UNIVERSIDAD
DE GRANADA

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

Doble Grado en Ingeniería Informática y Matemáticas

TRABAJO DE FIN DE GRADO

Algoritmo de Sugiyama para códigos skew RS

Presentado por:
Guillermo Galindo Ortuño

Tutor:
José Gómez Torrecillas
Departamento de Álgebra

Francisco Javier Lobillo Borrego
Departamento de Álgebra

Curso académico 2019-2020

Algoritmo de Sugiyama para códigos skew RS

Guillermo Galindo Ortuño

Guillermo Galindo Ortuño *Algoritmo de Sugiyama para códigos skew RS*.
Trabajo de fin de Grado. Curso académico 2019-2020.

Responsable de tutorización

José Gómez Torrecillas
Departamento de Álgebra

Francisco Javier Lobillo Borrego
Departamento de Álgebra

Doble Grado en Ingeniería Informática y Matemáticas

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

Universidad de Granada

Licencia

Índice general

Resumen

El objetivo de este trabajo es implementar una versión del algoritmo de Sugiyama para la familia de códigos skew RS. En primer lugar, un código es una estructura algebraica que permite añadir información a un mensaje para poder corregir errores al recibir dicho mensaje a través de un canal ruidoso. Dicho esto, existen numerosas familias de códigos distintas, y en este trabajo utilizaremos la familia de códigos cíclicos sesgados, en particular la subfamilia de códigos skew RS para proporcionar un algoritmo de decodificación eficiente. Para ello, realizaremos un estudio de multitud de conceptos matemáticos necesarios para correctamente presentar y demostrar dicho algoritmo. Empezaremos introduciendo los fundamentos de la teoría de códigos, donde explicaremos su estructura, y cómo se aprovecha esta en la transmisión de mensajes. Tras esto, presentaremos la extensiones de Ore, que son la base de la familia de códigos para la que está diseñado el algoritmo. Por último, describiremos la familia de códigos cíclicos sesgados y seguidamente la subfamilia de códigos skew RS, para concluir presentando el algoritmo y su correspondiente demostración.

PALABRAS CLAVE: teoría de códigos códigos lineales códigos cíclicos polinomios sesgados extensiones de Ore Sugiyama SageMath

Summary

The main goal of this project is to present, study and implement a version of the Sugiyama algoritm for a class of skew cyclic codes, the skew RS codes. In order to accomplish this, firstly we need to study the basics of linear codes and Ore extensions, which we cover in the first and second chapter of this thesis.

In the first chapter, we introduce the basic concepts about linear codes. In essence, a linear code over a finite field is a vector subspace with an associated distance, so it is a well-known mathematical structure. One of the most important concepts that is presented in this chapter is the generator matrix, which is the main tool for enconding messages into the code. The most common distance and the one that we study is the Hamming distance, defined as the number of coordinates that differ between two vectors. Next in this chapter, we discuss the decoding procedure, presenting two of the simpliest decoding algorithms and a proof of a result that stablishes the maximum number of errors that a linear code can correct. This latter result is crucial in coding theory, as research in this area tries to find efficient algorithms which can decode as many errors as possible. Finally, we briefly introduce cyclic codes, in which the codes we will construct in following chapters are based.

In the second chapter, we introduce Ore extensions. These have a similar structure as common polynomial rings over a field, with the difference that the product operation is not commutative. Instead, this product is *twisted* by a field automorphism. The usual concepts about polynomial rings, as degree or leading coefficient, also exist for Ore extensions, and are included in this chapter. We will also introduce algorithms to calculate the division from left or right. These algorithms are essential and allow us to show that Ore extensions are non-commutative principal ideal domains, and to develop a left and right version of the extended Euclidean algorithm. The fact that the product is not commutative does not permit to think of the evaluation of these polynomials in the usual manner. Thus, there exist evaluations on the right and on the left, which are defined in accordance to the polynomial remainder theorem, and the respective division algorithm. In order to compute this evaluation, we present the notion of *jth-norm*, and show its relation with the remainder of left and right division by linear factors.

In the third chapter, we will present the algorithm whose analysis and implementation in the SageMath framework is the main goal of this thesis, and with it all the mathematical structures and results needed. Along this whole chapter, $\mathbb{F}[x;\sigma]$ will

represent the Ore extension over $\mathbb{F}$ given by the $\mathbb{F}$-automorphism σ. Then, we will firstly introduce the class of *skew cyclic codes*, which are described as left ideals of the quotient ring $\mathbb{F}[x;\sigma]/\langle x^n - 1\rangle$ where n is the order of σ. Therefore, the code itself is given by the image via the coordinate map (with respect to the usual basis) of the left ideal mentioned. Consecutively, we will present the necessary results to construct skew cyclic codes of a designed Hamming distance, which we denote by *skew RS codes*. Later in this chapter, the concepts of *error locator polynomial* and *error evaluator polynomial* will be described, and we will show how they allow us to find and correct the errors in a received message by simply solving a linear system. Then we will prove a relation between these two polynomials, called the *key equation*. Using this relation, we will be able to present the decoding algorithm and prove that it correctly decodes the message. Lastly, we will cover the procedure to follow when a *key equation error* occurs in order to find the error locator and evaluator polynomials, and analyze how probable is the ocurrence of this type of errors.

In the fourth and last chapter, we present the documentation of the classes that we have developed for SageMath. This classes store the information needed to represent both skew cyclic codes and skew RS codes, and allow us to work with the already existing structures in SageMath. The classes implemented are:

- A class to represent skew cyclic codes.
- Two classes to encode messages into skew cyclic code words, one for the vector form and another for the polynomial form.
- A class to represent skew RS codes.
- A class with the implementation of the decoding algorithm that is the goal of this thesis.

KEYWORDS: coding theory linear codes cycic codes skew polynomias Ore extension Sugiyama SageMath

Introducción

Se puede establecer el inicio de la Teoría de Códigos Correctores de Errores en el trabajo pionero [Sha48] de C. Shannon, en el que se establecen los elementos básicos de la comunicación, se desarrolla el concepto de redundancia y cómo puede usarse para detectar y corregir errores al transmitir información a través de un canal con ruido. Un paso esencial en dicha teoría fue la introducción de estructuras algebraicas en la gestión de la redundancia. De esta forma surgen los códigos lineales que, matemáticamente, pueden ser vistos como subespacios vectoriales de $\mathbb{F}_q^n$, donde $\mathbb{F}_q$ es el cuerpo finito de q elementos, junto con una métrica, usualmente la métrica de Hamming. La linealidad permite sistematizar la construcción de la redundancia, lo que permite construir codificadores de forma eficiente. Un problema mucho más complicado es el de eliminar los errores introducidos mediante la decodificación. La técnica general más estudiada es la decodificación por síndrome, que requiere manejar tablas cuyo tamaño para códigos con parámetros no excesivamente grandes puede ser inmenso. Para salvar esta dificultad, se enriquece la estructura algebraica de los códigos lineales. Entre los más estudiados están los códigos cíclicos, que pueden ser descritos como ideales del anillo cociente $\mathbb{F}_q[x]/\langle x^n - 1\rangle$. La descomposición completa de $x^n - 1$ en la extensión apropiada de $\mathbb{F}_q$ permite, mediante el cálculo de las raíces, establecer cotas para la distancia del código. En este sentido podemos destacar los códigos BCH, introducidos en [Hoc59, BRC60b, BRC60a], y dentro de ellos los códigos de Reed–Solomon, [RS60]. Estos códigos pueden decodificarse de forma eficiente mediante varios algoritmos, entre los que destacamos el algoritmo de Sugiyama [SKHN75], basado en el algoritmo extendido de Euclides.

El innegable éxito de la estructura polinomial en el diseño de códigos con buenos parámetros y sus correspondientes algoritmos de decodificación sugirió la idea de utilizar polinomios sesgados (skew polynomials) introducidos por O. Ore en [Ore33]. Estos polinomios no conmutativos mantienen muchas de las propiedades de los polinomios conmutativos, como la existencia de algoritmos de división, en este caso a izquierda y derecha, de los que se deriva el cálculo de máximos común divisores y mínimos común múltiplos laterales mediante los correspondientes algoritmos extendidos de Euclides a ambos lados. Por el contrario la evaluación de polinomios sí es esencialmente diferente, tal y como se puede observar en [LL88, Ler95]. En 2007, Boucher, Geiselmann y Ulmer introducen en [BGU07] los *skew cyclic codes* (códigos cíclicos sesgados) mediante los polinomios anteriormente descritos con coeficientes en cuerpos finitos, y desarrollan junto con otros autores algunas propiedades de los mismos en [BU09a, BU09b, CLU09], incluyendo una primera adaptación al caso sesgado del algoritmo de Sugiyama.

En 2016, Gómez-Torrecillas, Lobillo y Navarro extienden en [GTLN16] la noción de ciclicidad sesgada a cuerpos de funciones racionales sobre cuerpos finitos, lo que cubre el caso de los llamados códigos convolucionales, y un año más tarde publican en [GTLN17], una versión del algoritmo de Sugiyama para códigos sesgados de Reed-Solomon (skew RS), versión adaptada en este trabajo para cubrir cualquier extensión cíclica de cuerpos.

Dichos algoritmos han sido implementados en SageMath como parte de este Trabajo de Fin de Grado. Citando [S+20], *"SageMath is a free open-source mathematics software system licensed under the GPL. It builds on top of many existing open-source packages: NumPy, SciPy, matplotlib, Sympy, Maxima, GAP, FLINT, R and many more. Access their combined power through a common, Python-based language or directly via interfaces or wrappers."* (SageMath es un software matemático libre de código abierto publicado bajo licencia GPL. Está construido sobre muchos paquetes de código abierto existentes: NumPy, SciPy, matplotlib, Sympy, Maxima, GAP, FLINT, R y muchos otros. Accede a su potencia combinada a través de un lenguaje común basado en Python o directamente a través de interfaces o empaquetadores.)

1 Conceptos básicos sobre códigos

En este capítulo presentaremos los conceptos básicos sobre códigos lineales, para lo cual nos hemos basado casi por completo en el primer capítulo de [HP03].

1.1. Códigos lineales, matrices generadora y de paridad

Sea $\mathbb{F}_q$ el cuerpo finito de q elementos y $\mathbb{F}_q^n$ el espacio vectorial de todas las n-tuplas sobre dicho cuerpo.

Definición 1 (Código lineal)**.** Si $\mathcal{C}$ es un subespacio vectorial de dimensión k de $\mathbb{F}_q^n$, diremos que $\mathcal{C}$ es un $[n,k]$ *código lineal* sobre $\mathbb{F}_q$.

Normalmente escribiremos los vectores $(a_1, a_2, \ldots, a_n)$ en $\mathbb{F}_q^n$ de la forma $a_1 a_2 \cdots a_n$ y llamaremos a los vectores en $\mathcal{C}$ *palabras código*. Además, utilizaremos nombres concretos para referirnos a códigos sobre algunos de los cuerpos más comunes. A los códigos sobre $\mathbb{F}_2$ los llamaremos *códigos binarios*, los códigos sobre $\mathbb{F}_3$ los notaremos como *códigos ternarios* y a los códigos sobre $\mathbb{F}_4$ los llamaremos *códigos cuaternarios*.

Dicho esto, las dos maneras más comunes de presentar un código lineal son dando una matriz generadora o una matriz de paridad.

Definición 2 (Matriz generadora)**.** Una *matriz generadora* de un $[n,k]$ código lineal $\mathcal{C}$ es cualquier matriz G de dimensiones $k \times n$ cuyas filas formen una base de $\mathcal{C}$.

Dada una matriz generadora G, para cualquier conjunto de k columnas independientes de esta, diremos que el correspondiente conjunto de coordenadas o posiciones es un *conjunto de información* de C. Las restantes $r = n - k$ coordenadas las notaremos como *conjunto de redundancia*, y llamaremos a r *redundancia* de $\mathcal{C}$. Si las primeras k coordenadas forman un conjunto de información, existe una única matriz generadora para el código de la forma $[I_k|A]$ donde I_k es la matriz identidad de orden k. Diremos que una matriz generadora así está en *forma estándar*.

Definición 3. Una *matriz de paridad* de un $[n,k]$ código lineal $\mathcal{C}$ es cualquier matriz H de dimensiones $(n-k) \times n$ tal que

$$\mathcal{C} = \{x \in \mathbb{F}_q^n | Hx^T = 0\}.$$

Como un código lineal es un subespacio de un espacio vectorial, es el núcleo de alguna aplicación lineal, y por tanto, para un código lineal siempre existe alguna

matriz de paridad H. Mencionemos que las filas de H son también independientes. Esto es porque, H es una aplicación lineal de $\mathbb{F}_q^n$ en $\mathbb{F}_q^{n-k}$, y el núcleo de dicha aplicación lineal es $\mathcal{C}$ que ya dijimos que tiene dimensión k, por tanto la dimensión de la imagen de la aplicación lineal dada por H es $n-k$ y así el rango de H también es $n-k$.

1.2. Pesos y distancias

La característica que distingue un código lineal de un mero subespacio vectorial es la distancia. En realidad, un código lineal debería definirse como un subespacio vectorial de un espacio vectorial dotado de una distancia. Aunque no las tratemos aquí, hay otras distancias, como la del rango, que se usan. De esta manera, el mismo subespacio vectorial puede considerarse como dos códigos distintos.

Definición 4. Dados dos vectores $x, y \in \mathbb{F}_q^n$, definimos la distancia *Hamming* entre ellos $d(x,y)$ como el numero de coordenadas en las que x e y difieren.

Veamos que en efecto esta es una distancia:

Proposición 5. *La función distancia $d(x,y)$ satisface las siguientes condiciones:*

1. $d(x,y) \geq 0$ *para todo* $x, y \in \mathbb{F}_q^n$.
2. $d(x,y) = 0$ *si y solo si* $x = y$.
3. $d(x,y) = d(y,x)$ *para todo* $x, y \in \mathbb{F}_q^n$
4. $d(x,z) \leq d(x,y) + d(y,z)$ *para todo* $x, y, z \in \mathbb{F}_q^n$

Demostración. Las tres primeras propiedades se comprueban directamente por la propia definición de d. Veamos pues la demostración de la propiedad 4.

Dados dos vectores $x, y \in \mathbb{F}_q^n$, definimos el conjunto $D(x,y) = \{i | x_i \neq y_i\}$, y denotamos el complementario por $D^c(x,y) = \{i | x_i = y_i\}$. Es claro que entonces el cardinal de $D(x,y)$ coincide con nuestra distancia.

Recordemos también algunas propiedades sobre cardinales de conjuntos que nos serán útiles para terminar la prueba. Sea un conjunto A, notemos por $|A|$ su cardinal. Entonces, para cualesquiera conjuntos A y B:

1. $|A| \leq |A \cup B|$
2. $|A \cup B| \leq |A| + |B|$
3. Si $|A| \leq |B|$, entonces $|A^c| \geq |B^c|$

Con esto, dados $x, y, z \in \mathbb{F}_q^n$, conjuntos tenemos que

$$\begin{aligned} D^c(x,z) &= \{i|x_i = z_i\} = \{i|x_i = z_i = y_i\} \cup \{i|x_i = z_i \neq y_i\} \\ \implies |D^c(x,z)| &\geq |\{i|x_i = z_i = y_i\}| = |\{i|x_i = y_i\} \cap \{i|z_i = y_i\}| \\ \implies |D(x,z)| &\leq |\{i|x_i \neq y_i\} \cup \{i|z_i \neq y_i\}| = |D(x,y) \cup D(y,z)| \\ \implies |D(x,z)| &\leq |D(x,y)| + |D(y,z)| \end{aligned}$$

quedando demostrado el resultado. □

Ahora, la *distancia (mínima)* de un código $\mathcal{C}$ es la mínima distancia entre dos palabras distintas de dicho código. Esta propiedad será crucial a la hora de determinar el número de errores que podrá corregir un código.

Definición 6. Diremos que el peso (*de Hamming*) $wt(x)$ de un vector $x \in \mathbb{F}_q^n$ es el número de coordenadas distintas de cero de x.

Si tenemos dos vectores $x, y \in \mathbb{F}_q^n$ es inmediato comprobar que $d(x,y) = wt(x-y)$. En el siguiente resultado mostramos la relación entre las propiedades de distancia y peso.

Proposición 7. *Si $\mathcal{C}$ es un código lineal, la distancia mínima coincide con el mínimo de los pesos de las palabras distintas de cero de $\mathcal{C}$.*

Demostración. Sea $\delta = d(x,y)$ la distancia mínima del código $\mathcal{C}$, que se alcanza entre dos vectores $x, y \in \mathcal{C}$, y $\omega = wt(z)$ el peso mínimo que se alcanza en un elemento $z \in C$. Sabemos que x, y y z existen pues el conjunto de distancias posibles es finito.

Por ser $\mathcal{C}$ un subespacio vectorial, $0 \in \mathcal{C}$, y por tanto $\delta \leq d(z,0) = wt(z) = \omega$. De nuevo por ser $\mathcal{C}$ un subespacio vectorial tenemos que $x - y \in \mathcal{C}$, luego $\omega \leq wt(x-y) = d(x,y) = \delta$, de donde $\delta = \omega$. □

Como consecuencia de este resultado, para códigos lineales, a la distancia mínima también se la llama *peso mínimo* del código. En adelante, si el peso mínimo d de un $[n,k]$ código es conocido, entonces nos referiremos al código como un $[n,k,d]$ código.

A continuación mostramos que existe una relación elemental entre el peso de una palabra y una matriz de paridad de un código lineal.

Proposición 8. *Sea $\mathcal{C}$ un código lineal con matriz de paridad H, y $c \in \mathcal{C}$ una palabra de dicho código. Entonces, las columnas de H que corresponden a coordenadas no nulas de c son linealmente dependientes. En el sentido contrario, si existe una dependencia lineal con coeficientes no nulos entre ω columnas de H, entonces existe una palabra en $\mathcal{C}$ de peso w cuyas coordenadas no nulas corresponden a dichas columnas.*

Demostración. Sea $c = (c_1, c_2, \ldots, c_n) \in \mathcal{C}$, $J \subset \{1, 2, \ldots, n\}$ tal que $c_j \neq 0$ si y solo si $j \in J$. Entonces, por ser H una matriz de paridad de $\mathcal{C}$ tenemos que

$$Hc^T = 0,$$

es decir, si $H = (h_{ij})$ con $i \in \{1, \ldots, n-k\}, j \in \{1, \ldots, n\}$,

$$Hc^T = \begin{pmatrix} \sum_{j=1}^{n} h_{1j}c_j \\ \sum_{j=1}^{n} h_{2j}c_j \\ \vdots \\ \sum_{j=1}^{n} h_{(n-k)j}c_j \end{pmatrix} = \sum_{j=1}^{n} c_j \begin{pmatrix} h_{1j} \\ h_{2j} \\ \vdots \\ h_{(n-k)j} \end{pmatrix} = \sum_{j \in J} c_j \begin{pmatrix} h_{1j} \\ h_{2j} \\ \vdots \\ h_{(n-k)j} \end{pmatrix} = 0$$

quedando demostrada la primera parte.

