámetros $n = 1024$, $k = 524$ y $t = 50$ el ataque es computacionalmente imposible.

3.3.1. Ataque de Decodificación de Conjuntos de Información

En Bernstein et al., 2008 se hicieron varias propuestas para mejorar las técnicas de decodificación de conjuntos de información (ISD), esto redujo a $2^{60,5}$ operaciones binarias el costo en ataques al McEliece original con parámetros $[1024, 524, 50]$, es decir, al código con

- Longitud del código n: 1024,
- Cuerpo de extensión binario F_{2^m}: $F_{2^{10}}$, es decir, $m = 10$,
- Grado t del polinomio de Goppa: 50,
- Dimensión k del código de Goppa: $k = n - mt = 524$.

Se demostró en Bernstein et al., 2008 que para obtener una seguridad de 128 bits, los códigos de Goppa deben tener una longitud de 2960 y una dimensión de 2288 con un polinomio de Goppa de grado 56 y 57 errores añadidos.

Sea $\hat{G}$ la clave pública del criptosistema de McEliece. Entonces, para un mensaje m, el texto cifrado c se obtiene como $m \cdot \hat{G} + e$, esto es:

$$\begin{aligned} m \cdot \hat{G} + e &= m_{1\times k} \cdot (G_1, G_2, \ldots, G_n)_{k\times n} + (e_1, e_2, \ldots, e_n)_{1\times n} \\ &= (mG_1 + e_1, mG_2 + e_2, \ldots, mG_n + e_n) \end{aligned}$$

donde, para $1 \leq i \leq n$, G_i representa la i-ésima columna de la matriz de clave pública dada.

Un detalle importante es que el peso de Hamming del vector de error $wt(e) = t$ es muy pequeño en comparación con la longitud de las palabras código. De esta forma, si se pudiera adivinar k de $n - t$ coordenadas c_i de c tal que $e_i = 0$, supongamos que para cada $1 \leq j \leq k$, $e_{i_j} = 0$ con $\{i_1, i_2, \ldots, i_k\} \subset \{1, 2, \ldots, n\}$, entonces la restricción a esas k columnas de c y la clave pública $\hat{G}$ se puede ver como:

$$\overline{c} = m \cdot \overline{\hat{G}} = (c_{i_1}, c_{i_2}, \ldots, c_{i_k})_{1\times k} = m_{1\times k} \cdot (G_{i_1}, G_{i_2}, \ldots, G_{i_k})_{k\times k}$$

Es decir, si la matriz $\hat{G}^{-1}$ de tamaño $k \times k$ es invertible, entonces el mensaje m se puede recuperar simplemente multiplicando por la inversa de $\overline{\hat{G}}$. Por tanto se requieren

$$\frac{\binom{n}{k}}{\binom{n-t}{k}}$$

suposiciones para tener éxito y el costo resultante es

$$k^3 * \frac{\binom{n}{k}}{\binom{n-t}{k}} \approx k^3 \cdot \left(1 - \frac{t}{n}\right)^{-k}$$

donde k^3 es el costo de invertir una matriz de tamaño $k \times k$. Al reemlazar con los parámetros originales del criptosistema de McEliece, encontramos que el costo para hallar un conjunto de información es aproximadamente $1{,}9 \times 10^{24}$, por lo que este algoritmo no resulta útil para realizar dicho ataque.

Observe que, la clave pública $\hat{G}$ es nuevamente una matriz generadora para algún código con distancia mínima mayor o igual que $2t + 1$. Consideremos dos mensajes u y u_0 entonces estos mensajes son distintos o son iguales.

Caso (i): Si $u \neq u_0$, entonces $d(u\hat{G}, u_0\hat{G}) > 2t$, es decir, $wt(u\hat{G} + u\hat{G}) > 2t$. Ahora, si $wt(e) = t$, tenemos $wt(u_0\hat{G} + u\hat{G} + e) > t$.

Caso (ii): Si $u = u_0$, entonces $u\hat{G} = u_0\hat{G}$. Ahora, si $wt(e) = t$, tenemos $wt(u_0\hat{G}+u\hat{G}+e) = t$.

Así, al recibir un texto cifrado $c = u\hat{G} + e$, adivina un mensaje u_0 y comprueba $wt(u_0\hat{G} + c)$. Si este peso no es igual a t, entonces los mensajes son distintos. Si el vector de error e fue elegido de tal manera que $wt(e) \leq t$, entonces también funcionan argumentos similares verificando $wt(u_0\hat{G} + c) \leq t$.

