UNIVERSITAT DE BARCELONA
Facultat de Matemàtiques i Informàtica

Treball final del grau de Matemàtiques i Informàtica

Descodificacions dels codis BCH

Enrique Ji Wang

Director: Dr. Artur Travesa i Grau
Barcelona, 17 de gener de 2024

Abstract

Error-correcting codes play a very important role in the transmission and storage of data, allowing for the correction of errors that may occur during these processes. Among these codes, BCH codes (Bose-Chaudhuri-Hocquenghem) stand out for their efficiency and widespread use. In this thesis, we will study BCH codes and, in particular, focus on the decoding problem. We will explain three decoding algorithms for these codes and implement them in the C programming language.

Resum

Els codis correctors d'errors tenen un paper molt important en la transmissió i emmagatzematge de dades, permeten corregir errors que s'hagin produït durant aquests processos. Entre aquests codis, els codis BCH (Bose-Chaudhuri-Hocquenghem) destaquen per la seva eficiència i àmplia utilització. En aquest treball estudiarem els codis BCH, i en particular, ens centrarem en el problema de la descodificació. Explicarem tres algoritmes de descodificació per a aquests codis, i els implementarem en el llenguatge de programació C.

Resumen

Los códigos correctores de errores desempeñan un papel muy importante en la transmisión y almacenamiento de datos, permitiendo corregir errores que puedan ocurrir durante estos procesos. Entre estos códigos, los códigos BCH (Bose-Chaudhuri-Hocquenghem) destacan por su eficiencia y amplio uso. En este trabajo, estudiaremos los códigos BCH y, en particular, nos centraremos en el problema de la decodificación. Explicaremos tres algoritmos de decodificación para estos códigos y los implementaremos en el lenguaje de programación C.

2020 Mathematics Subject Classification. 11T71, 68P30, 94B15, 94B35

Agraïments

Vull agrair al meu tutor, el Dr. Artur Travesa i Grau, l'ajuda i la direcció durant el desenvolupament d'aquest treball.

Índex

1 Introducció **1**

1.1 Canal de comunicació 1

1.2 Cossos finits 1

1.3 Factorització de $x^n - 1$ 3

1.4 Algoritme d'Euclides 4

2 Codis lineals **7**

2.1 Codis lineals 7

2.2 Distàncies i pesos 8

2.3 Descodificació 8

3 Codis cíclics **11**

3.1 Polinomi generador 11

3.2 Zeros d'un codi cíclic 13

3.3 Codificació 14

4 Codis BCH **15**

4.1 Fita BCH 15

4.2 Codis BCH 16

4.3 Descodificació 17

4.3.1 Algoritme Peterson-Gorenstein-Zierler 17

4.3.2 Algoritme Berlekamp-Massey 21

4.3.3 Algoritme Sugiyama 24

5 Implementació **29**

5.1 Inicialització del codi BCH 29

5.2 Codificació 32

5.3 Descodificació per Peterson-Gorenstein-Zierler 32

5.4 Descodificació per Berlekamp-Massey 35

5.5 Descodificació per Sugiyama 37

5.6 Comparació dels algoritmes 38

Referències **43**

1 Introducció

Tot canal de comunicació té una probabilitat de fallar i es pot introduir un error en el missatge que s'envia que corromp la informació. Davant d'aquest problema, sorgeix l'estudi matemàtic de la teoria de la informació i la teoria de codis el 1948[HKS21], quan Claude Shannon publica l'article "A Mathematical Theory of Communication"[Sha48], on defineix el concepte de capacitat d'un canal i prova que la comunicació sense pèrdues és possible per sota d'aquesta capacitat.

Per exemple, quan una sonda espacial envia imatges a la Terra, per les distàncies astronòmiques i la debilitat del senyal, la probabilitat d'error és molt alta. I com que les imatges no es troben guardades a la sonda, sinó que es transmeten directament, això fa que no sigui possible re-transmetre aquestes imatges, per tant, si hi ha un error, es perd part de la informació. Els resultats de Shannon garanteixen que les dades es poden codificar abans de la transmissió de manera que les dades alterades es puguin descodificar correctament amb un cert grau de precisió.