Por otro lado, dado $J = \{j_1, \ldots, j_w\} \subset \{1, \ldots, n\}$, supongamos que tenemos una dependencia lineal entre las columnas asociadas a J dada por $c_{j_1}h_{ij_1} + \cdots + c_{j_w}h_{ij_w}$ para cualquier $i = 1, \ldots, n-k$. Entonces, si construimos $c = (c_1, \ldots, c_n) \in \mathbb{F}_q^n$ como sigue

$$\begin{cases} c_j = 0 & \text{si } j \notin J \\ c_j = c_{j_l} & \text{si } j = j_l \in J \end{cases}',$$

es evidente que $wt(c) = w$, y que

$$Hc^T = \sum_{j \in J} c_j \begin{pmatrix} h_{1j} \\ h_{2j} \\ \vdots \\ h_{(n-k)j} \end{pmatrix} = 0$$

terminando la demostración de la proposición. □

Una manera de encontrar la distancia mínima d de un código lineal es examinar todas las palabras no nulas. El siguiente corolario, que es consecuencia directa de la proposición recién demostrada, muestra como utilizar una matriz de paridad para hallar d.

Corolario 9. *Un código lineal tiene peso o distancia mínima d si y solo si su matriz de paridad tiene un conjunto de d columnas linealmente dependientes pero ninguno de $d-1$ columnas linealmente dependientes.*

1.3. Codificar y decodificar

1.3.1. Codificar

Sea $\mathcal{C}$ un $[n, k]$ código lineal sobre el cuerpo $\mathbb{F}_q$ con matriz generadora G. Como este es un subespacio vectorial de $\mathbb{F}_q^n$ de dimensión k, contiene q^k palabras, que están

en correspondencia uno a uno con q^k posibles mensajes. Por esto, la forma más simple es ver estos mensajes como k-tuplas x en $\mathbb{F}_q^k$. Así, lo más común es codificar un mensaje x como la palabra $c = xG$. Si G está en forma estándar, las primeras k coordenadas son los símbolos de información x; el resto de $n - k$ símbolos son los de paridad, es decir, la redundancia añadida a x con el fin de poder recuperarla si ocurre algún error. Dicho esto, la matriz G puede no estar en forma estándar. En particular, si existen índices de columnas $i_1, i_2, \ldots, i_n$ tales que la matriz $k \times k$ formada por estas columnas es la matriz identidad, entonces el mensaje se encuentra en las coordenadas $i_1, i_2, \ldots, i_n$ separado pero sin modificar, es decir, el símbolo del mensaje x_j se encuentra en la componente i_j de la palabra código. Si esto ocurre diremos que el codificador es *sistemático*.

1.3.2. Decodificar

El proceso de decodificar, consistente en determinar qué palabra (y por tanto qué mensaje x) fue mandado al recibir un vector y, es más complejo. Encontrar algoritmos de decodificación eficientes es una área de investigación muy relevante en la teoría de códigos debido a sus aplicaciones prácticas. En general, codificar es sencillo y decodificar es complicado, especialmente si tiene un tamaño suficientemente grande.

Con la intencion de establecer las bases para decodificar, comenzamos con un posible modelo matemático de un canal transimitiendo información binaria. A este modelo se le llama *canal binario simétrico* (o CBS) con *probabilidad de recombinación* ϱ y lo mostramos en la figura 1.1. Si un 0 o un 1 son mandados por el canal, la probabilidad de recibirlo sin errores es $1 - \varrho$; si un 0 (respectivamente un 1) son mandados, la probabilidad de recibir un 1 (o un 0 respectivamente) es ϱ. En la mayoría de casos prácticos ϱ es muy pequeña. Esto es un ejemplo de un *canal discreto sin memoria* (o CDM), es decir, un canal en el que la salida y la entrada son discretos y la probabilidad de error en un bit es independiente de los bits anteriores en caso de existir. Asumiremos que lo normal sea que un bit se reciba sin errores, así que diremos $\varrho < 1/2$.

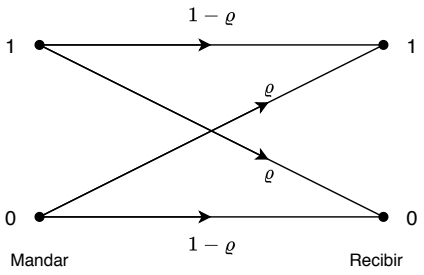

Figura 1.1: Canal Binario Simétrico

Si E_1 y E_2 son dos posibles eventos, denotaremos por prob(E_1) la probabilidad

de que E_1 ocurra y $\text{prob}(E_1 \mid E_2)$ la probabilidad de que E_1 ocurra supuesto que E_2 ocurre. Supongamos que $c \in \mathbb{F}_2^n$ es mandado e $y \in \mathbb{F}_2^n$ es recibido y decodificado como $\widehat{c} \in \mathbb{F}_2^n$. Así, $\text{prob}(y \mid c)$ es la probabilidad de que y sea recibida supuesto que la palabra c es mandada, y $\text{prob}(c \mid y)$ es la probabilidad de que la palabra c sea mandada supuesto que y es recibida. Estas probabilidades pueden calcularse a partir de las estadísticas asociadas al canal. La relación entre estas probabilidades dada por la regla de Bayes es:

$$\text{prob}(c \mid y) = \frac{\text{prob}(y \mid c)\,\text{prob}(c)}{\text{prob}(y)},$$

donde $\text{prob}(c)$ es la probabilidad de que c sea mandada y $\text{prob}(y)$ es la probabilidad de que y sea recibida. Hay dos aproximaciones naturales para que un decodificador haga una elección basada en dicha relación. La primera es elegir $\widehat{c} = c$ para aquella palabra código c con $\text{prob}(c \mid y)$ máxima; un decodificador de este tipo se llama un *decodificador de probabilidad máxima a posteriori (o decodificador MAP)*. Simbólicamente, un decodificador MAP realiza la elección

$$\widehat{c} = \arg\max_{c \in \mathcal{C}} \text{prob}(c \mid y).$$

Alternativamente, el decodificador podría elegir $\widehat{c} = c$ para aquella palabra c que maximice $\text{prob}(y \mid c)$; este tipo de decodificador se llama *decodificador de máxima similitud (o decodificador ML)*. Simbólicamente, un decodificador ML elige

$$\widehat{c} = \arg\max_{c \in \mathcal{C}} \text{prob}(y \mid c).$$

Consideremos un decodificador ML sobre un CBS. Si $y = y_1 \cdots y_n$ y $c = c_1 \cdots c_n$,

$$\text{prob}(y \mid c) = \prod_{i=1}^{n} \text{prob}(y_i \mid c_i),.$$

pues asumimos que los errores en bits distintos son independientes. Como dijimos, $\text{prob}(y_i \mid c_i) = \varrho$ si $y_i \neq c_i$ y $\text{prob}(y_i \mid c_i) = 1 - \varrho$ en caso contrario. Por tanto,

$$\text{prob}(y \mid c) = \varrho^{d(y,c)}(1-\varrho)^{n-d(y,c)} = (1-\varrho)^n \left(\frac{\varrho}{1-\varrho}\right)^{d(y,c)}.$$

Como $0 < \varrho < 1/2$, $0 < \varrho/(1-\varrho) < 1$. Por tanto, maximizar $\text{prob}(y \mid c)$ es equivalente a minimizar $d(y,c)$, esto es, encontrar la palabra código c más cercana al vector recibido y según la distancia Hamming; a esto se le llama *decodificación del vecino más cercano*. Por tanto, sobre un CBS, decodificar según máxima similitud y según el vecino más cercano es equivalente.

Sea $e = y - c$ de forma que $y = c + e$. El efecto del ruido en el canal de comunicación consiste en sumar un *vector de error* e a la palabra código c, y el objetivo al decodificar es determinar dicho e. Decodificar según el vecino más cercano es equivalente a encontrar un vector e de mínimo peso tal que $y - e$ esté en el código. Este

vector de error no tiene por qué ser único, pues puede existir más de una palabra código más cercana a y. En caso de que un decodificador sea capaz de encontrar todas las palabras más cercanas al vector recibido y diremos que se trata de un *decodificador completo*.

Para estudiar los vectores más cercanos a una palabra código dada, el concepto de esferas sobre palabras códigos es de utilidad. Se define naturalmente la *esfera* de radio r centrada en un vector $u \in \mathbb{F}_q^n$ como el conjunto

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

de todos los vectores cuya distancia a u es menor o igual que r. El número de elementos en $S_r(u)$ puede calcularse y el resultado es

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

Ahora, dichas esferas centradas en palabras código serán disjuntas si escogemos su radio suficientemente pequeño. El siguiente teorema muestra que se puede elegir un radio tal que ocurra esto en función de la distancia mínima del código.

Teorema 10. *Sea d la distancia mínima de un código $\mathcal{C}$, y $t = \lfloor (d-1)/2 \rfloor$, entonces las esferas de radio t sobre distintas palabras código son disjuntas.*

Demostración. Si $z \in S_t(c_1) \cap S_t(c_2)$, donde c_1 y c_2 son palabras código, entonces por la desigualdad triangular (proposición 5),

$$d(c_1, c_2) \leq d(c_1, z) + d(z, c_2) \leq 2t < d,$$

y por tanto $c_1 = c_2$. □

Corolario 11. *Siguiendo la notación del teorema anterior, si una palabra código c es transmitida y y es el vector recibido donde t o menos errores han ocurrido, entonces c es la única palabra código más cercana a y. En particular, la decodificación del vecino más cercano decodifica correctamente y de forma única cualquier vector en el como mucho hayan ocurrido t errores en su transmisión.*

Con el objetivo de poder decodificar tantos errores como sea posibles, este corolario implica que para n y k, queremos encontrar un código con un peso mínimo d tan grande como sea posible. Alternativamente, dados n y d, queremos ser capaces de mandar tantos mensajes distintos como sea posible; por tanto queremos encontrar un código con el mayor número de palabras código, es decir, con la mayor dimensión. Junto con estos requisitos también hay que tener en cuenta la eficiencia del algoritmo de decodificación.

Supuesto que la distancia mínima de $\mathcal{C}$ es d, existen dos palabras código distintas tales que las esferas de radio $t+1$ centradas en ellas no son disjuntas. Por tanto, si

ocurren más de t errores, la decodificación del vecino más cercano puede devolver más de una palabra más cercana. El radio de empaquetado de un código es el mayor de los radios de esferas centradas en palabras código tal que las esferas son disjuntas dos a dos. Esta discusión la recogemos en el siguiente teorema.

Teorema 12. *Sea $\mathcal{C}$ un $[n,k,d]$ código sobre $\mathbb{F}_q$. Entonces:*

1. *El packing radius de $\mathcal{C}$ vale $t = \lfloor (d-1)/2 \rfloor$.*
2. *El packing radius t de $\mathcal{C}$ está caracterizado por la propiedad de que la decodificación por el vecino más cercano siempre decodificará correctamente un vector recibido en el que hayan ocurrido t o menos errores pero no siempre decodificará correctamente un vector recibido en el que hayan ocurrido $t+1$ o más errores.*

El problema de decodificar se convierte ahora en buscar un algoritmo eficiente que corrija hasta t errores. Uno de los algoritmos más evidentes de decodificación es examinar todas las palabras código hasta que se encuentre una a distancia t o menos del vector recibido. Pero obviamente esto solo puede ser de utilidad para códigos con un número suficientemente pequeño de palabras código. Otro algoritmo obvio es crear una tabla que contenga la palabra código más cercana a cada posible vector de $\mathbb{F}_q^n$ y luego buscar el vector recibido en dicha tabla para decodificarlo. Esto también es impracticable si q^n es muy grande.

Para un $[n,k,d]$ código lineal $\mathcal{C}$ sobre $\mathbb{F}_q$, existe un algoritmo que utiliza una tabla similar al último mencionado pero con q^{n-k} en lugar de q^n entradas donde podemos buscar la palabra código más cercana buscando en dichas entradas. Este algoritmo general de decodificación se llama *decodificación de síndromes*. Como nuestro código $\mathcal{C}$ es un subgrupo abeliano del grupo aditivo de $\mathbb{F}_q^n$, las distintas clases de la forma $x+\mathcal{C}$ forman una partición de $\mathbb{F}_q^n$ en q^{n-k} conjuntos de tamaño q^k Dos vectores x e y pertenecen a la misma clase si y solo si $x-y \in \mathcal{C}$. El *peso de una clase* es el menor de los pesos de los vectores en dicha clase, y cualquier vector de peso mínimo en la clase se llama *líder de la clase*. El vector cero es el único líder del código $\mathcal{C}$. De manera más general, cualquier clase con peso mínimo como mucho $t = \lfloor (d-1)/2 \rfloor$ tiene un único líder de clase.

Proposición 13. *Sea $\mathcal{C}$ un $[n,k,d]$ código sobre $\mathbb{F}_q$, cualquier clase de peso como mucho $\lfloor (d-1)/2 \rfloor$ tiene un único líder de clase.*

Demostración. Razonemos por contradicción. Dado $x \in \mathbb{F}_q^n$ tal que x sea el líder de la clase $x+\mathcal{C}$ y $\mathrm{wt}(x) \leq \lfloor (d-1)/2 \rfloor$, supongamos que existe $y \in x+\mathcal{C}$ tal que $\mathrm{wt}(y) = \mathrm{wt}(x)$. Entonces,

$$\mathrm{wt}(x-y) \leq d(x,0) + d(0,y) = 2\,\mathrm{wt}(x) \leq d-1 < d,$$

siendo esto una contradicción con la distancia mínima de $\mathcal{C}$. □

Elijamos ahora una matriz de paridad H para $\mathcal{C}$. El síndrome de un vector $x \in \mathbb{F}_q^n$ con respecto a la matriz de paridad H es el vector en $\mathbb{F}_q^{n-k}$ dado por

$$\text{syn}(x) = Hx^T.$$

El código $\mathcal{C}$ entonces está formado por todos los vectores cuyo síndrome valga 0. Como sabemos que H tiene rango $n-k$, cada vector en $\mathbb{F}_q^{n-k}$ es el síndrome de algún vector. Ahora, si $x_1, x_2 \in F_q^n$ están en la misma clase, entonces $x_1 - x_2 = c \in \mathcal{C}$. Por tanto, $\text{syn}(x_1) = H(x_2 + c)^T = Hx_2^T + Hc^T = Hx_2^T = \text{syn}(x_2)$. Por esto x_1 y x_2 tienen el mismo síndrome. Por otro lado, si $\text{syn}(x_1) = \text{syn}(x_2)$, entonces $H(x_2 - x_1)^T = 0$ luego $x_2 - x_1 \in \mathcal{C}$. Por esto tenemos la siguiente proposición.

Proposición 14. *Dos vectores pertenecen a la misma clase si y solo si tienen el mismo síndrome.*

Esto nos dice que existe una relación uno a uno entre clases de $\mathcal{C}$ y síndromes. Llamaremos C_s a la clase formada por todos los vectores de $\mathbb{F}_q^n$ con síndrome s.

Supongamos ahora que una palabra código transimitida sobre un canal de comunicación es recibida como un vector y. Como al decodificar según el vecino más cercano buscamos un vector e de mínimo peso tal que $y - e \in \mathcal{C}$, esta decodificación es equivalente a buscar el vector e de menor peso en la clase de $\mathcal{C}$ que contenga a y, eso es el líder de la clase que contenga a y. Presentamos a continuación el *Algoritmo de Decodificación por Síndromes*. Supongamos H una matriz de paridad fija del código.

1. Para cada síndrome $s \in F_q^{n-k}$ elegir un líder de clase e_s de la clase $\mathcal{C}_s$. Crear una tabla que empareje el síndrome con el líder de clase.

   Este proceso es en realidad preprocesado que solo necesitaremos hacer una vez, y que se debe realizar antes de recibir cualquier vector.

2. Al recibir un vector y, calcular su síndrome utilizando la matriz de paridad H de la forma $s = \text{syn}(y) = Hy^T$.

3. y entonces es decodificado como $c = y - e_s$.

Este es el primer algoritmo de decodificación no trivial que mostramos. Este resulta muy útil para comprender en qué consiste el concepto de decodificar y es un algoritmo válido para códigos de tamaño pequeño, pues para códigos de longitud suficientemente grande el tamaño de la tabla puede ser desmesurado y no ser útil debido a las limitaciones en memoria.

1.4. Códigos cíclicos

A continuación introduciremos brevemente el concepto de *códigos cíclicos*, ya que en estos están basados los códigos que utilizaremos en nuestro algoritmo de decodificación principal. Un código lineal $\mathcal{C}$ de longitud n sobre $\mathbb{F}_q$ es cíclico si, para cada

vector $c = c_0 \cdots c_{n-2}c_{n-1}$ en $\mathcal{C}$, el vector $c_{n-1}c_0 \cdots c_{n-2}$ obtenido a partir de c al ciclar las coordenadas $i \to i+1 \pmod{n}$ está también en $\mathcal{C}$.

Cuando se habla de códigos cíclicos, por defecto se representan las palabras código en forma polinomial. Para ello se utiliza la biyección entre los vectores $c = c_0 \cdots c_{n-2}c_{n-1}$ en $\mathbb{F}_q^n$ y los polinomios $c(x) = c_0 + c_1x + \cdots c_{n-1}x^{n-1}$ en $\mathbb{F}_q[x]$ de grado menor estricto que n. Utilizando esta representación de las palabras código, la condición de ciclicidad es equivalente a que si $c(x) = c_0 + c_1x + \cdots c_{n-1}x^{n-1}$ está en $\mathcal{C}$, entonces también lo está $xc(x) = c_{n-1}x^n + c_0x + c_1x^2 + \cdots + c_{n-2}x^{n-1}$ supuesto que multiplicamos módulo $x^n - 1$. Esto sugiere que el contexto correcto para estudiar los códigos cíclicos sea el anillo de polinomios cociente

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

Bajo la correspondencia de vectores con polinomios recién dada, es directo ver que los códigos cíclicos son ideales de $\mathcal{R}_n$ y los ideales de $\mathcal{R}_n$ son códigos cíclicos, y por tanto el estudio de códigos cíclicos en $\mathbb{F}_q^n$ es equivalente al estudio de ideales de $\mathcal{R}_n$.

2 Introducción a extensiones de Ore

A continuación introduciremos los conceptos principales sobre extensiones de Ore que utilizaremos como base para el siguiente capítulo. Las definiciones y principales resultados de este capítulo siguen el desarrollo realizado en [**?**].

2.1. Conceptos básicos sobre extensiones de Ore

Aunque las definiciones que realizamos a continuación se pueden hacer más generales, en este caso nos restringiremos al caso más particular necesario para continuar. Dicho esto, nuestro algoritmo trabajará sobre polinomios de Ore no conmutativos, con una única indeterminada x, con coeficientes en un cuerpo cualquiera. Precisando, nuestros polinomios serán elementos de un anillo asociativo $R = \mathbb{F}[x;\sigma]$, donde

- $\mathbb{F}$ es un cuerpo cualquiera.
- $\sigma : \mathbb{F} \to \mathbb{F}$ es un automorfismo de cuerpos de orden finito digamos n.