Una mejora a este ataque es el algoritmo de Lee-Brickell para recuperar el vector de error e del criptosistema original de McEliece usando la decodificación de conjuntos de información, el cual dada una matriz generadora G, un texto cifrado $y \in F_q^n$ y un parámetro $p \in \mathbb{N}$ sigue los siguientes pasos:

1. Elegir un conjunto de información aleatorio I de tamaño k y calcular y_I, G_I eligiendo las columnas correspondientes de G y $G' = G_I^{-1}G$ si existe la inversa.

2. Calcular $y_0 = y - y_I G'_I$.

3. Para cada subconjunto $\{a_1, \ldots, a_p\} \subset I$, para cada $x_1, x_2, \ldots, x_p \in \mathbb{F}_q \setminus \{0\}$, calcular el vector $\hat{g} = \sum_{i=1}^{p} x_i G'_{a_i}$.

4. Establecer $e = y_0 - \hat{g}$. Si el peso del vector e es $wt(e) = t$, devolver e.

5. Volver al paso 1.

Indicamos con G'_j la fila de G' en la cual en la posición j hay un 1. Note que, esta es única si j es un elemento de un conjunto de información.

El costo mínimo para esta implementación resulta ser aproximadamente $2^{73,4}$ para el conjunto de parámetros originales. En Peters, 2010, Peters generalizó los algoritmos de Stern y Lee-Brickell (ambos son variantes de los ataques ISD) en $\mathbb{F}_q$.

3.3.2. Ataque de reenvío de mensajes o mensajes relacionados

Supongamos que el emisor cifró un mensaje m dos veces, es decir se tienen dos textos cifrados c_1 y c_2 :

$$c_1 = m \cdot S \cdot G \cdot P + e_1$$
$$c_2 = m \cdot S \cdot G \cdot P + e_2$$

donde e_1 es diferente de e_2. Esto se llama condición de reenvío de mensaje y como el mismo mensaje se cifra dos veces, decimos que la profundidad de reenvío es 2. En este caso, es fácil recuperar m del sistema anterior. Sea $c_j(i)$ la i-ésima coordenada de c_j, entonces tenemos dos casos

- $c_1(i) + c_2(i) = e_1(i) + e_2(i) = 0$
- $c_1(i) + c_2(i) = e_1(i) + e_2(i) = 1$

Veamos que ocurre en cada caso:

- Si $e_1(i) + e_2(i) = 0$ entonces se tiene que $e_1(i) = 0 = e_2(i)$ o $e_1(i) = 1 = e_2(i)$. Suponiendo que el evento de elegir vectores de error es independiente, tenemos que la probabilidad de que $e_1(i) = 1 = e_2(i)$ es $\left(\frac{t}{n}\right)^2$. Así, para el caso de los parámetros originales del criptosistema de McEliece, esta probabilidad es

  $$\left(\frac{50}{1024}\right)^2 \approx 0{,}0024.$$

  Entonces, cuando consideramos $e_1(i) + e_2(i) = 0$, el caso mas probable es que tengamos$e_1(i) = 0 = e_2(i)$; equivalentemente, ni $c_1(i)$ ni $c_2(i)$ son alterados por vectores error.

- Si $e_1(i) + e_2(i) = 1$ entonces $c_1(i)$ o $c_2(i)$ es alterado por el vector error.

Ahora queremos calcular la probabilidad de adivinar k columnas no alteradas de aquellas en las que ocurre el primer caso. Sea p_m la probabilidad de que precisamente m coordenadas sean alteradas por e_1 y e_2. Entonces

$$p_m = \text{Pr}(|\{i : e_1(i) = 1\} \cap \{i : e_2(i) = 1\}| = i) = \frac{\binom{t}{i}\binom{n-k}{t-i}}{\binom{n}{t}}.$$