L'emmagatzematge de dades es pot considerar com un altre canal de comunicació, com el cas d'un disc dur. On l'emissor és qui guarda una cançó, per exemple, el canal és el mateix disc dur i el receptor és qui escolta la música. En aquest cas, els errors es poden produir per la degradació dels materials o per ratllades en els discos.

Amb això arribem al problema fonamental de la teoria de codis, determinar quin és el missatge que s'ha enviat només amb la informació que arriba al receptor.

1.1 Canal de comunicació

Un canal de comunicació té la forma següent:

$$\text{Emissor} \longrightarrow \text{Codificació} \longrightarrow \text{Canal} \longrightarrow \text{Descodificació} \longrightarrow \text{Receptor}$$

Figura 1: Diagrama d'un canal de comunicació

Des de l'emissor, tenim un missatge $\mathbf{x} = x_1 \dots x_k$ per ser enviat. Aquest missatge es codifica afegint redundància i obtenim un missatge codificat, que anomenarem paraula, $\mathbf{c} = c_1 \dots c_n$. Aquesta paraula es transmet pel canal, on es pot produir un error $\mathbf{e} = e_1 \dots e_n$ i produeix el vector que es rep $\mathbf{y} = \mathbf{c} + \mathbf{e}$. Finalment, aquest vector es descodifica i s'obté una estimació del missatge original $\hat{\mathbf{x}}$, on esperem que $\hat{\mathbf{x}} = \mathbf{x}$. El teorema de Shannon garanteix que, donades les característiques del canal, podem trobar codificacions de manera que la probabilitat d'èxit sigui tan alta com vulguem, però mai 100%.

Cal notar que el teorema de Shannon no diu com trobar aquestes codificacions, només diu que existeixen. En aquest treball, ens centrarem en els codis BCH, que són una família de codis cíclics molt importants, ja que admeten una codificació i una descodificació eficients.

1.2 Cossos finits

En aquesta secció recordarem algunes propietats dels cossos finits que utilitzarem al llarg del treball. Per a una visió més detallada, es pot consultar [Cre21] i [Tra20], on també es troben les demostracions que s'ometen en aquesta secció.

Definició 1.1. *Un cos és un conjunt amb dues operacions* $(\mathbb{F}, +, \cdot)$ *que satisfan les propietats següents:*

(i) $(\mathbb{F}, +)$ és un grup abelià amb element neutre 0.

(ii) $(\mathbb{F} \setminus \{0\}, \cdot)$ és un grup abelià amb element neutre 1.

(iii) L'operació del producte $\cdot$ distribueix la suma $+$.

Definició 1.2. *Un cos finit és un cos amb un nombre finit d'elements.*

Recordem que anomenem *característica* d'un cos l'enter positiu més petit p tal que $p \cdot 1 = 1 + \cdots + 1 = 0$, on sumem p vegades 1. Si no existeix cap enter positiu p amb aquesta propietat, diem que el cos té característica 0.

Teorema 1.3. *Sigui* $\mathbb{F}_q$ *un cos finit de* q *elements. Aleshores:*

(i) $q = p^m$ per a algun primer p i algun enter positiu m.

(ii) $\mathbb{F}_q$ té característica p.

(iii) $\mathbb{F}_q$ és únic tret d'isomorfismes.

Un cop vist aquest teorema, a partir d'ara denotarem $\mathbb{F}_q$ al cos finit de q elements, p representarà un nombre primer i q un nombre de la forma $q = p^m$ per a algun enter positiu m.

Proposició 1.4. *Sigui* $\mathbb{F}$ *un cos finit de característica* p*, llavors*

$$(\alpha \pm \beta)^{p^n} = \alpha^{p^n} \pm \beta^{p^n}$$

per a qualsevol enter positiu n *i qualssevol* $\alpha, \beta \in \mathbb{F}$.

Per treballar sobre un cos finit, ens interessa poder sumar i multiplicar elements d'aquest cos de manera simple. Per a això, introduïm una nova representació dels elements de $\mathbb{F}_q$ que ens facilitarà realitzar els productes.

Recordem que anomenem $\mathbb{F}_q^* = \mathbb{F}_q \setminus \{0\}$ el grup multiplicatiu de $\mathbb{F}_q$, amb les següents propietats:

Proposició 1.5.