La construcción de $R = \mathbb{F}[x;\sigma]$ sigue de la siguiente forma:

- R es un $\mathbb{F}$-espacio vectorial a la izquierda sobre la base $\{x^n : n \geq 0\}$. Entonces, los elementos de R son polinomios a la izquierda de la forma $a_0 + a_1 x + \cdots + a_n x^n$ con $a_i \in \mathbb{F}$.
- La suma de polimios es la usual.
- El producto de R está basado en las siguientes reglas: $x^n x^m = x^{n+m}$, para $m, n \in \mathbb{N}$, y $xa = \sigma(a)x$ para $a \in \mathbb{F}$. Este producto se extiende recursivamente a R.

Ahora que hemos definido formalmente las extensiones de Ore, introduciremos varios conceptos análogos a los anillos de polinomios conmutativos comunes.

El grado $\deg(f)$ de un polinomio no nulo $f \in R$, al igual que su coeficiente líder, se definen de la manera usual, de forma que

$$f = \mathrm{lc}(f)x^{\deg(f)} + f_{\downarrow},$$

donde $f_{\downarrow}$ es un polinomio de grado menor que $\deg(f)$, y el coeficiente líder $\mathrm{lc}(f)$ es no nulo. Siguiendo las convenciones usuales para el polinomio nulo diremos que $\deg(0) = -\infty$, y que $\mathrm{lc}(0) = 0$.

Al igual que ocurre para polinomios conmutativos, directamente de la definición del producto de polinomios, dados $f, g \in R$:

$$\deg(fg) = \deg(f) + \deg(g).$$

Además, de la misma definición obtenemos también que

$$\mathrm{lc}(fg) = \mathrm{lc}(f)\sigma^{\deg(f)}(\mathrm{lc}(g)),$$

y por tanto R es un dominio de integridad no conmutativo.

El anillo R tiene algoritmos de división a la izquierda y derecha (veánse los algoritmos 1 y 2).

Algoritmo 1: División Euclídea a la izquierda

Entrada: $f, g \in \mathbb{F}[x;\sigma]$ con $g \neq 0$
Salida: $q, r \in \mathbb{F}[x;\sigma]$ tales que $f = qg + r$ y $\deg(r) < \deg(g)$
Inicialización: $q := 0, r := f$
while $\deg(g) \leq \deg(r)$ **do**
  $a = \mathrm{lc}(r)\sigma^{deg(r)-deg(g)}(\mathrm{lc}(g)^{-1})$
  $q := q + ax^{\deg(r)-\deg(g)}, r := r - ax^{deg(r)-deg(g)}g$

Algoritmo 2: División Euclídea a la derecha

Entrada: $f, g \in \mathbb{F}[x;\sigma]$ con $g \neq 0$
Salida: $q, r \in \mathbb{F}[x;\sigma]$ tales que $f = gq + r$ y $\deg(r) < \deg(g)$
Inicialización: $q := 0, r := f$
while $\deg(g) \leq \deg(r)$ **do**
  $a = \sigma^{-deg(g)}(\mathrm{lc}(g)^{-1}\,\mathrm{lc}(r))$
  $q := q + ax^{\deg(r)-\deg(g)}, r := r - gax^{deg(r)-deg(g)}$

Mostramos a continuación una demostración que justifica estos algoritmos, cuya versión original puede encontrarse en [BGTV03, Th. 4.34]

Teorema 15. *Sea $\mathbb{F}$ un cuerpo finito de q elementos siendo q una potencia de un primo, σ un autormorfismo de $\mathbb{F}$ no nulo, y $R = \mathbb{F}[x;\sigma]$ la extensión de Ore correspondiente. Entonces, dados $f, g \in R$ existen $q, r \in R$ únicos tales que:*

1. $f = qg + r$.
2. $\deg(r) < \deg(g)$.

Bajo las mismas hipótesis, existen también $q, r \in R$ únicos tales que:

1. $f = gq + r$
2. $\deg(r) < \deg(g)$.

Demostración. Para abreviar digamos $m = \deg(g), n = \deg(f)$. Veamos primero la prueba de la división a la izquierda. Si $m > n$, entonces no tenemos nada que probar, pues tomando $q = 0, r = f$ se cumple el resultado. Por otro lado, si $m \leq n$, sean $f = \sum_{i=0}^{n} a_i x^i$ y $g = \sum_{j=0}^{m} b_j x^j$, aplicaremos inducción sobre n. Si $n = 0$, entonces también $m = 0$, así que $f = a_0, g = b_0$, y por tanto tomamos $r = 0, q = a_0 b_0^{-1}$.

Por tanto, supongamos la afirmación cierta para todo f de grado menor que n. Sea $a = a_n \sigma^{n-m}(b_m^{-1})$. Entonces es claro que

$$\deg(ax^{n-m}g) = n,$$

$$\mathrm{lc}(ax^{n-m}g) = a_n.$$

Por tanto tenemos que

$$\deg(f - ax^{n-m}g) < n,$$

y por tanto, la hipótesis de inducción nos dice que existen q' y r' compliendo que $\deg(r') < \deg(g)$ y

$$f - ax^{n-m}g = q'g + r'.$$

Sea

$$q = ax^{n-m} + q',$$

entonces

$$f = ax^{n-m}g + q'g + r' = qg + r'.$$

Queda probar que q y r son únicos como tales. Supongamos que

$$f = q_1 g + r_1 = q_2 g + r_2,$$

con $\deg(r_1), \deg(r_2) < \deg(g)$. Entonces, $(q_1 - q_2)g = r_2 - r_1$, y podemos afirmar entonces que

$$\deg(q_1 - q_2) + \deg(g) = \deg((q_1 - q_2)g)$$

$$= \deg(r_2 - r_1) \leq \max(\deg(r_2), \deg(r_1)) < \deg(g).$$

Esto prueba que $\deg(q_1 - q_2) = -\infty$, demostrando que $q_1 - q_2 = 0$ y que $r_2 = r_1 = 0$ que termina la prueba de la primera parte.

La prueba de la división a la derecha es totalmente análoga, tomando $a = \sigma^{-m}(a_n b_m^{-1})$, y utilizando que $\deg(f - gax^{n-m}) < n$.

□

Los polinomios r y q obtenidos como salida del algoritmo 1 los llamaremos *resto a la izquierda* y *cociente a la izquierda*, respectivamente, de la división a la izquierda de f por g. Utilizaremos la notación $r = \mathrm{lrem}(f, g)$ y $q = \mathrm{lquo}(f, g)$. Asumimos convenciones y notaciones análogas para el algoritmo de división a la derecha.

Un polinomio $f \in R$ se dice *central* si para cualquier otro polinomio $g \in R$, se tiene que $fg = gf$.

Lema 16. *Sea $f \in R$ un polinomio central y $g, h \in R$ tales que $f = gh$. Entonces también se cumple la igualdad $f = hg$*

Demostración. Multiplicando f a la derecha por g y usando que f es central tenemos que, $ghg = fg = gf = ggh$, y por tanto $hg = gh = f$. □

2.2. Máximo común divisor y mínimo común multiplo

Veamos ahora que, como consecuencia del algoritmo de división a la izquierda, dado un ideal a la izquierda I de R, y cualquier polinomio no nulo $f \in I$ de grado mínimo, f es un generador de I. Notaremos en este caso $I = Rf$.

En efecto, para cualquier $g \in I$, utilizando el algoritmo 1 tenemos que $g = qf + r$ con $\deg(r) < \deg(f)$. Pero g y qf pertenecen a I, y por tanto r también. Como f era de grado mínimo en I, $r = 0$.

Análogamente se prueba que cualquier ideal a la derecha de R es principal. Por tanto R es un dominio de ideales principales no conmutativo.

Dados $f, g \in R$, $Rf \subset Rg$ significa que g es un *divisor a la derecha* de f, simbólicamente $g \mid_r f$, o que f es *múltiplo a la izquierda* de g.

Por ser R un dominio de ideales principales sabemos que $Rf + Rg = Rd$ para algún $d \in R$, y es inmediato comprobar que $d \mid_r f$ y $d \mid_r g$. Además, si tenemos d' con $d' \mid_r f$, $d' \mid_r g$, entonces $Rf + Rg \subset Rd'$, luego $Rd \subset Rd'$ y por tanto $d \mid_r d'$. En este caso diremos que d es un *máximo común divisor a la derecha* de f y g, estando unívocamente determinado salvo múltiplicación a la izquierda por una unidad de R. Utilizaremos la notación $d = (f, g)_r$. Además de aquí obtenemos directamente la **identidad de Bezout**.

Proposición 17 (Identidad de bezout)**.** *Sean f y g dos polinomios en R, y $d = (f, g)_r$, entonces existen $u, v \in R$ tales que $uf + vg = d$.*

Similarmente $Rf \cap Rg = Rm$ si y solo si m es un *mínimo común múltiplo a la izquierda* de f y g, notado por $m = [f, g]_l$. Este también es único salvo multiplicación a la izquierda por una unidad de R.

La versión a la derecha de todas estas definiciones y propiedades puede establecerse de forma completamente análoga.

A continuación mostramos las respectivas versiones del Algoritmo Extendido de Euclides a derecha e izquierda (algoritmos 3 y 4 respectivamente). Estas nos permiten calcular el máximo común divisor y el mínimo común multiplo tanto a izquierda como a derecha. En particular, para nuestro algoritmo principal utilizaremos la versión a la derecha de este algoritmo para obtener los coeficientes de Bezout.

La prueba del teorema sobre estos algoritmos es una adaptación de la demostración del teorema original para dominios euclideos conmutativos que se encuentra en [GT], a excepción del último resultado (el que calcula el mínimo común múltiplo) cuya demostración se encuentra en [BGTV03].

Algoritmo 3: Algoritmo extendido de Euclides a la derecha

Entrada: $f, g \in \mathbb{F}[x;\sigma]$ con $f \neq 0,\ g \neq 0$
Salida: $\{u_i, v_i, r_i\}_{i=0,\dots,h,h+1}$ tales que $r_i = fu_i + gv_i$ para todo $0 \leq i \leq h+1, r_h = (f,g)_l,$ y $fu_{h+1} = [f,g]_r$.
Inicialización:
$r_0 \leftarrow f, r_1 \leftarrow g$
$u_0 \leftarrow 1, u_1 \leftarrow 0$
$v_0 \leftarrow 0, v_1 \leftarrow 1$
$q \leftarrow 0, rem \leftarrow 0$
$i \leftarrow 1$
while $r_i \neq 0$ **do**
  $q,\ rem \leftarrow \text{rquot-rem}(r_{i-1}, r_i)$
  $r_{i+1} \leftarrow rem$
  $u_{i+1} \leftarrow u_{i-1} - u_i q$
  $v_{i+1} \leftarrow v_{i-1} - v_i q$
  $i \leftarrow i+1$
  return $\{u_i, v_i, r_i\}_{i=0,\dots,h,h+1}$

Teorema 18. *Los algoritmos 3 y 4 son correctos.*

Demostración. Realizaremos únicamente la demostración para el algoritmo 3, pues la demostración de la versión a la izquierda es análoga.

En primer lugar mencionemos que siempre que $r_i \neq 0$ se tiene que $\deg(r_{i+1}) < \deg(r_i)$, por tanto existe $h \geq 1$ tal que $r_h \neq 0$ pero $r_{h+1} = 0$.

Para cada $i \leq h$ tenemos que $r_i \neq 0$, y por tanto podemos utilizar la división a la derecha de r_{i-1} entre r_i para obtener

$$r_{i-1} = r_i q_{i+1} + r_{i+1}.$$

De aquí obtenemos que los divisores a la izquierda comunes de r_{i-1} y de r_i coinciden con los divisores a la izquierda comunes de r_i y de r_{i+1}. Luego

$$r_h = (0, r_h)_l = (r_{h+1}, r_h)_l = (r_h, r_{h-1-})_l = \cdots = (r_1, r_0)_l = (g, f)_l.$$

A continuación vamos a definir $u_i, v_i \in R$ con $i = 0, 1, \dots, h, h+1$. En primer lugar, tomamos

$$u_0 = 1,\ v_0 = 0,\ u_1 = 0,\ v_1 = 1.$$

Entonces, para $1 \leq i \leq h$ definimos

$$u_{i+1} = u_{i-1} - u_i q_{i+1}, \quad v_{i+1} = v_{i-1} - v_i q_{i+1}.$$

Algoritmo 4: Algoritmo extendido de Euclides a la izquierda

Entrada: $f, g \in \mathbb{F}[x;\sigma]$ con $f \neq 0,\ g \neq 0$
Salida: $\{u_i, v_i, r_i\}_{i=0,\dots,h,h+1}$ tales que $r_i = u_i f + v_i g$ para todo $0 \leq i \leq h+1, r_h = (f,g)_r,$ y $f u_{h+1} = [f,g]_l$.
Inicialización:
$r_0 \leftarrow f, r_1 \leftarrow g$
$u_0 \leftarrow 1, u_1 \leftarrow 0$
$v_0 \leftarrow 0, v_1 \leftarrow 1$
$q \leftarrow 0, rem \leftarrow 0$
$i \leftarrow 1$
while $r_i \neq 0$ **do**
  $q,\ rem \leftarrow \text{rquot-rem}(r_{i-1}, r_i)$
  $r_{i+1} \leftarrow rem$
  $u_{i+1} \leftarrow u_{i-1} - u_i q$
  $v_{i+1} \leftarrow v_{i-1} - v_i q$
  $i \leftarrow i+1$
  return $\{u_i, v_i, r_i\}_{i=0,\dots,h,h+1}$

Aplicaremos un argumento por inducción para comprobar que $r_i = f u_i + g v_i$. Es inmediato comprobar que para $i = 0, 1$ se cumple, así que supongamos que se cumple para $i-1,\ i$ y veamos que se cumple para $i+1$. En efecto

$$\begin{aligned} f u_{i+1} + g v_{i+1} =& f(u_{i-1} - u_i q_{i+1}) + g(v_{i-1} - v_i q_{i+1}) = \\ & f u_{i-1} + g v_{i-1} - (f u_i + g v_i) q_{i+1} = r_{i-1} - r_i q_{i+1} = r_{i+1} \end{aligned}.$$

Para concluir veamos que $u_{h+1} f = [f,g]_r$. Observemos en primer lugar que $0 = r_{h+1} = f u_{h+1} + g v_{h+1}$, luego $f u_{h+1} = -g v_{h+1}$ es un múltiplo a la derecha común de f y g. Supongamos $f u' = -g v'$ un múltiplo común a la derecha de f y g cualquiera. Entonces, definimos

$$\begin{aligned} c_0 &= -v' \\ c_1 &= u' \\ &\dots \\ c_{i+1} &= c_{i-1} - q_{i+1} c_i \quad \text{para} \quad 1 \leq i \leq h \end{aligned}.$$

Así, las siguientes igualdades se cumplen

$$\begin{aligned} r_{i+1} c_i - r_i c_{i+1} &= 0 \\ u_{i+1} c_i - u_i c_{i+1} &= (-1)^{i+1} u'. \\ v_{i+1} c_i - v_i c_{i+1} &= (-1)^{i+1} v' \end{aligned}$$

para $1 \leq i \leq h$. Para demostrarlas realizamos una inducción sencilla. Para $i = 0$, tenemos que

$$\begin{aligned} r_1c_0 - r_0c_1 &= -gv' - fu' = 0 \\ u_1c_0 - u_0c_1 &= -c_1 = -u' \\ v_1c_0 - v_0c_1 &= c_0 = -v' \end{aligned} \quad .$$

Supuestas las tres igualdades ciertas para $i-1$,

$$\begin{aligned} r_{i+1}c_i - r_ic_{i+1} &= (r_{i-1} - r_iq_{i+1})c_i - r_i(c_{i-1} - q_{i+1}c_i) = 0 \\ u_{i+1}c_i - u_ic_{i+1} &= (u_{i-1} - u_iq_{i+1})c_i - u_i(c_{i-1} - q_{i+1}c_i) = u_{i-1}c_i - u_ic_{i-1} = -((-1)^iu'). \\ v_{i+1}c_i - v_ic_{i+1} &= (v_{i-1} - v_iq_{i+1})c_i - v_i(c_{i-1} - q_{i+1}c_i) = v_{i-1}c_i - v_ic_{i-1} = -((-1)^iv') \end{aligned}$$

Ahora, por $r_{h+1} = 0 \neq r_h$, sabemos que $c_{h+1} = 0$, y por tanto u_{h+1} y v_{h+1} dividen a la izquierda a u' y v' respectivamente. Luego,

$$fu_{h+1} = -gv_{h+1} = [f, g]_r.$$

□

Veamos un lema que nos será útil posteriormente.

Lema 19. *Sean $f, g \in \mathbb{F}[x;\sigma]$ y $\{u_i, v_i, r_i\}_i = 0, \ldots, h$ los coeficientes obtenidos al aplicar el Algoritmo Extendido de Euclides a la derecha a f y g. Notemos $R_0 = \begin{pmatrix} u_0 & u_1 \\ v_0 & v_1 \end{pmatrix}$, $Q_i = \begin{pmatrix} 0 & 1 \\ 1 & -q_{i+1} \end{pmatrix}$ y $R_i = R_0Q_1 \cdots Q_i$ para cualquier $i = 0, \ldots, h$. Por tanto, para todo $i = 0, \ldots, h$ se cumplen las siguientes afirmaciones:*

1. $(fg)R_i = (r_ir_{i+1})$.
2. $R_i = \begin{pmatrix} u_i & u_{i+1} \\ v_i & v_{i+1} \end{pmatrix}$.
3. *R_i tiene inverso a izquierda y derecha.*
4. $(u_i, v_i)_r = 1$.
5. $\deg f = \deg r_{i-1} + \deg v_i$.

Demostración. Veamos primero las dos primeras afirmaciones, pues probando 2 la afirmación 1 es inmediata por la demostración del algoritmo 3. Razonando por inducción, la propiedad es cierta para $i = 0$, así que supongamos que fijo i lo es para $i-1$. Entonces tenemos que

$$R_i = \begin{pmatrix} u_{i-1} & u_i \\ v_{i-1} & v_i \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ 1 & -q_{i+1} \end{pmatrix} = \begin{pmatrix} u_i & u_{i-1} - u_iq_{i+1} \\ v_i & v_{i-1} - v_iq_{i+1} \end{pmatrix} = \begin{pmatrix} u_i & u_{i+1} \\ v_i & v_{i+1} \end{pmatrix},$$

quedando demostrados las afirmaciones 1 y 2. De nuevo si probamos 3 obtenemos 4 inmediatamente. Observemos que

$$T_i = \begin{pmatrix} q_{i+1} & 1 \\ 1 & 0 \end{pmatrix}$$

es una inversa a la izquierda y derecha de Q_i. Por tanto, $S_i = T_i \cdots T_1 R_0$ es una inversa a izquierda y derecha de R_i y 3 y 4 quedan probadas.