Por lo tanto, se espera que $|\{i \in \{1,2,\ldots,n\} : e_1(i) + e_2(i) = 1\}|$ sea

$$\sum_{m=0}^{t}(2t-2m)p_m$$

Para el conjunto original de parámetros del criptosistema de McEliece,

$$\sum_{m=0}^{t}(2t-2m)p_m \approx 95{,}1.$$

Por ejemplo, supongamos que $|\{i \in \{1,2,\ldots,n\} : e_1(i) + e_2(i) = 1\}| = 94$. Entonces $|\{i \in \{1,2,\ldots,n\} : e_1(i) + e_2(i) = 0\}| = 1024 - 94 = 930$, de los cuales son alterados aproximadamente $|\{i \in \{1,2,\ldots,n\} : e_1(i) + e_2(i) = 0\}| \times 0{,}0024 \approx 3$. Tenemos que la probabilidad de adivinar 5413 columnas no alteradas de aquellas en las que ocurre el primer caso es aproximadamente

$$\frac{\binom{927}{524}}{\binom{930}{524}} \approx 0{,}0828.$$

Así, en este caso, se espera que el ataque tenga éxito con solo 12 intentos, a un costo de $12 \times 524^3 \approx 10^{10}$. Estos resultados son mejores que un ataque exhaustivo de decodificación de conjuntos de información en un factor de 10^{15}.

De esta forma podemos concluir que la decodificación de conjuntos de información es un método eficiente para atacar el sistema McEliece, pero no es práctico, pues con un código suficientemente largo el sistema es irrompible. En Bernstein et al., 2008 se dan las longitudes de código recomendadas y su correspondiente seguridad que se muestran en la siguiente tabla.

Cuadro 4: Longitudes de código recomendadas para el criptosistema McEliece.

Longitud n del código	Peso t del vector de error	Seguridad (en bits)
512	21	33.0
1024	38	57.9
2048	69	103.5
4096	127	187.9
8192	234	344.6
16384	434	637.4

Nota. Adaptado de Bernstein et al., 2008

Bibliografía

Berlekamp, E., McEliece, R., & Van Tilborg, H. (1978). On the inherent intractability of certain coding problems (corresp.) [vid. p. 385]. *IEEE Transactions on Information Theory*, *24*(3), 384-386.

Bernstein, D. J., Chou, T., Lange, T., Maurich, I. V., Misoczki, R., Niederhagen, R., Persichetti, E., Peters, C., Schwabe, P., Sendrier, N., Szefer, J., & Wang, W. (s.f.). Post-quantum cryptography [NIST PQC Competition, 2017 (vid. pags. 20-25)].

Bernstein, D. J., Lange, T., & Peters, C. (2008). Attacking and defending the McEliece cryptosystem [vid. pags. 35-40]. En J. Buchmann & J. Ding (Eds.), *PQCrypto* (pp. 31-46).

Gallian, J. A. (2006). *Contemporary Abstract Algebra* (6th) [vid. pags. 50-55]. Houghton Mifflin.

Goppa, V. D. (1970). A new class of linear correcting codes [vid. p. 25]. *Problemy Peredachi Informatsii*, *6*(3), 24-30.

Goppa, V. D. (1981). Codes on algebraic curves [vid. pags. 171-172]. *Soviet Math. Dokl.*, 170-172.

Goppa, V. D. (1997). Codes associated with divisors [vid. p. 23]. *Problems of Inform. Trans.*, *13*, 22-26.

Huffman, W. C., & Pless, V. (2010). *Fundamentals of Error-Correcting Codes* [vid. pags. 36, 39]. Cambridge University Press.

McEliece, R. J. (1978). *A Public-Key System Based on Algebraic Coding Theory* (inf. téc.) (vid. p. 115). Jet Propulsion Lab.

Patterson, N. (1975). The algebraic decoding of Goppa codes [vid. p. 204]. *IEEE Transactions on Information Theory*, *21*(2), 203-207.

Peters, C. (2010). Information-Set Decoding for Linear Codes over Fq [vid. p. 85]. En *PQCrypto* (pp. 81-94).

Rambaut Lemus, D. F. (2021). *Introducción a la criptografía post-cuántica basada en teoría de códigos* [Tesis doctoral, Universidad del Rosario] [vid. pags. 20-25].

Rivest, R. L., Shamir, A., & Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems [vid. p. 121]. *Communications of the ACM*, *21*(2), 120-126.

Shannon, C. E. (1948). A mathematical theory of communication [vid. p. 400]. *Bell System Technical Journal*, *27*, 379-423.

Shor, P. W. (1994). Algorithms for quantum computation: Discrete logarithms and factoring [vid. p. 126]. En S. Goldwasser (Ed.), *FOCS* (pp. 124-134).