(i) El grup $\mathbb{F}_q^*$ és cíclic d'ordre $q - 1$.

(ii) Sigui γ un generador d'aquest grup cíclic, aleshores

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

i $\gamma^i = 1$ si i només si $q - 1 \mid i$.

Per tant, si agafem un generador γ de $\mathbb{F}_q^*$, podem representar els elements no nuls de $\mathbb{F}_q$ com a potències de γ, on la multiplicació de dos elements es pot fer de manera molt senzilla: $\gamma^i \cdot \gamma^j = \gamma^{i+j} = \gamma^s$, amb $0 \leq s \leq q - 2$ on $i + j = s \pmod{q-1}$.

Utilitzant aquesta representació dels elements de $\mathbb{F}_q$, s'obté la següent caracterització dels cossos finits.

Teorema 1.6. *El cos finit de* q *elements* $\mathbb{F}_q$ *és el conjunt de tots els zeros del polinomi* $f_q(x) = x^q - x \in \mathbb{F}_p$ *i el cos de descomposició* $\mathbb{F}_p$ *sobre de* $f_q(x)$.

Continuant amb el grup multiplicatiu $\mathbb{F}_q^*$, ens interessa el nombre de generadors que té aquest grup i com trobar-los tots un cop hem determinat un d'ells. Per a això cal definir la funció ϕ d'Euler, $\phi(n)$ és el nombre d'enters i amb $1 \leq i \leq n$ i $\text{mcd}(i,n) = 1$. Aquí s'ha fet servir la notació $\text{mcd}(i,j)$ per a denotar el màxim comú divisor entre i i j.

Proposició 1.7. *Sigui γ un generador del grup multiplicatiu $\mathbb{F}_q^*$ de $\mathbb{F}_q$.*

(i) Hi ha $\phi(q-1)$ generadors de $\mathbb{F}_q^*$, que són els elements γ^i amb $(i, q-1) = 1$.

(ii) Per a cada d tal que $d \mid q-1$, hi ha $\phi(d)$ elements de $\mathbb{F}_q^*$ d'ordre d, aquests són els elements $\gamma^{(q-1)i/d}$ amb $\text{mcd}(i,d) = 1$.

Un element $\xi \in \mathbb{F}_q$ és una *arrel n-èsima de la unitat* si $\xi^n = 1$ i és una *arrel primitiva n-èsima de la unitat* si a més $\xi^i \neq 1$ per a tot i amb $1 \leq i < n$. Llavors un element generador γ de $\mathbb{F}_q^*$ és una arrel primitiva $(q-1)$-èsima de la unitat. També segueix de la proposició 1.5 que el cos $\mathbb{F}_q$ conté una arrel primitiva n-èsima de la unitat si i només si $n \mid q-1$, i en aquest cas $\gamma^{(q-1)/n}$ és una arrel primitiva n-èsima de la unitat.

Ara veurem una representació per a realitzar fàcilment l'operació de suma en els cossos finits. Recordem primer que l'anell dels polinomis $\mathbb{F}_q[x]$ és un domini d'ideals principals per ser $\mathbb{F}_q$ un cos i, per tant, també un domini de factorització única. A partir d'això, s'obté la següent caracterització dels cossos finits.

Proposició 1.8. *Sigui $f(x)$ un polinomi irreductible de grau m de $\mathbb{F}_p[x]$. Aleshores*

$$\mathbb{F}_p[x]/\langle f(x)\rangle = \{r(x) + \langle p(x)\rangle \mid grau(r(x)) < m\}$$

és un cos finit de p^m elements.

D'aquesta proposició, identifiquem els elements de $\mathbb{F}_q$ amb els polinomis de grau menor que m, amb la suma i el producte mòdul $p(x)$. Per a simplificar la notació, es fa servir la següent correspondència:

$$g(x) + \langle f(x)\rangle = g_{m-1}x^{m-1} + \cdots + g_1 x + g_0 + \langle f(x)\rangle \leftrightarrow g_{m-1} \ldots g_1 g_0.$$

Aquesta notació vectorial ens permet fer la suma de la manera ordinària (component a component).

Finalment, cal recordar el concepte de polinomis mínims.