Veamos 5. Para $i = 1$, $r_0 = f$ y $v_1 = 1$ cumpliéndose la igualdad, así que razonemos por inducción. Mencionemos que $\deg r_i < \deg r_{i-1}$ y que $\deg v_{i-1} < \deg v_i$ para cualquier $i > 1$. Entonces, como $r_{i+1} = r_{i-1} - r_i q_{i+1}$ y $v_{i+1} = v_{i-1} - v_i q_{i+1}$ para cualquier i,

$$\deg r_{i-1} = \deg r_i + \deg q_{i+1}$$
$$\deg v_{i+1} = \deg v_i + \deg q_{i+1}.$$

Ahora, por la hipótesis de inducción, $\deg f = \deg r_{i-1} + \deg v_i = \deg r_i + \deg q_{i+1} + \deg v_{i+1} - \deg q_{i+1} = \deg r_i + \deg v_{i+1}$ □

Ahora, con el objetivo de poder definir la evaluación de nuestro polinomios, dado $j \geq 0$, definimos la *norma j-ésima* para cualquier $\gamma \in \mathbb{F}$ de forma recursiva como sigue:

$$N_0(\gamma) = 1$$
$$N_{j+1}(\gamma) = \gamma\sigma(N_j(\gamma)) = \gamma\sigma(\gamma)\ldots\sigma^j(\gamma).$$

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

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

En la siguiente proposición se muestran dos propiedades de esta norma que nos serán de utilidad más adelante.

Proposición 20. *Sean $\gamma, \alpha, \beta \in \mathbb{F}$ tales que $\beta = \alpha^{-1}\sigma(\alpha)$, entonces*

$$N_i(\sigma^k(\gamma)) = \sigma^k(N_i(\gamma)),$$
$$N_i(\sigma^k(\beta)) = \sigma^k(\alpha)^{-1}\sigma^{k+i}(\alpha).$$

Demostración. La primera igualdad se prueba de forma inmediata usando la propia definición de la norma, pues

$$\begin{aligned} N_i(\sigma^k(\gamma)) &= \sigma^k(\gamma)\sigma(\sigma^k(\gamma))\cdots\sigma^{i-1}(\sigma^k(\gamma)) \\ &= \sigma^k(\gamma)\sigma^k(\sigma(\gamma))\cdots\sigma^k(\sigma^{i-1}(\gamma)) = \sigma^k(N_i(\gamma)). \end{aligned}$$

Para la segunda igualdad, utilizando la igualdad anterior y de nuevo la definición de norma tenemos que

$$\begin{aligned} N_i(\sigma^k(\beta)) &= \sigma^k(N_i(\beta)) = \sigma^k\left(\alpha^{-1}\sigma(\alpha)\sigma(\alpha^{-1})\sigma^2(\alpha)\sigma^2(\alpha^{-1})\ldots\sigma^i(\alpha)\right) \\ &= \sigma^k(\alpha)^{-1}\sigma^{k+i}(\alpha). \end{aligned}$$

□

La *evaluación la izquierda* de un polinomio no conmutativo $f \in R$ en un $a \in \mathbb{F}$ se define como el resto de la división a la derecha de f por $x - a$, y de forma análoga para la *evaluación a la derecha*. Estas evaluaciones nos permiten hablar de raíces a izquierda y derecha de estos polinomios. Las propiedades en general de estas son estudiadas en [LL88], donde se encuentra en esencia la prueba que presentamos del siguiente lema.

Lema 21. *Sea $\gamma \in \mathbb{F}$ y $f = \sum_{i=0}^{n} f_i x^i \in R$. Entonces:*

1. *El resto de la división a la izquierda de f por $x - \gamma$ es $\sum_{i=0}^{n} f_i N_i(\gamma)$.*
2. *El resto de la división a la derecha de f por $x - \gamma$ es $\sum_{i=0}^{n} \sigma^{-i}(f_i) N_{-i}(\gamma)$.*

Demostración. Para demostrar *i)* observamos primero un caso especial de este resultado:

$$x^j - N_j(\gamma) \in R(x - \gamma) \ \forall j \geq 0.$$

Es evidente que el resultado es cierto para $j = 0$, asi que procedemos por inducción sobre j. Supongamos el resultado cierto para j, entonces

$$\begin{aligned} x^{j+1} - N_{j+1}(\gamma) &= x^{j+1} - \sigma(N_j(\gamma))\gamma \\ &= x^{j+1} + \sigma(N_j(\gamma))(x - \gamma) - \sigma(N_j(\gamma))x \\ &= x^{j+1} + \sigma(N_j(\gamma))(x - \gamma) - xN_j(\gamma) \\ &= \sigma(N_j(\gamma))(x - \gamma) + x(x^j - N_j(\gamma)) \in R(x - \gamma) \end{aligned}$$

Utilizando (2.2) tenemos entonces que

$$f - \sum_{i=0}^{n} f_i N_i(\gamma) = \sum_{i=0}^{n} f_i (x^i - N_i(\gamma)) \in R(x - \gamma)$$

y, por la unicidad del resto de la división euclídea tenemos que $r = \sum_{i=0}^{n} f_i N_i(\gamma)$.

Para la siguiente afirmación procedemos de forma similar. Veamos en primer lugar que

$$x^j - N_{-j}(\gamma) \in (x - \gamma)R \ \forall j \geq 0.$$

De nuevo, es obvio para $j = 0$, por tanto procedemos por inducción supuesto cierto para j.

$$\begin{aligned} x^{j+1} - N_{-j-1}(\gamma) &= x^{j+1} - \gamma\sigma^{-1}(N_j(\gamma)) \\ &= x^{j+1} + (x - \gamma)\sigma^{-1}(N_{-j}(\gamma)) - x\sigma^{-1}(N_{-j}(\gamma)) \\ &= x^{j+1} + (x - \gamma)\sigma^{-1}(N_{-j}(\gamma)) - N_{-j}(\gamma)x \\ &= (x - \gamma)\sigma^{-1}(N_{-j}(\gamma)) + (x^j - N_{-j}(\gamma))x \in (x - \gamma)R \end{aligned}$$

Así, usando (2.2) vemos que

$$f - \sum_{i=0}^{n} \sigma^{-i}(f_i) N_{-i}(\gamma) = \sum_{i=0}^{n} x^i \sigma^{-i}(f_i) - N_{-i}(\gamma)\sigma^{-i}(f_i)$$

$$= \sum_{i=0}^{n} (x^i - N_{-i}(\gamma))\sigma^{-i}(f_i) \in (x-\gamma)R.$$

Así que por la unicidad del resto queda probado 2. □

3 Algoritmo para códigos Reed-Solomon sesgados

Ya disponemos de la conceptos necesarios para construir los códigos cíclicos sesgados, que serán los que utilizará el algoritmo de decodificación principal. En este capítulo comenzaremos definiendo estos códigos, junto con los resultados que requiere nuestro algoritmo. Una vez hecho esto presentaremos dicho algoritmo, y acabaremos mostrando como solucionar un error que puede ocurrir durante la decodificación y discutiendo con qué frecuencia ocurren este tipo de errores. El desarrollo que hemos seguido se encuentra basado en [GTLN17].

3.1. Introducción

A lo largo de este capítulo seguiremos la notación introducida en el capítulo anterior, por tanto $\mathbb{F}$ será un cuerpo cualquiera, σ un autormorfismo de $\mathbb{F}$ de orden finito n, y $R = \mathbb{F}[x;\sigma]$ el anillo de polinomio sesgados correspondiente.

Para introducir los nuevos códigos, intentaremos seguir parte del razonamiento que realizamos en la sección 1.4, pero ahora trabajando sobre un anillo de polinomios sesgados en lugar del caso conmutativo usual. En primer lugar, utilizaremos la identificación análoga a la descrita en 1.4 entre vectores $c = c_0c_1\cdots c_{n-1}$ de $\mathbb{F}^n$ y polinomios $c(x) = c_0 + c_1x + \cdots + c_{n-1}x^{n-1}$ de $\mathbb{F}[x;\sigma]$.

El primer paso es construir el cociente $F[x;\sigma]/\langle x^n - 1\rangle$, y para que este sea un anillo necesitamos que ver que el polinomio $x^n - 1$ es central en R. Efectivamente, dado $f = f_mx^m + \ldots + f_1x + f_0 \in \mathbb{F}$, tenemos que

$$\begin{aligned}(x^n-1)f &= x^nf_mx^m - f_mx^m + \cdots + x^nf_1x - f_1x + x^nf_0 - f_0\\ &= \sigma^n(f_m)x^{n+m} - f_mx^m + \cdots + \sigma^n(f_1)x^{n+1} - f_1 + \sigma^n(f_0)x^n - f_0\\ &= f_mx^{n+m} - f_mx^m + \cdots + f_1x^{n+1} - f_1 + f_0x^n - f_0\\ &= f(x^n-1).\end{aligned}$$

Entonces el ideal por la izquierda generado por $x^n - 1$ es también un ideal por la derecha, así que podemos considerar el anillo cociente $\mathcal{R} = \mathbb{F}[x;\sigma]/\langle x^n - 1\rangle$.

De forma análoga al caso de los códigos cíclicos usuales, cada ideal a la izquierda $I \leq \mathcal{R}$ define un código $\mathcal{C} = \mathfrak{v}(I)$ de longitud n, donde $\mathfrak{v} : \mathcal{R} \to \mathbb{F}^n$ es el mapa de coordenadas asociado a la base $\mathcal{B} = \{1, x, \ldots, x^{n-1}\}$. Así, la longitud del código coincide con el orden de σ. Como es usual en teoría de códigos, representaremos los elementos de $\mathcal{R}$ por polinomios de grado menor que n. De esta forma, cuando

escribamos una palabra código como $c(x)$, técnicamente nos referimos a la clase $c(x) + R(x^n - 1)$ en $\mathcal{R}$. De aquí en adelante suponemos establecidas estas condiciones. Llamaremos a cualquier código de este tipo *código cíclico sesgado*, o CCS para abreviar.

Los códigos cíclicos sesgados también tienen una propiedad de ciclicidad similar a la que tienen los códigos cíclicos usuales. Sea $\mathcal{C}$ un código cíclico sesgado, y $c = c_0 c_1 \cdots c_{n-1} \in \mathcal{C}$ una palabra código cualquiera, entonces

$$c' = \sigma(c_{n-1})\sigma(c_0) \cdots \sigma(c_{n-2}) \in \mathcal{C}.$$

A continuación demostramos un teorema con varias propiedades que necesitaremos más adelante sobre los códigos cíclicos sesgados. El teorema es una adaptación al caso no conmutativo de un resultado del capítulo 4 de [HP03], pero la demostración se ha realizado utilizando argumentos de módulos más generales.

Teorema 22. *Sea $\mathcal{C}$ un código cíclico sesgado en $\mathcal{R}$. Entonces existe un polinomio $g \in \mathcal{C}$ con las siguientes propiedades:*

1. *$\mathcal{C}$ esta generado como ideal a la izquierda en R por g.*
2. *g divide a izquierda y a derecha a $(x^n - 1)$.*
3. *la dimensión de $\mathcal{C}$ es $k = n - \deg g$.*
4. *Sea $g = \sum_{i=0}^{n-k} g_i x^i$. Entonces*

$$G = \begin{bmatrix} g_0 & g_1 & \cdots & g_{n-k} & & 0 \\ 0 & \sigma(g_0) & \cdots & \sigma(g_{n-k-1}) & \sigma(g_{n-k}) & \\ & \cdots & \cdots & \cdots & \cdots & \\ 0 & & \sigma^{k-1}(g_0) & \cdots & \sigma^{k-1}(g_{n-k-1}) & \sigma^{k-1}(g_{n-k}) \end{bmatrix}$$

$$\leftrightarrow \begin{bmatrix} g & & & \\ & xg & & \\ & & \cdots & \\ & & & x^{k-1}g \end{bmatrix}$$

es una matrix generadora de $\mathcal{C}$.

Demostración. Por el tercer teorema del isomorfismo de Noether para R-módulos, tenemos que existe un R-submódulo Rg de R (R era un dominio de ideales principales) tal que $\langle x^n - 1\rangle \subseteq Rg \subset R$ y $\mathcal{C} = Rg/\langle x^n - 1\rangle$. Esto nos dice que $g \mid_r x^n - 1$, pero como $x^n - 1$ es central, usando el lema 16 tenemos que $g \mid_l x^n - 1$, probando los apartados 1 y 2.

Ahora, por el mismo teorema, utilizando el resultado de este análogo al tercer teorema de isomorfía

$$R/Rg \cong (R/\langle x^n - 1\rangle)/(Rg/\langle x^n - 1\rangle)$$

como R-módulos, y por tanto también como $\mathbb{F}$-espacios vectoriales. Entonces, como la dimensión de R/Rg como $\mathbb{F}$-espacio vectorial es $\deg g$, la dimensión de $\mathcal{C}$ es $n - \deg g = k$, probando 3. Para concluir, el conjunto $\{x^i g \mid 0 \leq i \leq k\}$ genera $\mathcal{C}$, y por tanto se cumple 4. □

Dado un código cíclico sesgado $\mathcal{C}$, al polinomio $g(x)$ que nos proporciona el teorema anterior lo llamaremos el *polinomio generador de* $\mathcal{C}$. Entonces, gracias a este teorema, ya podemos codificar mensajes. Ya sea, multiplicando el polinomio asociado a dicho mensaje por el polinomio generador $g(x)$, o de la forma usual utilizando la matriz generatriz.

Al ser $\mathcal{R}$ un dominio de ideales principales sabíamos, ya que todo código cíclico sesgado estaba generado por al menos un polinomio. Ahora, el teorema anterior nos dice que además está generado por un divisor a la derecha de $x^n - 1$. Por esto nos será útil estudiar la descomposición en factores de $x^n - 1$.

3.2. Códigos Reed-Solomon Sesgados

Nuestro siguiente objetivo será proporcionar un método sistemático para contruir CCSs de una determinada distancia Hamming. Debido a la analogía con los códigos Reed-Solomon, los llamaremos *códigos Reed-Solomon sesgados* o *códigos skew RS* para abreviar. El siguiente resultado, que es un caso particular de [LL88, Corolario 4.13], será importante en resultados posteriores. A continuación mostramos solo una prueba elemental de este.

Lema 23. *Sea L un cuerpo, σ un automorfismo de L de orden finito n, y $K = L^\sigma$ el subcuerpo invariante bajo σ. Sea $\{\alpha_0, \ldots, \alpha_{n-1}\}$ una K-base de L. Entonces, para todo $t \leq n$, y cada subconjunto $k_0 < k_1 < \ldots < k_{t-1} \subset \{0, 1, \ldots, n-1\}$*

$$\begin{vmatrix} \alpha_{k_0} & \alpha_{k_1} & \ldots & \alpha_{k_{t-1}} \\ \sigma(\alpha_{k_0}) & \sigma(\alpha_{k_1}) & \ldots & \sigma(\alpha_{k_{t-1}}) \\ \vdots & \vdots & \ddots & \vdots \\ \sigma^{t-1}(\alpha_{k_0}) & \sigma^{t-1}(\alpha_{k_1}) & \ldots & \sigma^{t-1}(\alpha_{k_{t-1}}) \end{vmatrix} \neq 0.$$

Demostración. Realizaremos la prueba por inducción sobre t. El caso $t = 1$ se cumple trivialmente. Por tanto, supongamos que el lema se cumple para un cierto $t \geq 1$. Tenemos que comprobar que, para toda matriz $(t+1) \times (t+1)$

$$\Delta = \begin{pmatrix} \alpha_{k_0} & \alpha_{k_1} & \ldots & \alpha_{k_t} \\ \sigma(\alpha_{k_0}) & \sigma(\alpha_{k_1}) & \ldots & \sigma(\alpha_{k_t}) \\ \vdots & \vdots & \ddots & \vdots \\ \sigma^t(\alpha_{k_0}) & \sigma^t(\alpha_{k_1}) & \ldots & \sigma^t(\alpha_{k_t}) \end{pmatrix}.$$

el determinante $|\Delta|$ es distinto de cero. Supongamos por el contrario que $|\Delta| = 0$. Por la hipótesis de inducción tenemos que las primeras t columnas de Δ son linealmente independientes, luego existen $a_0, \ldots, a_{t-1} \in L$ tales que

$$(\alpha_{k_t}, \sigma(\alpha_{k_t}), \ldots, \sigma^t(\alpha_{k_t})) = \sum_{j=0}^{t-1} a_j(\alpha_{k_j}, \sigma(\alpha_{k_j}), \ldots, \sigma^t(\alpha_{k_j})).$$

Es decir, $a_0, \ldots, a_{t-1}$ satisfacen el sistema lineal

$$\begin{cases} \alpha_{k_t} = a_0\alpha_{k_0} + \cdots + a_{t-1}\alpha_{k_{t-1}} \\ \sigma(\alpha_{k_t}) = a_0\sigma(\alpha_{k_0}) + \cdots + a_{t-1}\sigma(\alpha_{k_{t-1}}) \\ \vdots \\ \sigma^t(\alpha_{k_t}) = a_0\sigma^t(\alpha_{k_0}) + \cdots + a_{t-1}\sigma^t(\alpha_{k_{t-1}}) \end{cases}. \tag{3.1}$$

Para cada $j = 0, \ldots, t-1$, restamos en (3.1) la ecuación $j+1$ transformada por σ^{-1} a la ecuación j. Esto produce el siguiente sistema lineal homogéneo

$$\begin{cases} 0 = (a_0 - \sigma^{-1}(a_0))\alpha_{k_0} + \cdots + (a_{t-1} - \sigma^{-1}(a_{t-1}))\alpha_{k_{t-1}} \\ 0 = (a_0 - \sigma^{-1}(a_0))\sigma(\alpha_{k_0}) + \cdots + (a_{t-1} - \sigma^{-1}(a_{t-1}))\sigma(\alpha_{k_{t-1}}) \\ \vdots \\ 0 = (a_0 - \sigma^{-1}(a_0))\sigma^{t-1}(\alpha_{k_0}) + \cdots + (a_{t-1} - \sigma^{-1}(a_{t-1}))\sigma^{t-1}(\alpha_{k_{t-1}}) \end{cases}.$$

Por la hipótesis de inducción la matriz de coeficientes de (3.2) es no singular, de forma que para todo $j = 0, \ldots, t-1$, tenemos que $a_j - \sigma^{-1}(a_j) = 0$, y por tanto $a_0, \ldots, a_{t-1} \in K$. Como consecuencia, la ecuación (3.1) establece una dependencia lineal sobre K de la K-base $\{\alpha_0, \ldots, \alpha_{n-1}\}$, creando una contradicción. Por tanto, $|\Delta| \neq 0$ y el resultado queda demostrado. □

A continuación comenzaremos a estudiar los divisores a la derecha de $x^n - 1$, empezando por dar un método para hallar divisores a la derecha lineales.