Definició 1.9. *Sigui α un element algebraic sobre un cos $\mathbb{F}_q$. El polinomi mínim de α és el polinomi mònic $M_\alpha(x) \in \mathbb{F}_q[x]$ de grau més petit tal que $M_\alpha(\alpha) = 0$. Dos elements algebraics de $\mathbb{F}_q$ són conjugats si tenen el mateix polinomi mínim.*

Teorema 1.10.

(i) El polinomi mínim de α sobre $\mathbb{F}_q$ és irreductible sobre $\mathbb{F}_q$.

(ii) Si $g(x)$ és un polinomi de $\mathbb{F}_q[x]$ tal que $g(\alpha) = 0$, aleshores $M_\alpha(x) \mid g(x)$.

(iii) $M_\alpha(x)$ és únic.

1.3 Factorització de $x^n - 1$

En aquesta secció recordarem quins són els factors irreductibles de $x^n - 1$ sobre $\mathbb{F}_q$, que ens serà molt útil per als codis cíclics més endavant.

Proposició 1.11. *El polinomi $x^n - 1$ sobre $\mathbb{F}_q$ no té arrels repetides si i només si n i q són coprimers.*

Demostració. Suposem que n i q no són coprimers, llavors podem escriure $n = mp^k$, on $\text{mcd}(m, q) = 1$ i p és la característica de $\mathbb{F}_q$. Per tant, tenim que

$$x^n - 1 = x^{mp^k} - 1 = (x^m - 1)^{p^k},$$

que té factor repetit $x^m - 1$.

Recíprocament, si n i q són coprimers, tenim que el polinomi derivat, $D(x^n-1, x) = nx^{n-1}$, no té arrels comunes amb $x^n - 1$. Per tant, $x^n - 1$ no té arrels repetides. □

Com que només ens interessa el cas en què $x^n - 1$ no té arrels repetides, a partir d'ara assumirem que n i q són coprimers.

Definició 1.12. *Sigui s un enter tal que $0 \leq s < n$, anomenem la classe q-ciclotòmica de s mòdul n al conjunt*

$$C_s = \{s, sq, \ldots, sq^{r-1}\} (\text{mod } n),$$

on r és l'enter positiu més petit tal que $sq^r = s (\text{mod } n)$.

Definim *ordre* de q mòdul n, $ord_n(q)$, com el nombre més petit s tal que $q^s \equiv 1 \pmod{n}$. Per tant, si agafem $t = ord_n(q)$, tenim que $\mathbb{F}_{q^t}$ és el cos de descomposició de $x^n - 1$ sobre $\mathbb{F}_q$. De les propietats dels polinomis mínims, obtenim el següent teorema.

Teorema 1.13. *Sigui n un enter positiu coprimer amb q, $t = ord_n(q)$ i α una arrel primitiva n-èsima de la unitat en $\mathbb{F}_{q^t}$.*

(i) Per a cada enter s amb $0 \leq s < n$, el polinomi mínim de α^s sobre $\mathbb{F}_q$ és

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

on C_s és la classe q-ciclotòmica de s mòdul n.

(ii) Els conjugats de α^s són els elements α^i amb $i \in C_s$.

(iii) La factorització de $x^n - 1$ en factors irreductibles sobre $\mathbb{F}_q$ és

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

on s recorre un conjunt de representants de les classes q-ciclotòmiques mòdul n.

1.4 Algoritme d'Euclides

Acabem aquest capítol recordant l'algoritme d'Euclides, que ens serà molt útil per a trobar el màxim comú divisor de dos polinomis. Primer de tot, recordem la divisió entera de polinomis.

Proposició 1.14. *Sigui $f(x), g(x)$ dos polinomis de $\mathbb{F}_q[x]$ amb $g(x) \neq 0$.*

(i) Existeixen polinomis $h(x), r(x) \in \mathbb{F}_q[x]$, únics per als quals se satisfà que $f(x) = g(x)h(x) + r(x)$, on $\text{gr}(r(x)) < \text{gr}(g(x))$ or $r(x) = 0$.

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

Podem utilitzar aquesta divisió recursivament per a trobar el màxim comú divisor de dos polinomis. Aquest procés es coneix com l'algoritme d'Euclides.