Proposición 24. *Sea $\beta \in \mathbb{F}$ tal que $\beta = \sigma(c)c^{-1}$ para algún $c \in \mathbb{F}$. Entonces $x - \beta$ divide por la derecha a $x^n - 1$.*

Demostración. Por la proposición 20 tenemos que $N_n(\beta) = \sigma^0(c^{-1})\sigma^n(c) = c^{-1}c = 1$, y por el lema 21, el resto de la división a la izquierda de $x^n - 1$ por $x - \beta$ es

$$N_n(\beta) - N_0(\beta) = 1 - 1 = 0,$$

finalizando la prueba. □

Observación 25. En realidad, el enunciado de la proposición es un si y solo si, de forma que los únicos divisores lineales de $x^n - 1$ son de la forma $x - \beta$ para un $\beta \in \mathbb{F}$ con $\beta = \sigma(c)c^{-1}$. Este resultado se encuentra enunciado y demostrado en [GTLN16], aunque no se ha incluido pues solo necesitamos la implicación que mostramos.

Lema 26. *Sea $\alpha \in \mathbb{F}$ tal que $\{\alpha, \sigma(\alpha), \ldots, \sigma^{n-1}(\alpha)\}$ sea una base de $\mathbb{F}$ como $\mathbb{F}^\sigma$-espacio vectorial. Fijemos $\beta = \alpha^{-1}\sigma(\alpha)$. Para todo subconjunto $T = \{t_1 < t_2 < \cdots < t_m\} \subset \{0, 1, \ldots, n-1\}$, los polinomios*

$$g^l = [x - \sigma^{t_1}(\beta), x - \sigma^{t_2}(\beta), \ldots, x - \sigma^{t_m}(\beta)]_l$$

y

$$g^r = [x - \sigma^{t_1}(\beta^{-1}), x - \sigma^{t_2}(\beta^{-1}), \ldots, x - \sigma^{t_m}(\beta^{-1})]_r$$

tienen grado m. Además, si $x - \sigma^s(\beta) \mid_r g^l$ o $x - \sigma^s(\beta^{-1}) \mid_l g^r$, entonces $s \in T$.

Demostración. Supongamos que $\deg g^l < m$, así que $g^l = \sum_{i=0}^{m-1} g_i x^i$. Como g es un múltiplo a la izquierda de $x - \sigma^{t_j}(\beta)$ para todo $1 \leq j \leq m$, por el lema 21 tenemos que

$$\sum_{i=0}^{m-1} g_i N_i(\sigma^{t_j}(\beta)) = 0 \text{ para todo } 1 \leq j \leq m$$

Esto es un sistema lineal homogéneo cuya matriz de coeficientes es la traspuesta de la matriz M dada por

$$\begin{pmatrix} N_0(\sigma^{t_1}(\beta)) & N_0(\sigma^{t_2}(\beta)) & \ldots & N_0(\sigma^{t_m}(\beta)) \\ N_1(\sigma^{t_1}(\beta)) & N_1(\sigma^{t_2}(\beta)) & \ldots & N_1(\sigma^{t_m}(\beta)) \\ \vdots & \vdots & \ddots & \vdots \\ N_{m-1}(\sigma^{t_1}(\beta)) & N_{m-1}(\sigma^{t_2}(\beta)) & \ldots & N_{m-1}(\sigma^{t_m}(\beta)) \end{pmatrix}.$$

Fijémonos que $N_i(\sigma^{t_j}(\beta)) = \sigma^{t_j}(N_i(\beta)) = \sigma^{t_j}(N_i(\alpha^{-1}\sigma(\alpha))) = \sigma^{t_j}(\alpha^{-1})\sigma^{t_{j+i}}(\alpha)$ para todo $1 \leq j \leq m$ y $0 \leq i \leq m-1$. Por tanto, $|M| = 0$ si y solo si el determinante de la matriz M',

$$\begin{pmatrix} \sigma^{t_1}(\alpha) & \sigma^{t_2}(\alpha) & \cdots & \sigma^{t_m}(\alpha) \\ \sigma^{t_1+1}(\alpha) & \sigma^{t_2+1}(\alpha) & \cdots & \sigma^{t_m+1}(\alpha) \\ \vdots & \vdots & \ddots & \vdots \\ \sigma^{t_1+m-1}(\alpha) & \sigma^{t_2+m-1}(\alpha) & \cdots & \sigma^{t_m+m-1}(\alpha) \end{pmatrix},$$

o equivalentemente

$$\begin{pmatrix} \sigma^{t_1}(\alpha) & \sigma^{t_2}(\alpha) & \cdots & \sigma^{t_m}(\alpha) \\ \sigma(\sigma^{t_1}(\alpha)) & \sigma(\sigma^{t_2}(\alpha)) & \cdots & \sigma(\sigma^{t_m}(\alpha)) \\ \vdots & \vdots & \ddots & \vdots \\ \sigma^{m-1}(\sigma^{t_1}(\alpha)) & \sigma^{m-1}(\sigma^{t_2+m-1}(\alpha)) & \cdots & \sigma^{m-1}(\sigma^{t_m}(\alpha)) \end{pmatrix},$$

es cero. Sin embargo, por el lema 23, $|M'| \neq 0$, luego la única solución del sistema lineal (3.2) es $g_0 = \cdots = g_{m-1} = 0$, siendo una contradicción. Por tanto $\deg g^l = m$.

Para el otro polinomio razonamos de forma análoga. Si $\deg g^r < m$ y $g^r = \sum_{i=0}^{m-1} g_i x^i$, obtenemos el sistema lineal

$$\sum_{i=0}^{m-1} \sigma^{-i}(g_i) N_{-i}(\sigma^{t_j}(\beta^{-1})) = 0 \text{ para todo } 1 \leq j \leq m.$$

Vemos que $N_{-i}(\sigma^{t_j}(\beta^{-1})) = \sigma^{t_j}(\alpha^{-1})\sigma^{t_j-i+1}(\alpha)$ para $0 \leq i \leq m-1$ y $1 \leq j \leq m$. Entonces, de nuevo por el lema 23 el sistema tiene una única solución $\sigma^{-i}(g_i) = 0$ para $0 \leq i \leq m-1$, luego $g_0 = g_1 = \cdots = g_{m-1} = 0$. Como dijimos esto es una contradicción y por tanto $\deg g^r = m$. □

Recordemos que el teorema de la base normal nos asegura la existencia de un elemento α que cumpla las condiciones del teorema anterior.

Corolario 27. *Sean $\alpha, \beta \in \mathbb{F}$ verificando las condiciones del lema 26. Entonces*

$$x^n - 1 = [x - \beta, x - \sigma(\beta), \ldots, x - \sigma^{n-1}(\beta)]_l = [x - \beta, x - \sigma(\beta), \ldots, x - \sigma^{n-1}(\beta)]_r.$$

Demostración. Por ser $x^n - 1$ central, el lema 16 nos dice que cualquier divisor a la izquierda lo es también a la derecha, y por tanto nos basta con probar solo una de las dos igualdades.

Para ver la primera igualdad, por el lema 26, solo necesitamos ver que $x - \sigma^k(\beta)$ divide a $x^n - 1$ para todo $0 \leq k \leq n-1$. Realizando un argumento similar al que hicimos para la prueba de las proposición 24, tenemos que, por la proposición 20,

$$N_n(\sigma^k(\beta)) = \sigma^k(\alpha^{-1})\sigma^{k+n}(\alpha) = \sigma^k(\alpha^{-1})\sigma^k(\alpha) = 1.$$

Entonces, por el lema 21, el resto de dividir $x^n - 1$ por $x - \sigma^k(\beta)$ es

$$N_n(\sigma^k(\beta)) - N_0(\sigma^k(\beta)) = 1 - 1 = 0,$$

concluyendo la demostración de la primera igualdad, y por tanto del corolario. □

Con esto, ya tenemos los resultados suficientes para definir los códigos sobre los que estarán definidos nuestros algoritmos.

Definición 28. Sean $\alpha, \beta \in \mathbb{F}$ verificando las condiciones del lema 26. Un código Reed-Solomon (RS) sesgado de distancia fijada $\delta \leq n$ es un CCS generado por

$$[x - \sigma^r(\beta), x - \sigma^{r+1}(\beta), \cdots, x - \sigma^{r+\delta-2}(\beta)]_l,$$

para algún $r \geq 0$.

Teorema 29. *Sea $\mathcal{C}$ un código skew RS de distancia fijada δ. La distancia Hamming de $\mathcal{C}$ es δ.*

Demostración. Definamos en primer lugar

$$g = [x - \sigma^r(\beta), x - \sigma^{r+1}(\beta), \ldots, x - \sigma^{r+\delta-2}(\beta)]_l$$

un generador de $\mathcal{C}$ como ideal a la izquierda de $\mathcal{R}$. Entonces, una matriz de paridad H de $\mathcal{C}$ es

$$\begin{pmatrix} N_0(\sigma^r(\beta)) & N_1(\sigma^r(\beta)) & \cdots & N_{n-1}(\sigma^r(\beta)) \\ N_0(\sigma^{r+1}(\beta)) & N_1(\sigma^{r+1}(\beta)) & \cdots & N_{n-1}(\sigma^{r+1}(\beta)) \\ \vdots & \vdots & \ddots & \vdots \\ N_0(\sigma^{r+\delta-2}(\beta)) & N_1(\sigma^{r+\delta-2}(\beta)) & \cdots & N_{n-1}(\sigma^{r+\delta-2}(\beta)) \end{pmatrix}.$$

pues sus filas dan la evaluación a la derecha de las raíces de g. Entonces, por el corolario 9, nos basta con probar no existe ningún conjunto de $\delta - 1$ columnas linealmente dependientes. Para ello procedemos de forma similar a la demostración del lema 26. Como ya utilizamos antes, $N_i(\sigma^k(\beta)) = \sigma^k(N_i(\beta)) = \sigma^k(\alpha^{-1})\sigma^{i+k}(\alpha)$ para cualesquiera enteros i y k. Por tanto, dado cualquier conjunto de columnas de tamaño $\delta - 1$, podemos verlo como la matriz M

$$\begin{pmatrix} N_{k_1}(\sigma^r(\beta)) & N_{k_2}(\sigma^r(\beta)) & \cdots & N_{k_{\delta-1}}(\sigma^r(\beta)) \\ N_{k_1}(\sigma^{r+1}(\beta)) & N_{k_2}(\sigma^{r+1}(\beta)) & \cdots & N_{k_{\delta-1}}(\sigma^{r+1}(\beta)) \\ \vdots & \vdots & \ddots & \vdots \\ N_{k_1}(\sigma^{r+\delta-2}(\beta)) & N_{k_2}(\sigma^{r+\delta-2}(\beta)) & \cdots & N_{k_{\delta-1}}(\sigma^{r+\delta-2}(\beta)) \end{pmatrix},$$

con $\{k_1 < k_2 < \cdots < k_{\delta-1}\} \subset \{0, 1, \ldots, n-1\}$. Ahora, $|M| = 0$, si y solo $|M'| = 0$, donde M' es la matriz

$$\begin{pmatrix} \sigma^{k_1+r}(\alpha) & \sigma^{k_2+r}(\alpha) & \cdots & \sigma^{k_{\delta-1}+r}(\alpha) \\ \sigma^{k_1+r+1}(\alpha) & \sigma^{k_2+r+1}(\alpha) & \cdots & \sigma^{k_{\delta-1}+r+1}(\alpha) \\ \vdots & \vdots & \ddots & \vdots \\ \sigma^{k_1+r+\delta-2}(\alpha) & \sigma^{k_2+r+\delta-2}(\alpha) & \cdots & \sigma^{k_{\delta-1}+r+\delta-2}(\alpha) \end{pmatrix}$$

$$= \begin{pmatrix} \sigma^{k_1+r}(\alpha) & \sigma^{k_2+r}(\alpha) & \cdots & \sigma^{k_{\delta-1}+r}(\alpha) \\ \sigma(\sigma^{k_1+r}(\alpha)) & \sigma(\sigma^{k_2+r})(\alpha) & \cdots & \sigma(\sigma^{k_{\delta-1}+r}(\alpha)) \\ \vdots & \vdots & \ddots & \vdots \\ \sigma^{\delta-2}(\sigma^{k_1+r}(\alpha)) & \sigma^{\delta-2}(\sigma^{k_2+r}(\alpha)) & \cdots & \sigma^{\delta-2}(\sigma^{k_{\delta-1}+r}(\alpha)) \end{pmatrix}.$$

Por ser $\{\alpha, \sigma(\alpha), \ldots, \sigma^{n-1}(\alpha)$ una base de la extensión $\mathbb{F}^\sigma \subset \mathbb{F}$, por el lema 23, $|M'| \neq 0$, y por tanto esas columnas son linealmente independientes. □

3.3. Algoritmo Principal

De aquí en adelante $\mathcal{C}$ denotará un código skew RS de distancia fijada δ generado, como ideal a la izquierda de $\mathcal{R}$, por $g = [x - \sigma^r(\beta), x - \sigma^{r+1}(\beta), \ldots, x - \sigma^{r+\delta-2}(\beta)]_l$,

para algún $r \geq 0$, donde β lo elegimos como en la definición 28. Sabemos que la distancia mínima de $\mathcal{C}$ es exactamente δ por el teorema 29. Sea $\tau = \lfloor (\delta - 1)/2 \rfloor$ que es el máximo número de errores que nuestro código puede corregir. Por simplicidad, supondremos que $r = 0$. Esto no es una restricción, pues siempre podemos escribir $\beta' = \sigma^r(\beta)$. Entonces, $\beta' = (\alpha')^{-1}\sigma(\alpha')$, donde $\alpha' = \sigma^r(\alpha)$, y es claro que α' también proporciona una base normal. Por tanto, $g = [x - \beta', x - \sigma(\beta'), \ldots, x - \sigma^{\delta-2}(\beta')]$.

Sea $c \in \mathcal{C}$ una palabra que es transmitida a través de un canal binario simétrico y el polinomio $y = c + e$ es recibido, donde $e = e_1 x^k + \cdots + e_\nu x^{k_\nu}$ con $\nu \leq \tau$. Definimos el polinomio *localizador de errores* como

$$\lambda = \left[1 - \sigma^{k_1}(\beta)x, 1 - \sigma^{k_2}(\beta)x, \ldots, 1 - \sigma^{k_\nu}(\beta)x\right]_r.$$

En primer lugar mostraremos que λ determina las posiciones con un error no nulo.

Lema 30. *Para cualquier subconjunto* $\{t_1, \ldots, t_m\} \subset \{0, 1, \ldots, n-1\}$,

$$\left[1 - \sigma^{t_1}(\beta)x, \ldots, 1 - \sigma^{t_m}(\beta)x\right]_r = \left[x - \sigma^{t_1 - 1}(\beta^{-1}), \ldots, x - \sigma^{t_m - 1}(\beta^{-1})\right]_r.$$

Demostración. Para cualquier $a \in \mathbb{F}$,

$$1 - ax = (x - \sigma^{-1}(a^{-1}))(-\sigma^{-1}(a)), \quad x - \sigma^{-1}(a^{-1}) = (1 - ax)(-\sigma^{-1}(a^{-1})).$$

Entonces, los polinomios del resultado se dividen a la izquierda mutuamente, y por tanto ambos mínimos comunes múltiplos coinciden. □

Proposición 31. $1 - \sigma^d(\beta)x$ *divide a la izquierda a* λ *si y solo si* $x - \sigma^{d-1}(\beta^{-1})$ *divide a la izquierda a* λ *si y solo si* $d \in \{k_1, \ldots, k_\nu\}$.

Demostración. Por el lema anterior, $1 - \sigma^d(\beta)x$ divide a la izquierda λ si y solo si $x - \sigma^{d-1}(\beta^{-1})$ divide a la izquierda a λ. Ahora, por el lema 26, $x - \sigma^{d-1}(\beta^{-1})$ es un divisor a la izquierda de λ si y solo si $d \in \{k_1, \ldots, k_\nu\}$. □

Por tanto, una vez que λ es conocido, las coordenadas del error pueden ser localizadas siguiendo la siguiente regla: $d \in 0, 1, \ldots, n-1$ es una posición de error si y solo si $\sigma^{d-1}(\beta - 1)$ es una raíz a la izquierda de λ. En realidad, λ puede ser sustituido por cualquier otro polinomio en R asociado a la derecha a λ, es decir, cualquier polinomio que difiera de λ en la multiplicación a la derecha por un elemento no nulo de $\mathbb{F}$.

Ahora, para cualquier $1 \leq j \leq \nu$ existe $p_j \in R$ tal que $\lambda = (1 - \sigma^{k_j}(\beta)x)p_j$ con $\deg p_j = \nu - 1$. Definimos entonces el polinomio *evaluador de errores* como $\omega = \sum_{j=1}^{\nu} e_j \sigma^{k_j}(\alpha) p_j$. Así, si conocemos el polinimio localizador de errores λ y el polinomio evaluador de errores ω, podremos calcular los valores $e_1, e_2, \ldots, e_\nu$ resolviendo un sistema lineal dado por $\omega = \sum_{j=1}^{\nu} e_j \sigma^{k_j}(\alpha) p_j$. Además, es directo comprobar que $\deg \omega < \nu$, pues, como ya dijimos, $\deg(p_j) = \nu - 1$ para todo $1 \leq j \leq \nu$.

Finalmente, para cada $0 \leq i \leq n-1$, el i-ésimo síndrome S_i del polinomio recibido $y = \sum_{j=0}^{n-1} y_j x^j$ se define como el resto de la división a la izquierda de y por $x - \sigma^i(\beta)$. Este S_i es la evaluación a la derecha de y en $\sigma^i(\beta)$. Siempre que $0 \leq i \leq 2\tau - 1$, las evaluaciónes a la derecha en c son cero, y por tanto, por el lema 21 tenemos:

$$\begin{aligned} S_i &= \sum_{j=0}^{n-1} y_j N_j(\sigma^i(\beta)) \\ &= \sum_{j=1}^{\nu} e_j N_{k_j}(\sigma^i(\beta)) \\ &= \sum_{j=1}^{\nu} e_j \sigma^i(N_{k_j}(\beta)) \\ &= \sum_{j=1}^{\nu} e_j \sigma^i(\alpha^{-1}) \sigma^{k_j+i}(\alpha) \\ &= \sigma^i(\alpha^{-1}) \sum_{j=1}^{\nu} e_j \sigma^{k_j+i}(\alpha). \end{aligned}$$

Por esto, $\sigma^i(\alpha) S_i = \sum_{j=1}^{\nu} e_j \sigma^{k_j+i}(\alpha)$ y llamamos al polinomio $S = \sum_{i=0}^{2\tau-1} \sigma^i(\alpha) S_i x^i$ el *polinomio síndrome* de y.

Teorema 32. *Los polinomios localizador de errores y evaluador de errores cumplen la ecuación clave no conmutativa:*

$$\omega = S\lambda + x^{2\tau} u.$$

donde $u \in R$ es de grado menor que ν.