Teorema 1.15. *Sigui $f(x), g(x)$ dos polinomis de $\mathbb{F}_q[x]$ amb $g(x) \neq 0$.*

(i) Repetim el següent procés fins que $r_n(x) = 0$ per a algun n:

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

Llavors $\operatorname{mcd}(f(x), g(x)) = cr_{n-1}(x)$, on $c \in \mathbb{F}_q$ s'escull per a què $r_{n-1}(x)$ sigui mònic.

(ii) Existeixem polinomis $a(x), b(x) \in \mathbb{F}_q[x]$ tals que

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

2 Codis lineals

Els codis lineals són un tipus de codis molt estudiats i que tenen una gran quantitat d'aplicacions gràcies a la seva estructura algebraica. En aquest capítol introduirem la definició de codi lineal i estudiarem algunes de les seves propietats. També aprofitarem per a introduir conceptes bàsics de la teoria de codis utilitzant aquests codis.

2.1 Codis lineals

Utilitzem $\mathbb{F}_q^n$ per a denotar l'espai vectorial de dimensió n sobre $\mathbb{F}_q$, que és el conjunt de vectors de longitud n de coeficients en $\mathbb{F}_q$.

Un codi $\mathcal{C}$ de longitud n sobre $\mathbb{F}_q$ és un subconjunt de $\mathbb{F}_q^n$. Farem servir la notació $a_0a_1\dots a_{n-1}$ per a escriure els vectors $(a_0, a_1, \dots, a_{n-1})$ de $\mathbb{F}_q^n$ i anomenarem els vectors de $\mathcal{C}$ *paraules del codi*.

Per a què un codi sigui més fàcil d'estudiar, és còmode que tingui una estructura, per a això imposem la linealitat.

Definició 2.1. *Un codi $\mathcal{C}$ de longitud n sobre $\mathbb{F}_q$ és lineal si és un subespai vectorial de $\mathbb{F}_q^n$. Si $\mathcal{C}$ és un codi lineal de dimensió k, direm que $\mathcal{C}$ és un codi lineal de longitud n i dimensió k sobre $\mathbb{F}_q$.*

Anomenarem els codis sobre $\mathbb{F}_2$ *codis binaris*.

Observació 2.2. Directament de la definició, tenim que un codi lineal de longitud n i dimensió k sobre $\mathbb{F}_q$ té q^k paraules.

Com que els codis lineals són espais vectorials, podem descriure'ls donant-ne una base.

Definició 2.3. *Una matriu generadora d'un codi lineal $\mathcal{C}$ de longitud n i dimensió k és qualsevol matriu G de mida $k \times n$ tal que les files formen una base de $\mathcal{C}$.*

Si $\mathcal{C}$ és un codi amb matriu generadora G, tenim que les paraules del codi són exactament les combinacions lineals de les files de G, ja que

$$\mathcal{C} = \{xG \mid x \in \mathbb{F}_q^k\}.$$

Per tant, tenim un mètode molt simple per a codificar un missatge: donat un missatge $x \in \mathbb{F}_q^k$, el codifiquem com xG.

Exemple 2.4. Considerem el codi binari amb matriu generadora

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

Llavors podem codificar els missatges $\mathbf{x} = (x_0, x_1, x_2) \in \mathbb{F}_2^3$ com

$$\begin{bmatrix} x_0 & x_1 & x_2 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} = (x_0 + x_2, x_0 + x_1, x_1 + x_2, x_1).$$

2.2 Distàncies i pesos

Una invariant fonamental dels codis és la distància mínima entre les seves paraules.

Definició 2.5. *La distància (de Hamming) entre dos vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}_q^n$ és el nombre de components en què difereixen:*

$$d(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{n} \delta(x_i, y_i),$$

on $\delta(x_i, y_i) = 1$ si $x_i \neq y_i$ i $\delta(x_i, y_i) = 0$ si $x_i = y_i$.

Proposició 2.6. *La funció d satisfà les propietats següents:*

(i) (no negativitat) $\mathrm{d}(\mathbf{x}, \mathbf{y}) \geq 0$ per a tot $\mathbf{x}, \mathbf{y} \in \mathbb{F}_q^n$.