Demostración. Por definición sabemos que

$$S = \sum_{i=0}^{2\tau-1} \sum_{j=1}^{\nu} e_j \sigma^{k_j+i}(\alpha) x^i,$$

y que para cualquier $1 \leq j \leq \nu$

$$\lambda = (1 - \sigma^{k_j}(\beta)x) p_j.$$

Queremos llegar a que

$$\omega = \sum_{j=1}^{\nu} e_j \sigma^{k_j}(\alpha) p_j = S\lambda + x^{2\tau u}$$

para algún u de grado menor que ν. Tomamos ahora $u = \sum_{i=0}^{2\tau-1} \sum_{j=1}^{\nu} \sigma^{-2\tau}(e_j) \sigma^{k_j}(\alpha) p_j$. Entonces

$$\begin{aligned} S\lambda + x^{2\tau} u &= \sum_{i=0}^{2\tau-1} \sum_{j=1}^{\nu} e_j \sigma^{k_j+i}(\alpha) x^i (1 - \sigma^{k_j}(\beta)x) p_j + x^{2\tau} \sigma^{-2\tau}(e_j) \sigma^{k_j}(\alpha) p_j \\ &= \sum_{j=1}^{\nu} e_j \left(\sum_{i=0}^{2\tau-1} \sigma^{k_j+i}(\alpha) x^i (1 - \sigma^{k_j}(\beta)x) + x^{2\tau} \sigma^{k_j}(\alpha) \right) p_j \end{aligned},$$

tras intercambiar las sumatorias, y sacar factor común e_j y p_j. Ahora nos centramos en la sumatoria interior.

$$\begin{aligned}
&\sum_{i=0}^{2\tau-1} \sigma^{k_j+i}(\alpha)x^i(1-\sigma^{k_j}(\beta)x) + x^{2\tau}\sigma^{k_j}(\alpha) \\
&= \sum_{i=0}^{2\tau-1} x^i\sigma^{k_j}(\alpha)(1-\sigma^{k_j}(\beta)x) + x^{2\tau}\sigma^{k_j}(\alpha) \\
&= \sum_{i=0}^{2\tau-1} x^i\sigma^{k_j}(\alpha) - x^i\sigma^{k_j}(\alpha)\sigma^{k_j}(\alpha^{-1})\sigma^{k_j+1}(\alpha)x) + x^{2\tau}\sigma^{k_j}(\alpha). \\
&= \sum_{i=0}^{2\tau-1} x^i\sigma^{k_j}(\alpha) - x^{i+1}\sigma^{k_j}(\alpha) + x^{2\tau}\sigma^{k_j}(\alpha) \\
&= \left(\sum_{i=0}^{2\tau-1} x^i - x^{i+1} + x^{2\tau}\right)\sigma^{k_j}(\alpha) = \sigma^{k_j}(\alpha)
\end{aligned}$$

Con esto, sustituyendo en la expresión anterior tenemos que

$$\begin{aligned}
S\lambda + x^{2\tau}u &= \sum_{j=1}^{\nu} e_j \left(\sum_{i=0}^{2\tau-1} \sigma^{k_j+i}(\alpha)x^i(1-\sigma^{k_j}(\beta)x) + x^{2\tau}\sigma^{k_j}(\alpha)\right) p_j \\
&= \sum_{j=1}^{\nu} e_j\sigma^{k_j}(\alpha)p_j = \omega
\end{aligned}.$$

□

Ahora intentaremos resolver esta ecuación, para lo que utilizaremos el Algoritmo Euclídeo Extendido a la derecha presentado en 3. Recordemos que, para cualesquiera $f, g \in R$, cada paso i del algoritmo proporciona coeficientes $\{u_i, v_i, r_i\}$ tales que $fu_i + fv_i = r_i$, donde $(f,g)_l = r_h$ y $\deg r_{i+1} < \deg r_i$ para cualquier $0 \leq i \leq h-1$.

Teorema 33. *La ecuación clave no conmutativa*

$$x^{2\tau}u + S\lambda = \omega \tag{3.2}$$

es un múltiplo a la derecha de la ecuación

$$x^{2\tau}u_I + Sv_I = r_I, \tag{3.3}$$

donde u_I, v_I y r_I son los coeficientes de Bezout dados por el Algoritmo Euclídeo Extendido a la derecha con entrada $x^{2\tau}$ y S, e I es el índice determinado por las condiciones $\deg r_{I-1} \geq \tau$ y $\deg r_I < \tau$. En particular, $\lambda = v_I g$ y $\omega = r_I g$ para algún $g \in R$.

Demostración. Recordemos que $\deg S < 2\tau$, $\deg \lambda \leq \nu \leq \tau$ y que $\deg \omega < \nu \leq \tau$. Entonces, $\deg u < \tau$, pues en otro caso $\deg x^{2\tau}u \geq 3\tau > \deg S\lambda$, y por tanto $\deg \omega \geq$

3λ que es contradicción. Por otro lado, por el lema 19 *VI)*, $\deg v_I + \deg r_{I-1} = 2\tau$, y utilizando la hipótesis $\deg r_{I-1} \geq \tau$ tenemos que $\deg v_I \leq \tau$.

Consideremos ahora el mínimo común múltiplo a la derecha $[\lambda, v_I]_r = \lambda a = v_I b$, donde $a, b \in R$ con $\deg a \leq \deg v_I \leq \tau$ y $\deg b \leq \deg \lambda \leq \tau$. Entonces $(a,b)_r = 1$. Multiplicando (3.2) a la derecha por a, y (3.3) a la derecha por b, obtenemos

$$x^{2\tau} ua + S\lambda a = \omega a$$

y

$$x^{2\tau} u_I b + S v_I b = r_I b.$$

Ahora, restando ambas ecuaciones obtenemos $x^{2\tau}(ua - u_I b) = \omega a - r_I b$. Por $\deg \omega < \tau, \deg a \leq \tau, \deg r_I < \tau, \deg b \leq \tau$, tenemos que $\deg(\omega a - r_I b) < 2\tau$, luego $ua = u_I b$ y $\omega a = r_I b$, pues en caso contrario el grado del término izquierdo sería mayor o igual que 2τ. De hecho $(a,b)_r = 1$ nos lleva a que $[u, u_I]_r = ua = u_I b$ y $[\omega, r_I]_r = \omega a = r_I b$, y en particular $\deg a \leq \deg r_I < \tau$.

Sea $[a,b]_l = a'a = b'b$. Por ser $[\lambda, v_I]_r$ un múltiplo a la izquierda de a y b, existe $m \in R$ tal que $[\lambda, v_I]_r = m[a,b]_l$, es decir, $\lambda a = v_I b = ma'a = mb'b$. Por esto, $\lambda = ma'$ y $v_I = mb'$ y, por minimalidad, $(\lambda, v_I)_l = m$. Podemos utilizar argumentos similares para probar que existen $m', m'' \in R$ tales que $u_I = m'b'$ y $u = m'a'$, y $r_I = m''b'$ y $\omega = m''a'$. Por el lema 19 *V)* $(u_I, v_I)_r = 1$, luego $b' = 1$. Esto completa la prueba, pues tenemos $b = bb' = aa'$, y por tanto $\lambda = v_I a'$, $\omega = r_I a'$ y $u = u_I a'$. □

Usando la notación del teorema recién demostrado, vemos que, por $\lambda = v_I a'$ y $\omega = r_I a'$, si $(\lambda, \omega)_r = 1$ entonces $\lambda = v_I$ y $\omega = r_I$. En este caso, el teorema 33 proporciona un método algorítmico para calcular ambos polinomios, el localizador de errores y el evaluador de errores. Como ya dijimos, estos dos polinomios nos permiten calcular el error al recibir un polinomio $y = c + e$, con c y e cumpliendo las hipótesis previamente mencionadas, por tanto, podemos describir un algoritmo de decodificación para códigos skew RS (vease el algoritmo 5) que será válido siempre que $(\lambda, \omega)_r = 1$.

Ejemplo 34. En este ejemplo mostraremos la ejecución de este algoritmo utilizando la implementación que ha sido realizada en SageMath.

Sea $\mathbb{F} = \mathbb{F}_2(t)$ el cuerpo finito con 2^{12} elementos, donde $x^{12} + x^7 + x^6 + x^5 + x^3 + x + 1 = 0$. Escribiremos los elementos de $\mathbb{F}$ como potencias de t excepto el 0 y el 1. Consideremos entonces $\sigma = \tau^{10}$ donde τ es el endomorfismo de Frobenius, por ejemplo $sigma(t) = t^{1024}$. El orden de σ es 6, luego un código cíclico sesgado sobre $\mathbb{F}$ será un ideal a la izquierda del álgebra cociente $\mathcal{R} = \mathbb{F}[x;\sigma]/\langle x^6 - 1\rangle$.

Tomemos ahora $\alpha = t$, que proporciona una base normal de $\mathbb{F}$ como $\mathbb{F}^\sigma$-espacio vectorial, y $\beta = \sigma(t)t^{-1} = t^{1023}$. Consideramos entonces el código skew RS generado por

$$g = [x - \beta, x - \sigma(\beta), x - \sigma^2(\beta), x - \sigma^3(\beta)].$$

Algoritmo 5: Algoritmo de decodificación para códigos skew RS

Entrada: Un polinomio $y = \sum_{i=0}^{n-1} y_i x^i$ obtenido de la transmisión de una palabra código c perteneciente a un código skew RS C generado por $g = [\{x - \sigma^i(\beta)\}_{i=0,\dots,\delta-2}]_l$ con capacidad para corregir $\tau = \lfloor (\delta - 1)/2 \rfloor$ errores.

Salida: Una palabra código c' o un *error de ecuación clave*.

for $0 \leq i \leq 2\tau - 1$ **do**
  $S_i \leftarrow \sum_{j=0}^{n-1} y_j N_j(\sigma^i(\beta))$
$S \leftarrow \sum_{i=0}^{2\tau-1} \sigma^i(\alpha) S_i x^i$
if $S = 0$ **then**
  return y
$\{u_i, v_i, r_i\}_{i=0,\dots,I} \leftarrow \text{REEA}(x^{2\tau}, S)$
$I \leftarrow$ primera iteración en REEA tal que $\deg r_i < \tau$
$pos \leftarrow \emptyset$
for $0 \leq i \leq n - 1$ **do**
  if $\sigma^{i-1}(\beta^{-1})$ *es una raíz a la izquierda de* v_I **then**
    $pos = pos \cup \{i\}$
if $\deg v_I > \text{Cardinal}(pos)$ **then**
  return *error en la ecuación clave*
for $j \in pos$ **do**
  $p_j \leftarrow \text{rquo}(v_I, 1 - \sigma^j(\beta)x)$
Resolver el sistema lineal $r_I = \sum_{j \in pos} e_j \sigma^j(\alpha) p_j$
$e \leftarrow \sum_{j \in pos} e_j x^j$
return $y - e$

Por el teorema 29 + tiene distancia Hamming 5 + y por tanto puede corregir hasta 2 errores. Supongamos que queremos mandar el mensaje $m = tx + 1$ + así que el polinomio codificado que transmitiremos será $c = mg = tx^5 + t^{2715}x^4 + t^{1452}x^3 + t^{2080}x^2 + t^{3786}x + t^{759}$. Después de la transmisión, recibimos el polinomio $y = tx^5 + t^{2715}x^4 + t^{1452}x^3 + t^{333}x^2 + t^{3786}x + t$ por ejemplo, donde hay errores en las posiciones 0 y 2.

```
sage: m = t*x + 1
sage: c = m*g
sage: y = copy(c)
sage: y[0] = t
sage: y[2] = t**333
```

En primer lugar calculamos el polinomio síndrome S.

```
sage: for i in range(2*tau):
sage:   S_i = sum([R(y[j]*norm(j, sigma, (sigma**i)(beta)))
          for j in range(n)])
sage:   S = S + (sigma**i)(alpha) * S_i * x**i
sage: S
(t^10 + t^9 + t^8 + t^7 + t^5 + 1)*x^3 + ...
     + t^8 + t^5 + t^4 + t^3 + 1
```

Como S es distinto de 0, el polinomio y no pertenece al código y han ocurrido errores en la transmisión. Aplicamos el algoritmo extendido de Euclides a la derecha sobre $x^{2\tau}$ y S, y buscamos la primera posición en la que se cumpla que $\deg r_i < \tau$.

```
sage: u, v, r = right_extended_euclidean_algorithm(R,
        R(x**(2*tau)), S)
sage: I = 0
sage: for (i, r_i) in enumerate(r):
sage:    if r_i.degree() < tau:
sage:        I = i
sage:        break
sage: I
3
```

A continuación calculamos las posiciones del error, y comprobamos que no se dé un error en la ecuación clave.

```
sage: pos = []

sage: for i in range(n):
sage:     if (R(x - (sigma**(i-1))(beta**(-1))))
            .left_divides(v[I]):
sage:           pos.append(i)

sage: v[I].degree() > len(pos):
False
```

En efecto, no se da un error, luego sabemos que los polinomios v_I y r_I (en el código `v[I]` y `r[I]`) son respectivamente los polinomios localizador (λ) y evaluador de errores (ω). El siguiente paso entonces es calcular los polinomios p_j de forma que $\lambda = (1 - \sigma^j(\beta)x)p_j$ para cada $j \in pos$.

```
sage: p = {}
sage: for j in pos:
sage:     p[j] = (v[I].left_quo_rem(
            R(1 - (sigma**j)(beta)*x)))[0]
```

Ya tenemos todos los ingredientes necesarios para construir el sistema lineal y resolverlo, lo que nos proporcionará el error que ha ocurrido durante la transmisión.

```
sage: omega = _to_complete_list(r[I], tau)
sage: Sigma = matrix([[(sigma**j)(alpha) * p[j][i]
        for j in pos] for i in range(tau)])kj,,kjj
sage: E = Sigma.transpose().solve_left(vector(omega))
sage: e = sum([E[j] * x**(pos[j]) for j in range(len(pos))
    ])
```

Por último solo tenemos que restarle el error a la palabra recibida, y comprobamos que coincide con la palabra código transmitida concluyendo el ejemplo.

```
sage: y - e == c
True
```

En caso de que se dé $(\lambda, \omega)_r \neq 1$, el algoritmo no es válido pues v_I y r_I no serían los polinomios localizador y evaluador de errores respectivamente. Por el lema 21 es directo ver que si se cumple la condición que presentamos para devolver un error entonces $(\lambda, \omega)_r = 1$. Para el caso en que se de la igualdad $\deg v_I = \text{Cardinal}(pos)$, aunque a priori no podemos descartar que ocurra un error, en la siguiente sección vemos un resultado (teorema 37) que nos dice que si se da la igualdad entonces no puede ocurrir un error. Por tanto, solo puede ocurrir un error en la ecuación clave si se cumple dicha condición.

3.4. Errores en la ecuación clave

En esta sección abordaremos el problema producido por un error en la ecuación clave, es decir, cuando el máximo común divisor a la derecha de los polinomios localizador y evaluador de errores no coincide con la unidad. Además concluiremos la sección analizando cómo de probable es que ocurra un error de este tipo, y estudiando la complejidad del algoritmo

Nuestro principal objetivo entonces es encontrar un método que nos permita, aun existiendo el error mencionado, encontrar los polinomios localizador de errores y evaluador de errores.

A lo largo de toda esta sección seguiremos la notación utilizada en la sección anterior (3.3).

El primer resultado que mostramos será que, si se produce un único error en el mensaje, en cuyo caso el polinomio localizador será de grado uno, este siempre puede ser corregido.

Lema 35. $\deg v_I \geq 1$. *Como consecuencia, si* $\deg \lambda = 1$, *entonces* v_I *y* λ *son asociados a la derecha.*

Demostración. La prueba se basa en los grados de los polinomios del teorema 33. Por este teorema, $\omega = r_I g$ para algún $g \in R$. Como ya mencionamos anteriormente, $\deg \omega < \nu$, y por tanto $\deg g < \nu$. Ahora, el mismo teorema nos dice que $\lambda = v_I g$, y por el teorema 26 tenemos que $\nu = \deg \lambda = \deg v_I + \deg g$. De aquí sigue que $\deg v_I \geq 1$. □

Volvamos ahora a considerar el caso en el que se produce un error en la ecuación clave, y como afrontarlo. Por el teorema 33 y siguiendo el algoritmo 5 obtenemos los polinomios r_I, u_i, v_I, que cumplen la ecuación $x^{2\tau} u_I + S v_I = r_I$, con $\lambda = v_I g$ y $\omega = r_I g$ para algún $g \in R$.

Ahora bien, por el lema recién demostrado, sabemos que $\deg v_I \geq 1$. Si $\deg g = 0$, entonces v_I nos sirve como polinomio localizador, y el algoritmo 5 decodificará correctamente el polinomio recibido. En caso contrario, nuestra estrategia consistirá en, empezando por v_I, encontrar una cadena creciente de divisores a la izquierda de λ, lo que nos permitirá encontrar una posición nueva del error por cada elemento de dicha cadena. En primer lugar, queremos demostrar un criterio que nos diga si hemos encontrado ya todas las posiciones del error, y de ser así tendremos el polinomio localizador. Para demostrar esto necesitaremos primero probar el siguiente lema.

Lema 36. *Sean* $\{t_1 < t_2 < \cdots < t_m\} \subset \{0, 1, \ldots, n-1\}$ *con* $m > 1$, *y*

$$q = [1 - \sigma^{t_1}(\beta)x, 1 - \sigma^{t_2}(\beta)x, \ldots, 1 - \sigma^{t_m}(\beta)x]_r.$$

Sean $q_1, q_2, \ldots, q_m$ *tales que* $q = (1 - \sigma^{t_j}(\beta)x)q_j$ *para cualquier* $1 \leq j \leq m$. *Entonces:*

1. $[q_1, q_2, \ldots, q_m]_l = q$ *y* $(q_1, q_2, \ldots, q_m)_r = 1$.
2. $R/Rq = \bigoplus_{j=1}^{m} Rq_j/Rq$.
3. *Para cada* $f \in R$ *con* $\deg f < m$ *existen* $a_1, a_2, \ldots, a_m \in \mathbb{F}$ *tales que* $f = \sum_{j=1}^{m} a_j q_j$.
4. *El conjunto* $\{q_1, \ldots, q_m\}$ *módulo* Rq *proporciona una base de* R/Rq *como un* $\mathbb{F}$*-espacio vectorial.*

Demostración. 1. Utilizando el lema 30 para seguidamente poder aplicar el lema 26 tenemos que $\deg q = m$, y por tanto

$$\deg q_j = m - 1 \quad \forall j \in \{1, \ldots, m\}.$$

Como $m > 1$, el grado del mínimo común múltiplo a la derecha de cualesquiera dos elementos en $\{q_1, q_2, \ldots, q_m\}$ debe ser al menos $m - 1 + 1 = m$, luego el grado de $[q_1, \ldots, q_m]_l$ debe ser al menos m. Pero q es claramente un mínimo común múltiplo a la izquierda de $q_1, \ldots, q_m$. Así, $q = [q_1, \ldots, q_m]_l$.