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

(iii) (simetria) $\mathrm{d}(\mathbf{x}, \mathbf{y}) = \mathrm{d}(\mathbf{y}, \mathbf{x})$ per a tot $\mathbf{x}, \mathbf{y} \in \mathbb{F}_q^n$.

(iv) (desigualtat triangular) $\mathrm{d}(\mathbf{x}, \mathbf{z}) \leq \mathrm{d}(\mathbf{x}, \mathbf{y}) + \mathrm{d}(\mathbf{y}, \mathbf{z})$ per a tot $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{F}_q^n$.

Demostració. Les propietats (i), (ii) i (iii) són evidents de la definició. Per demostrar la propietat (iv), observem que si hi ha una diferència en la posició i entre $\mathbf{x}$ i $\mathbf{z}$, llavors hi ha una diferència en la posició i entre $\mathbf{x}$ i $\mathbf{y}$ o entre $\mathbf{y}$ i $\mathbf{z}$. Per tant, la distància entre $\mathbf{x}$ i $\mathbf{z}$ no pot ser més gran que la suma de les distàncies entre $\mathbf{x}$ i $\mathbf{y}$ i entre $\mathbf{y}$ i $\mathbf{z}$. □

Amb aquest teorema veiem que la distància és una mètrica sobre $\mathbb{F}_q^n$.

Definició 2.7. *La distància (mínima) d'un codi $\mathcal{C}$ és la distància més petita entre dues paraules diferents del codi.*

Definició 2.8. *El pes (Hamming) $wt(\boldsymbol{x})$ d'un vector $\boldsymbol{x} \in \mathbb{F}_q^n$ és el nombre de components no nuls de $\boldsymbol{x}$.*

Observació 2.9. De la definició de pes, veiem que $wt(\mathbf{x}) = d(\mathbf{x}, \mathbf{0})$.

Teorema 2.10. *Si $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}_q^n$, llavors $d(\boldsymbol{x}, \boldsymbol{y}) = wt(\boldsymbol{x} - \boldsymbol{y})$. Si $\mathcal{C}$ és un codi lineal, llavors la distància mínima d es igual al pes mínim de les paraules no nul·les del codi.*

Demostració. Per a la primera afirmació, veiem com el vector $\mathbf{x} - \mathbf{y}$ té un 0 exactament en les posicions on són iguals. Per tant, el pes de $\mathbf{x} - \mathbf{y}$ és la distància entre $\mathbf{x}$ i $\mathbf{y}$. Per a la segona, suposem que $d = d(\mathbf{x}, \mathbf{y})$ és la distància mínima entre dues paraules diferents del codi $\mathcal{C}$. Com que el codi es lineal, tenim que el vector $\mathbf{x} - \mathbf{y} \in \mathcal{C}$ té pes d i per tant, $w \leq d$, on w denota el pes mínim. L'altra desigualtat és directe de l'observació anterior, per tant, $d = w$, com volíem veure. □

Gràcies a aquest teorema, per als codis lineals, la distància mínima també és anomenada el *pes mínim* del codi. Si el pes mínim d d'un codi de longitud n i dimensió k és conegut, direm que el codi té paràmetres $[n, k, d]$.

2.3 Descodificació

La descodificació és el procés de recuperar el missatge original $\mathbf{x}$ a partir del missatge rebut $\mathbf{y}$, o, equivalentment, trobar la paraula del codi $\mathbf{c}$ que s'ha enviat, ja que ja hem vist que hi ha una correspondència bijectiva entre les paraules del codi i els missatges.

Aquest procés consisteix a trobar la paraula del codi $\mathbf{c}$ més propera a $\mathbf{y}$, es a dir, minimitzar $d(\mathbf{y}, \mathbf{c})$. Sigui $\mathbf{e}$ el vector error afegit pel canal de comunicació tal que $\mathbf{y} = \mathbf{c} + \mathbf{e}$, la descodificació és equivalent a trobar el vector $\mathbf{e}$ amb pes mínim tal que $\mathbf{y} - \mathbf{e}$ es trobi en el codi.