Veamos ahora que $p = (q_1, q_2, \ldots, q_m)_r = 1$. Supongamos $p \neq 1$, y $q_j = q'_j p$ para todo $j = 1, \ldots, m$. Entonces

$$q = (1 - \sigma^{t_j}(\beta)x)q'_j p \quad \forall j \in \{1, \ldots, m\}.$$

Esto es una contradicción pues, si definimos

$$q' = (1 - \sigma^{t_1}(\beta)x)q'_1 = \cdots = (1 - \sigma^{t_m}(\beta)x)q'_m,$$

entonces q' es un múltiplo común a la derecha de cada $1 - \sigma^{t_j}(\beta)x$ de grado menor que q así que este no podría ser el mínimo común múltiplo a la derecha de dichos términos.

2. Cada uno de los polinomios q_j es un divisor a la derecha de q y por tanto $Rq \subset Rq_j$ para $j = 1, \ldots, m$. Por esto, como del apartado anterior sabemos que $(q_1, \ldots, q_m)_r = 1$, tenemos que

$$R/Rq = \sum_{j=1}^{m} Rq_j/Rq.$$

Además, por $q = (1 - \sigma^{t_j}(\beta)x)q_j$ para $1 \leq j \leq m$, $Rq_j/Rq \equiv R/R(1 - \sigma^{t_j}(\beta)x)$ es uno dimensional sobre F. Así, como la dimensión de R/Rq como $\mathbb{F}$-espacio vectorial es $\deg q = m$, tenemos la suma directa que buscábamos.

3. Se obtiene directamente del apartado 2.

4. Se obtiene directamente del apartado 2.

□

Con este lema ya estamos en posición de enunciar y probar el teorema para saber si hemos encontrado ya todas las posiciones de error.

Teorema 37. *Sean $q, p, s \in R$ tales que $x^{2\tau}q + Sp = s$, $qg = u$, $pg = \lambda$, y $sg = \omega$ para algún $g \in R$. Consideremos $T = \{t_1, t_2, \ldots, t_m\} \subset \{0, 1, \ldots, n-1\}$ el conjunto de índices que verifican que $\sigma^{j-1}(\beta^{-1})$ es una raíz a la izquierda de p si y solo si $j \in T$. Entonces $m = \deg p$ si y solo g es una constante.*

Demostración. En primer lugar mencionemos que los elementos de T son en realidad posiciones del error. Esto se cumple ya que dividen por la izquiera a p, y por $\lambda = pg$, también a λ, así que podemos aplicar la proposición 31. Dicho esto, reordenamos el conjunto de las posiciones del error de forma que $T = \{k_1, k_2, \ldots, k_m\}$.

Si $\deg g = 0$, entonces las raíces de p coinciden con las de λ, y por tanto, por el lema 26, $m = \deg p = \deg \lambda = \nu$. Supongamos ahora que $m = \deg p$. Por el lema 26 sabemos que $[x - \sigma^{k_1}(\beta^{-1}), \ldots, x - \sigma^{k_m}(\beta^{-1})]_r$ tiene grado m, y por tanto tiene el mismo grado que p. Por esto $p = [x - \sigma^{k_1-1}(\beta^{-1}), x - \sigma^{k_2-1}(\beta^{-1}), \ldots, x - \sigma^{k_m-1}(\beta^{-1})]_r$, y aplicando directamente el lema 30 tenemos que:

$$p = [1 - \sigma^{k_1}(\beta)x, \ldots, 1 - \sigma^{k_m}(\beta)x]_r.$$

Por el apartado 3 del lema 36 , cada polinomio de grado menor que m puede escribirse como una combinación $\mathbb{F}$-lineal de los polinomios $p'_1, \ldots, p'_m$. Ahora, por $sg = \omega$ y $pg = \lambda$,

$$\deg s = \deg \omega - \deg g = \deg \omega + \deg p - \deg \lambda \leq \nu - 1 + m - \nu = m - 1,$$

luego podemos escribir s de la forma $\sum_{i=1}^{m} a_i p'_i$ para ciertos $a_1, a_2, \ldots, a_m \in F$. Recordemos que, siguiendo la notación de la sección anterior, los polinomios $p_1, \ldots, p_\nu$ eran aquellos tales que $\lambda = (1 - \sigma^{k_j}(\beta)x)p_j$ donde k_j era una posición del error cualquiera. Entonces, por $\lambda = pg$, para cada $j = 1, \ldots, m$ se cumple que $(1 - \sigma^{k_j}(\beta)x)p_j = (1 - \sigma^{k_j}(\beta)x)p'_j g$, luego $p_j = p'_j g$. Ahora, usando que $sg = \omega$, tenemos

$$\begin{aligned} \omega &= \sum_{j=1}^{m} e_j \sigma^{k_j}(\alpha) p_j + \sum_{j=m+1}^{\nu} e_j \sigma^{k_j}(\alpha) p_j = sg = \left(\sum_{j=1}^{m} a_j p'_j \right) g \\ &= \sum_{j=1}^{m} a_j p_j \end{aligned}.$$

Ahora, volviendo a aplicar el lema 36 (IV), pero esta vez sobre todas las posiciones de error y el polinomio localizador, obtenemos que $\{p_1, \ldots, p_\nu\}$ es una base de $R/R\lambda$ como $\mathbb{F}$-espacio vectorial. Así, como $e_i \sigma^{k_i} \neq 0$ para cada $i \leq \nu$, la ecuación 3.4 nos dice que $m = \nu$, y por tanto, $\deg g = 0$. □

Ahora que gracias a este teorema sabemos cuando hemos encontrado todas las posiciones del error, el siguiente paso será construir el algoritmo que nos permita, en caso de existir, encontrar una posición nueva. De esta forma, en caso de que se produzca un error en la ecuación clave en el algoritmo 5, podremos hallar nuevas posiciones del error iterativamente hasta que sepamos que no existen más.

Algoritmo 6: Encuentra-una-posición

Entrada: Un polinomio no constante p tal que $\lambda = pg$ para algún $g \in R$,
$pos = \{i \geq 0$ tales que $(1 - \sigma^i(\beta)x) \mid_l p\}$, con $\deg p >$ Cardinal(pos).
Salida: $d \notin pos$ tal que $(1 - \sigma^d(\beta)x)$ divide a la izquierda a λ.

```
f ← p, e ← deg f
for 0 ≤ i ≤ n − 1 do
    if i ∉ pos then
        f ← [f, 1 − σ^i(β)x]_r
        if deg f = e then
            return i
        else
            e ← e + 1
```

Proposición 38. *El algoritmo 6 encuentra una nueva posición del error correctamente.*

Demostración. Sea $T = \{t_1, \ldots, t_r\} = \{0, 1, ..., n-1\} \setminus pos$. Definimos ahora

$$\begin{array}{ll} \lambda_0 = \lambda, & \lambda_i = [\lambda_{i-1}, 1 - \sigma^{t_i}(\beta)x]_r \quad \text{para } 1 \leq i \leq r \\ f_0 = p, & f_i = [f_{i-1}, 1 - \sigma^{t_i}(\beta)x]_r \quad \text{para } 1 \leq i \leq r \end{array}.$$

Veamos que $f_i \mid_l \lambda_i$ para cualquier $i = 0, \ldots, r$. En efecto, para $i = 0$ el resultado es cierto por hipótesis, y supuesto que se cumple para $i-1$, f_{i-1} divide a la izquierda a λ_i por dividir a λ_{i-1}, y $1 - \sigma^{t_i}(\beta)x$ lo divide también de forma trivial, luego por definición el mínimo común múltiplo a la derecha de estos divide a λ_i. Dicho esto, probaremos en primer lugar que el algoritmo devuelve una posición del error.

Supongamos que la secuencia $\{\deg f_i\}_{0 \leq i \leq r}$ es estrictamente creciente. Por esto, $\deg f_r = r + \deg p > n - \text{Cardinal}(pos) + \deg p > n$. Esto sin embargo no es posible, pues $f_r \mid_l \lambda_r = x^n - 1$. Por tanto, existe un $d \geq 0$ mínimo tal que $\deg f_{d-1} = \deg f_d$. Entonces, $1 - \sigma^{t_d}(\beta)x \mid_l f_{d-1} \mid_l \lambda_{d-1} = [\lambda, 1 - \sigma^{t_1}(\beta)x, \ldots, 1 - \sigma^{t_{d-1}}(\beta)x]_r$. Como $t_d \neq t_1, \ldots, t_{d-1}$, $1 - \sigma^{t_d}(\beta)x$ divide a la izquierda λ concluyendo la prueba. □

Así, utilizando recursivamente el algoritmo 6 encontraremos eventualmente todas las posiciones del error, y así conseguiremos también los polinomios localizador de errores y evaluador de errores. Estos pasos los mostramos formalmente en el algoritmo 7.

Algoritmo 7: Algoritmo para resolver el error en la ecuación clave

Entrada: Polinomios v_i, r_I tales que $\lambda = v_I g, \omega = r_I g$ para algún $g \in R$, el conjunto $pos = \{i \geq 0 \mid (1 - \sigma^i(\beta)x) \mid_l v_I\}$

Salida: El polinomio localizador de errores λ y el polinomio evaluador de errores ω.

```
f ← v_I
while Cardinal(pos) < deg f do
    d ← Find-a-position(f, pos)
    f ← [f, 1 − σ^d(β)x]_r
    for 0 ≤ i ≤ n − 1 do
        if i ≠ pos y 1 − σ^i(β)x |_l f then
            pos ← pos ∪ {i}
g ← rquo(f, v_I)
return f, r_I g
```

Por tanto, esto nos permite que, en caso de que se de un error de ecuación clave en el algoritmo 5, calcular los polinomios localizador y evaluador de errores. De esta manera podemos concluir la decodificación resolviendo el sistema lineal con el que finaliza dicho algoritmo

Para concluir esta sección analizaremos con qué frecuencia ocurre un error en la ecuación clave. De hecho, dado un conjunto de posiciones de error, mostraremos que los valores de los errores deben satisfacer una relación no trivial. Recordemos que dicho error en la ecuación clave puede ocurrir únicamente si $(\lambda, \omega)_r \neq 1$.

Proposición 39. *Los siguientes enunciados son equivalentes:*

1. $(\lambda, \omega)_r = 1$.
2. $\omega + R\lambda$ *genera* $R/R\lambda$ *como un* R*-módulo a la izquierda.*
3. *El conjunto* $\{x^i(\omega + R\lambda) \mid 0 \leq i \leq \nu - 1\}$ *es linealmente independiente sobre* $\mathbb{F}$.

Demostración. La equivalencia entre 1 y 2 es una consequencia directa de la identidad de Bezout, ya que $(\lambda, \omega)_r = 1$ si y solo si $R\omega + R\lambda = R$. Para terminar la prueba de la proposición veremos la equivalencia entre 2 y 3. Es claro que $\omega + R\lambda$ genera el R-módulo a la izquierda $R/R\lambda$ si y solo si $\{x^i(\omega + R\lambda) \mid 0 \leq i \leq \nu\}$ genera $R/R\lambda$ como un $\mathbb{F}$-espacio vectorial. Como la dimensión como $\mathbb{F}$-espacio vectorial de $R/R\lambda$ es ν, la equivalencia entre 2 y 3 es clara. □

Lema 40. *La coordenada j-ésima de* $x^i\omega + R\lambda$ *respecto a* $\{p_1, \ldots, p_\nu\}$ *es* $\sigma^i(e_j)\sigma^{k_j}(\alpha)$*, para cada* $1 \leq j \leq \nu$.

Demostración. Notemos primero que $R(1 - \sigma^{t_j}(\beta)x) = R(x - \sigma^{t_j}(\beta^{-1}))$ para $j = 1, \ldots, m$ por el razonamiento de la demostración del lema 30. Por el lema 21, $\sigma^{t_j}(\beta^{-1})$ es una raíz a la derecha de $x^i - N_i(\sigma^{t_j}(\beta^{-1}))$. Entonces, $x^i - N_i(\sigma^{t_j}(\beta^{-1})) \in R(1 - \sigma^{t_j}(\beta)x)$. Multiplicando a la derecha por p_j, $x_i p_j - N_i(\sigma^{t_j}(\beta^{-1}))p_j \in R\lambda$. Por esto, en $R/R\lambda$,

$$\begin{aligned} x^i\omega &= \sum_{j=1}^{\nu} x^i e_j \sigma^{k_j}(\alpha) p_j \\ &= \sum_{j=1}^{\nu} \sigma^i(e_j)\sigma^{k_j+i}(\alpha) x^i p_j \\ &= \sum_{j=1}^{\nu} \sigma^i(e_j)\sigma^{k_j+i}(\alpha) N_i(\sigma^{k_j}(\beta^{-1}))p_j. \end{aligned}$$

Ahora, la coordenada asociada a p_j viene dada por

$$\begin{aligned} &\sigma^i(e_j)\sigma^{k_j+i}(\alpha) N_i(\sigma^{k_j}(\beta^{-1})) \\ &\quad = \sigma^i(e_j)\sigma^{k_j+i}(\alpha)\sigma^{k_j}(N_i(\beta^{-1})) \\ &\quad = \sigma^i(e_j)\sigma^{k_j+i}(\alpha)\sigma^{k_j}(\alpha\sigma^i(\alpha^{-1}))' \\ &\quad = \sigma^i(e_j)\sigma^{k_j}(\alpha) \end{aligned}$$

concluyendo la prueba. □

Proposición 41. $(\lambda, \omega)_r = 1$ *si y solo si*

$$\begin{vmatrix} e_1 & e_2 & \cdots & e_\nu \\ \sigma(e_1) & \sigma(e_2) & \cdots & \sigma(e_\nu) \\ \vdots & \vdots & \ddots & \vdots \\ \sigma^{\nu-1}(e_1) & \sigma^{\nu-1}(e_2) & \cdots & \sigma^{\nu-1}(e_\nu) \end{vmatrix} \neq 0$$

Demostración. Utilizando la proposición y el lema recién demostrados, $(\lambda, \omega)_r = 1$ si y solo si la matriz A dada por

$$A = \begin{pmatrix} e_1\sigma^{k_1}(\alpha) & e_2\sigma^{k_2}(\alpha) & \cdots & e_\nu\sigma^{k_\nu}(\alpha) \\ \sigma(e_1)\sigma^{k_1}(\alpha) & \sigma(e_2)\sigma^{k_2}(\alpha) & \cdots & \sigma(e_\nu)\sigma^{k_\nu}(\alpha) \\ \vdots & \vdots & \ddots & \vdots \\ \sigma^{\nu-1}(e_1)\sigma^{k_1}(\alpha) & \sigma^{\nu-1}(e_2)\sigma^{k_2}(\alpha) & \cdots & \sigma^{\nu-1}(e_\nu)\sigma^{k_\nu}(\alpha) \end{pmatrix}$$

tiene rango completo, es decir, su determinante es distinto de o. Como $\sigma^{k_j}(\alpha) \neq 0$ para cada $j = 1, \ldots, \nu$, tenemos que dicho determinante es distinto de o si y solo si el determinante en 41 es no nulo. □

Teorema 42. $(\lambda, \omega)_r \neq 1$ *si y solo si los valores del error* $e_1, \ldots, e_\nu$ *son linealmente dependientes sobre* $\mathbb{F}^\sigma$.

Demostración. Consideremos la matriz

$$A = \begin{pmatrix} e_1 & e_2 & \cdots & e_\nu \\ \sigma(e_1) & \sigma(e_2) & \cdots & \sigma(e_\nu) \\ \vdots & \vdots & \ddots & \vdots \\ \sigma^{\nu-1}(e_1) & \sigma^{\nu-1}(e_2) & \cdots & \sigma^{\nu-1}(e_\nu) \end{pmatrix},$$

de forma que por la proposición 41 nos dice que $(\lambda, r)_r \neq 1$ si y solo si esta no es de rango completo. Ahora, siguiendo la notación de [LL88], A es una $\nu \times \nu$ matriz Wronskiana con respecto al endomorfismo σ; simbolicamente $A = W^\sigma_{\nu,\nu}(e_1, \ldots, e_\nu)$. Por [LL88, Corolario 4.13], A es invertible si y solo si $e_1, \ldots, e_\nu$ son linealmente independientes sobre $\mathbb{F}^\sigma$, y por tanto A no es de rango completo si y solo si $e_1, \ldots, e_\nu$ son linealmente dependientes sobre $\mathbb{F}^\sigma$, concluyendo la prueba. □

Por tanto, como dijimos, hemos probado que para que se dé un error en la ecuación clave ($(\lambda, \omega)_r \neq 1$) tiene que existir una dependencia lineal entre las posiciones del error.

4 Implementación en SageMath del Algoritmo de Sugiyama para códigos cíclicos sesgados

El desarrollo del algoritmo de Sugiyama para códigos RS sesgados, juntos con la estructura necesaria para representar dichos códigos se ha realizado en el entorno SageMath ([S+20]). Este es una implementación de código abierto de software matemático y científico basado en Python.

Ya existen en SageMath clases esqueleto para representar códigos lineales, así como para implementar métodos de codificación y decodificación para estos. Por ello, para nuestra implementación hemos seguido las directrices de presentes en

`https://doc.sagemath.org/html/en/thematic_tutorials/structures_in_coding_theory.html`.

Además de la estructura base para códigos, Sage también contiene una implementación de anillos de polinomios sesgados, la cual se usa a lo largo de todo el desarrollo del algoritmo y en las clases esqueleto para los nuevos códigos.

La primera clase que se ha implementado es `SkewCyclicCode` para representar cualquier código cíclico sesgado, la cual hereda de la clase abstracta `AbstractLinearCode`. Para poder codificar mensajes en palabras código de este tipo de códigos se han diseñado las clases `SkewCyclicCodeVectorEncoder` y `SkewCyclicCodePolynomialEncoder`, ambas heredando de la clase de Sage `Encoder`. A continuación, heredando de la clase `SkewCyclicCode` se implementó la clase `SkewRSCode` que almacena la estructura necesaria para trabajar con códigos Reed-Solomon sesgados. Por último, la clase `SkewRSCodeSugiyamaDecoder`, que hereda de la clase `Decoder`, contiene la implementación del algoritmo de decodificación desarrollado en este trabajo.

El código fuente se encuentra en el archivo `skew_cyclic_code.sage` en

`https://github.com/guillegalor/tfg/tree/master/code`.

En la misma carpeta se encuentra otro archivo con ejemplos para probar las clases implementadas.

A continuación mostramos la documentación junto con ejemplos de uso de cada una de las estructuras y funciones implementadas.

4.1. Clase de códigos cíclicos sesgados

class SkewCyclicCode(self, generator_pol=None) Clase para representar un código cíclico sesgado. Hereda de `AbstractLinearCode`.

ARGUMENTOS

generator_pol Polinomio generador para construir el código. Debe dividir a $x^n - 1$, donde n es el grado del automorfismo asociado al anillo de polonimios sesgado al que pertenece g.

EJEMPLOS

```
sage: F.<t> = GF(3^10)
sage: sigma = F.frobenius_endomorphism()
sage: R.<x> = F['x', sigma]
sage: g = x**2 + t**24561*x + t**47264
sage: C = SkewCyclicCode(generator_pol=g)
sage: C
[10, 8] Skew Cyclic Code over Finite Field in t of size
    3^10
```

```
sage: F.<t> = GF(3^10)
sage: sigma = F.frobenius_endomorphism()
sage: R.<x> = F['x', sigma]
sage: g = x**3 - 1
sage: C = SkewCyclicCode(generator_pol=g)
Traceback (most recent call last)
...
ValueError: Provided polynomial must divide x^n - 1,
   where n is the order of the ring automorphism.
```

MÉTODOS

generator_polynomial(self) Devuelve el polinomio generador de `self`.

EJEMPLOS

```
sage: F.<t> = GF(3^10)
sage: sigma = F.frobenius_endomorphism()
sage: R.<x> = F['x', sigma]
sage: g = x**2 + t**24561*x + t**47264
sage: C = SkewCyclicCode(generator_pol=g)
sage: C.generator_polynomial()
x^2 + ...  + t^5 + t^2 + 2
```

skew_polynomial_ring(self) Devuelve el anillo de polinomios de sesgados de self.

EJEMPLOS

```
sage: F.<t> = GF(3^10)
sage: sigma = F.frobenius_endomorphism()
sage: R.<x> = F['x', sigma]
sage: g = x**2 + t**24561*x + t**47264
sage: C = SkewCyclicCode(generator_pol=g)
sage: C.skew_polynomial_ring()
Skew Polynomial Ring in x over Finite Field in t of
      size 3^10 twisted by t |--> t^3
```

4.2. Clases de codificadores para códigos cíclicos sesgados

class SkewCyclicCodeVectorEncoder(self, code) Clase para codificar un vector como palabras código. Hereda de Encoder.

ARGUMENTOS

code Código sobre el que codificaremos los vectores.

EJEMPLOS

```
sage: F.<t> = GF(3^10)
sage: sigma = F.frobenius_endomorphism()
sage: R.<x> = F['x', sigma]
sage: g = x**2 + t**24561*x + t**47264
sage: C = SkewCyclicCode(generator_pol=g)
sage: E = SkewCyclicCodeVectorEncoder(C)
sage: E
Vector-style encoder for [10, 8] Skew Cyclic Code over
   Finite Field in t of size 3^10
```

MÉTODOS

generator_matrix(self) Construye una matriz generatriz asociada al código asociado a este codificador

EJEMPLOS

```
sage: F.<t> = GF(3^10)
sage: sigma = F.frobenius_endomorphism()
sage: R.<x> = F['x', sigma]
```

```
sage: g = x**2 + t**24561*x + t**47264
sage: C = SkewCyclicCode(generator_pol=g)
sage: E = SkewCyclicCodeVectorEncoder(C)
sage: E.generator_matrix()
[2*t^9 + t^8 + 2*t^6 + t^5 + t^2 + 2 ... 0]
[                    ...                  ]
[                    ...                  ]
[                    ...                  ]
[                    ...                  ]
[                    ...                  ]
[                    ...                  ]
[0                   ...                 1]
```

class SkewCyclicCodePolynomialEncoder(self, code) Clase para codificar polinomios en palabras código. Hereda de `Encoder`.

ARGUMENTOS

code Código sobre el que codificaremos los vectores.

EJEMPLOS

```
sage: F.<t> = GF(3^10)
sage: sigma = F.frobenius_endomorphism()
sage: R.<x> = F['x', sigma]
sage: g = x**2 + t**24561*x + t**47264
sage: C = SkewCyclicCode(generator_pol=g)
sage: E = SkewCyclicCodePolynomialEncoder(C)
sage: E
Polynomial-style encoder for [10, 8] Skew Cyclic Code
    over Finite Field in t of size 3^10
```

MÉTODOS

encode(self, p) Transforma el polinomio `p` en un elemento del código asociado a `self`.

ARGUMENTOS

p Polinomio que será codificado.

EJEMPLOS

```
sage: F.<t> = GF(3^10)
sage: sigma = F.frobenius_endomorphism()
sage: R.<x> = F['x', sigma]
sage: g = x**2 + t**24561*x + t**47264
```

```
sage: C = SkewCyclicCode(generator_pol=g)
sage: E = SkewCyclicCodePolynomialEncoder(C)
sage: m = x + t + 1
sage: E.encode(m)
(t^8 + 2*t^7 + 2*t^6 + 2*t^4 + t^3 + t^2 + 1, ... ,
    0)
```

unencode_nocheck(self, c) Devuelve el mensaje en forma de polinomio asociado a c sin comprobar si c pertenece al código

ARGUMENTOS

c Un vector de la misma longitud del código asociado a `self`

EJEMPLOS

```
sage: F.<t> = GF(3^10)
sage: sigma = F.frobenius_endomorphism()
sage: R.<x> = F['x', sigma]
sage: g = x**2 + t**24561*x + t**47264
sage: C = SkewCyclicCode(generator_pol=g)
sage: E = SkewCyclicCodePolynomialEncoder(C)
sage: m = x + t + 1
sage: w = E.encode(m)
sage: E.unencode_nocheck(w)
x + t + 1
```

message_space(self) Devuelve el espacio de mensajes asociado a `self`.

EJEMPLOS

```
sage: F.<t> = GF(3^10)
sage: sigma = F.frobenius_endomorphism()
sage: R.<x> = F['x', sigma]
sage: g = x**2 + t**24561*x + t**47264
sage: C = SkewCyclicCode(generator_pol=g)
sage: E = SkewCyclicCodePolynomialEncoder(C)
sage: E.message_space()
Skew Polynomial Ring in x over Finite Field in t of
    size 3^10 twisted by t |--> t^3
```

4.3. Clase de Códigos Reed-Solomon Sesgados

class SkewRSCode(self, hamming_dist=None,skew_polynomial_ring=None, alpha=None) Clase para representar un código RS sesgado. Hereda de `SkewCyclicCode`.

ARGUMENTOS

hamming_dist La distancia hamming de `self`.

skew_polynomial_ring El anillo de polinomios sesgados base.

alpha Un generador normal de la extensión de cuerpos sobre el cuerpo fijo por el automorfismo asociado a `skew_polynomial_ring`.

EJEMPLOS

```
sage: F.<t> = GF(2^12, modulus=x**12 + x**7 +
               x**6 + x**5 + x**3 + x +1)
sage: sigma = (F.frobenius_endomorphism())**10
sage: R.<x> = F['x', sigma]
sage: alpha = t
sage: RS_C = SkewRSCode(hamming_dist=5,
   skew_polynomial_ring=R, alpha=alpha)
sage: RS_C
[6, 2] Skew Reed Solomon Code over Finite
Field in t of size 2^12
```

```
sage: F.<t> = GF(3^10)
sage: sigma = F.frobenius_endomorphism()
sage: R.<x> = F['x', sigma]
sage: RS_C = SkewRSCode(hamming_dist=4,
   skew_polynomial_ring=R, alpha=t)
ValueError: Provided alpha must be an normal generator
   of the field extension given by sigma
```

MÉTODOS

hamming_dist(self) Devuelve la distancia Hamming asociada a este código.

EJEMPLOS

```
sage: F.<t> = GF(2^12, modulus=x**12 + x**7 +
               x**6 + x**5 + x**3 + x +1)
sage: sigma = (F.frobenius_endomorphism())**10
sage: R.<x> = F['x', sigma]
sage: alpha = t
sage: RS_C = SkewRSCode(hamming_dist=5,
   skew_polynomial_ring=R, alpha=alpha)
sage: RS_C.hamming_dist()
2
```

4.4. Clase del decodificador para códigos RS sesgados

class SkewRSCodeSugiyamaDecoder(self, code) Clase para decodificar un vector recibido en una palabra del código asociado a `self`. Hereda de `Decoder`.

ARGUMENTOS

code Código al que pertenecerá la palabra que devuelve el decodificador.

EJEMPLOS

```
sage: F.<t> = GF(2^12, modulus=x**12 + x**7 +
              x**6 + x**5 + x**3 + x +1)
sage: sigma = (F.frobenius_endomorphism())**10
sage: R.<x> = F['x', sigma]
sage: alpha = t
sage: RS_C = SkewRSCode(hamming_dist=5,
   skew_polynomial_ring=R, alpha=alpha)
sage: D = SkewRSCodeSugiyamaDecoder(RS_C)
sage: D
Decoder through the Sugiyama like algorithm of the [6,
   2] Skew Reed Solomon Code over Finite Field in t
of size 2^12
```

MÉTODOS

decode_to_code(self, word) Decodifica el vector `word` en un elemento del código asociado a `self`.

ARGUMENTOS

word Vector de elemtos del mismo cuerpo sobre el que está construido el anillo de polinomios sesgados del código asociado a `self`.

EJEMPLOS

```
sage: F.<t> = GF(2^12, modulus=x**12 + x**7 +
              x**6 + x**5 + x**3 + x +1)
sage: sigma = (F.frobenius_endomorphism())**10
sage: R.<x> = F['x', sigma]
sage: alpha = t
sage: RS_C = SkewRSCode(hamming_dist=5,
   skew_polynomial_ring=R, alpha=alpha)
sage: D = SkewRSCodeSugiyamaDecoder(RS_C)
sage: P_E = SkewCyclicCodePolynomialEncoder(RS_C)
sage: m = x + t
sage: codeword = P_E.encode(m)
```

```
sage: noisy_codeword = copy(codeword)
sage: noisy_codeword[3] = t**671
sage: decoded_word = D.decode_to_code(
    noisy_codeword)
sage: codeword == decoded_word
True
```

decoding_radius(self) Devuelve el número de errores que `self` es capaz de corregir.

EJEMPLOS

```
sage: F.<t> = GF(2^12, modulus=x**12 + x**7 +
              x**6 + x**5 + x**3 + x +1)
sage: sigma = (F.frobenius_endomorphism())**10
sage: R.<x> = F['x', sigma]
sage: alpha = t
sage: RS_C = SkewRSCode(hamming_dist=5,
    skew_polynomial_ring=R, alpha=alpha)
sage: D = SkewRSCodeSugiyamaDecoder(RS_C)
sage: D.decoding_radius()
2
```

4.5. Funciones auxiliares

left_extended_euclidean_algorithm(skew_polynomial_ring, f, g) Implementación del algoritmo extendido de Euclides a la izquierda.

ARGUMENTOS

skew_polynomial_ring anillo de polinomios sesgados

f polinomio del anillo pasado como argumento

g polinomio del anillo pasado como argumento de grado menor que `f`

SALIDA

u, v r vectores de polinomios sesgados tales que `u[i]*f + v[i]*g = r[i]` para cualquier posición `i`

EJEMPLOS

```
sage: F.<t> = GF(3^10)
sage: sigma = F.frobenius_endomorphism()
sage: R.<x> = F['x', sigma]
sage: a = x + t +1;
```

```
sage: b = t^2 * x**2 + (t+1)*x +1
sage: f = a*b
sage: g = b
sage: left_extended_euclidean_algorithm(R, f, g)
([1, 0, 1], [0, 1, 2*x + 2*t + 2], [t^6*x^3 + (2*t^3 +
    t^2 + 1)*x^2 + (t^2 + 2*t + 2)*x + t + 1, t^2*x^2 +
    (t + 1)*x + 1, 0])
```

right_extended_euclidean_algorithm(skew_polynomial_ring, f, g) Implementación del algoritmo extendido de Euclides a la derecha.

ARGUMENTOS

skew_polynomial_ring anillo de polinomios sesgados

f polinomio del anillo pasado como argumento

g polinomio del anillo pasado como argumento de grado menor que `f`

SALIDA

u, v r vectores de polinomios sesgados tales que `f*u[i] + g*v[i] = r[i]` para cualquier posición `i`

EJEMPLOS

```
sage: F.<t> = GF(3^10)
sage: sigma = F.frobenius_endomorphism()
sage: R.<x> = F['x', sigma]
sage: a = x + t +1;
sage: b = t^2 * x**2 + (t+1)*x +1
sage: f = a*b
sage: g = a
sage: right_extended_euclidean_algorithm(R, f, g)
([1, 0, 1], [0, 1, 2*t^2*x^2 + (2*t + 2)*x + 2], [t^6*x
    ^3 + (2*t^3 + t^2 + 1)*x^2 + (t^2 + 2*t + 2)*x + t +
     1, x + t + 1, 0])
```

left_lcm(pols) Calcula el mínimo común múltiplo a la izquierda para una lista de polinomios sesgados

ARGUMENTOS

pols Lista de polinomios sesgados

EJEMPLOS

```
sage: F.<t> = GF(3^10)
sage: sigma = F.frobenius_endomorphism()
```

```
3 sage: R.<x> = F['x', sigma]
4 sage: a = x + t +1;
5 sage: b = t^2 * x**2 + (t+1)*x +1
6 sage: left_lcm((a,b))
7 x^3 + (2*t^9 + 2*t^8 + 2*t^7 + t^6 + t^5 + 2*t^4 + t^2
      + t)*x^2 + (2*t^9 + t^8 + 2*t^7 + 2*t^6 + 2*t^5 + 2*
      t^3 + t)*x + t^8 + t^6 + 2*t^4 + t^3 + 2*t + 2
```

norm(j, sigma, gamma) Calcula la norma `j`-ésima de `gamma`.

ARGUMENTOS

j Número entero

sigma Automorfismo del cuerpo donde esté definido `gamma`

gamma Un elemento de un cuerpo cualquiera

EJEMPLOS

```
1 sage: F.<t> = GF(3^10)
2 sage: sigma = F.frobenius_endomorphism()
3 sage: R.<x> = F['x', sigma]
4 sage: norm(2, sigma, t**2 + t)
5 t^8 + t^7 + t^5 + t^4
```

Conclusiones

Como dijimos en la introducción de este trabajo, los objetivos de este eran estudiar los fundamentos de los códigos lineales, las extensiones de Ore, y la versión del algoritmo de Sugiyama para la familia de códigos skew RS, así como llevar a cabo la implementación del algoritmo y la representación de estos códigos en el entorno SageMath.

Podemos concluir que todos estos objetivos han sido alcanzados. En primer lugar presentamos los códigos lineales, explicando su estructura y como se aprovecha esta en el proceso de transmisión de un mensaje. Seguidamente estudiamos la extensiones de Ore, que son la base de la familia de códigos que utiliza la versión del algoritmo de Sugiyama que aquí presentamos. Tras este estudio, introducimos la familia de códigos cíclicos sesgados, y en particular la de códigos skew RS, y tras el análisis de estas pudimos mostrar el algoritmo de Sugiyama para códigos skew RS y demostrar su correcto funcionamiento. Por último, una vez comprendido el fundamento teórico de este algoritmo, este fue implementada en el entorno SageMath junto con las estructuras necesarias para representar dicha familia de códigos, y cuya documentación se encuentra incluida en el último capítulo.

Como futuro trabajo se podría proponer añadir distintos decodificadores basados en otros algoritmos utilizando las clases aquí diseñadas, así como contribuir al entorno SageMath con estas.

Bibliografía

Las referencias se listan por orden alfabético. Aquellas referencias con más de un autor están ordenadas de acuerdo con el primer autor.

[BGTV03] José Luis Bueso, José Gómez-Torrecillas, and Alain Verschoren. *Algorithmic methods in non-commutative algebra: Applications to quantum groups*, volume 17. Springer Science & Business Media, 2003.

[BGU07] Delphine Boucher, Willi Geiselmann, and Félix Ulmer. Skew-cyclic codes. *Applicable Algebra in Engineering, Communication and Computing*, 18(4):379–389, 2007.

[BRC60a] Raj Chandra Bose and Dwijendra K. Ray-Chaudhuri. Further results on error correcting binary group codes. *Information and Control*, 3(3):279–290, 1960.

[BRC60b] Raj Chandra Bose and Dwijendra K. Ray-Chaudhuri. On a class of error correcting binary group codes. *Information and Control*, 3(1):68–79, 1960.

[BU09a] Delphine Boucher and Felix Ulmer. Codes as modules over skew polynomial rings. In *IMA International Conference on Cryptography and Coding*, pages 38–55. Springer, 2009.

[BU09b] Delphine Boucher and Felix Ulmer. Coding with skew polynomial rings. *Journal of Symbolic Computation*, 44(12):1644–1656, 2009.

[CLU09] Lionel Chaussade, Pierre Loidreau, and Felix Ulmer. Skew codes of prescribed distance or rank. *Designs, Codes and Cryptography*, 50(3):267–284, 2009.

[GT] José Gómez-Torrecillas. Álgebra I, apuntes curso 2019/2020.

[GTLN16] José Gómez-Torrecillas, FJ Lobillo, and Gabriel Navarro. A new perspective of cyclicity in convolutional codes. *IEEE Transactions on Information Theory*, 62(5):2702–2706, 2016.

[GTLN17] J. Gómez-Torrecillas, F. J. Lobillo, and G. Navarro. A Sugiyama-like decoding algorithm for convolutional codes. *IEEE Transactions on Information Theory*, 63(10):6216–6226, 2017.

[Hoc59] Alexis Hocquenghem. Codes correcteurs d'erreurs. *Chiffres*, 2(2):147–56, 1959.

[HP03] W. Cary Huffman and Vera Pless. *Fundamentals of Error-Correcting Codes*. Cambridge University Press, 2003.

[Ler95] André Leroy. Pseudo linear transformations and evaluation in Ore extensions. *Bull. Belg. Math. Soc. - Simon Stevin*, 2(3):321–347, 1995.

[LL88] T. Y. Lam and A. Leroy. Vandermonde and Wronskian matrices over division rings. *Journal of Algebra*, 119(2):308–336, 1988.

[Ore33] Oystein Ore. Theory of non-commutative polynomials. *Annals of Mathematics*, 34(3):pp. 480–508, 1933.

[RS60] Irving S. Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

[S+20] W. A. Stein et al. *Sage Mathematics Software (Version 9.1)*. The Sage Development Team, 2020. `http://www.sagemath.org`.

[Sha48] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(4):623–656, 1948.

[SKHN75] Yasuo Sugiyama, Masao Kasahara, Shigeichi Hirasawa, and Toshihiko Namekawa. A method for solving key equation for decoding goppa codes. *Information and Control*, 27(1):87–99, 1975.

Agradecimientos

En primer lugar, agredecer a mis dos tutores, José Gómez Torrecillas y Francisco Javier Lobillo Borrego por su gran ayuda, consejos y completa disposición a lo largo del desarrollo de todo este trabajo.

Quiero agredecer también a mi familia, a mis padres, Guillermo Galindo Maqueda Ruiz y María Isabel Ortuño Ruiz, y a mi hermanos, Luis Galindo Ortuño e Isabel Galindo Ortuño por todo el apoyo, la confianza, y el cariño recibidos durante todos estos años, y sin los cuales no habría podido llegar hasta aquí.

A mis amigos, tanto de la residencia como compañeros de clase, que han hecho que mi paso por Granada sea algo que no voy a olvidar, y que han estado presentes tanto en los mejores como en los no tan buenos momentos de estos cinco años.

Por último, agredecer a toda la comunidad que contribuye en SageMath, por hacer posible la implementación de este trabajo, y por su aportación al conocimiento científico y al sofware de forma libre y gratuita.