Definició 2.11. *Una esfera de radi r centrada en el vector $\boldsymbol{u} \in \mathbb{F}_q^n$ és el conjunt*

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

Aquestes esferes són disjuntes dos a dos si agafem un radi prou petit, en particular, tenim el següent resultat:

Teorema 2.12. *Si d és la distància mínima d'un codi $\mathcal{C}$ i $t = \lfloor (d-1)/2 \rfloor$, llavors les esferes de radi t centrades en les paraules del codi són disjuntes dos a dos.*

Demostració. Suposem que hi ha dues paraules $\mathbf{u}$ i $\mathbf{v}$ del codi tals que les seves esferes no són disjuntes, llavors existeix un vector $\mathbf{z} \in S_r(\mathbf{u}) \cap S_r(\mathbf{v})$. Per la desigualtat triangular tenim que

$$d(\mathbf{u}, \mathbf{v}) \leq d(\mathbf{u}, \mathbf{z}) + d(\mathbf{z}, \mathbf{v}) \leq 2t < d.$$

Com que d és la distància mínima, tenim que $\mathbf{u} = \mathbf{v}$. □

Corol·lari 2.13. *Utilitzant la notació del teorema anterior, si la paraula $\mathbf{c}$ és enviada i es rep el vector $\mathbf{y}$ amb t o menys errors, llavors $\mathbf{c}$ és l'única paraula del codi que es troba la més propera a $\mathbf{y}$. En particular, es descodificarà únicament i correctament qualsevol vector que com a molt tingui t errors.*

Aquest corol·lari implica que donades n i k, volem trobar el codi amb el pes mínim d més gran possible. I de la mateixa manera, donades n i d, volem enviar el missatge més llarg possible, que en el cas lineal, és trobar el codi amb la dimensió k més gran possible.

Aprofitem per a introduir la següent definició:

Definició 2.14. *Diem que un codi $\mathcal{C}$ corregeix t errors si pot descodificar correctament qualsevol vector amb t o menys errors. De la mateixa manera, diem que $\mathcal{C}$ detecta s errors si pot detectar que hi ha hagut un error si el nombre d'errors és menor o igual que s.*

Continuant amb la notació del teorema, tenim que $\mathcal{C}$ corregeix t erros i detecta $d-1$ errors. Aquesta última afirmació es dedueix del fet que la distància mínima entre dues paraules del codi és d, si hi ha $d-1$ errors o menys (però no cap), el vector rebut no es troba en el codi.

Intuïm que la descodificació és molt més complexa que la codificació, per tant, el problema ara és trobar un mètode eficient per a corregir fins a t errors. L'algoritme més senzill és examinar totes les paraules del codi i trobar la més propera a $\mathbf{y}$, però òbviament això només és viable per a codis petits. Per a codis grans, necessitem mètodes més sofisticats, que existeixen per a les diferents famílies de codis. En aquest treball ens centrarem només en els codis BCH, en les seccions que veurem més endavant.

3 Codis cíclics

En aquest capítol estudiarem els codis cíclics, que són una família de codis lineals amb una estructura molt particular. Moltes famílies de codis molt importants són cíclics, com els codis de Golay o els Reed-Muller, i en particular els codis BCH.

En l'estudi dels codis cíclics, assumim que n i q son coprimers.

Definició 3.1. *Un codi lineal $\mathcal{C} \in \mathbb{F}_q^n$ és cíclic si per a tota paraula $\boldsymbol{c} = c_0c_1 \dots c_{n-1} \in \mathcal{C}$, també ho és la paraula $\boldsymbol{c}' = c_{n-1}c_0c_1 \dots c_{n-2}$, que s'obté fent un desplaçament cíclic sobre les coordenades, $i \mapsto i + 1 (\mathrm{mod}\ n)$.*

És convenient quan tractem de codis cíclics utilitzar $0, 1, \dots, n-1$ com a índexs de les coordenades i pensar-los com a enters mòdul n. Per tant, quan parlem de coordenades consecutives, sempre ho farem en el sentit cíclic, és a dir, després de l'índex $n-1$ ve el 0. Aquest conveni també ens facilitarà usar la representació en formes polinomials de les paraules del codi que definirem tot seguit.

Quan parlem de codis cíclics sobre $\mathbb{F}_q$, tenim una correspondència bijectiva entre el vector $\mathbf{c} = c_0c_1 \dots c_{n-1}$ en $\mathbb{F}_q^n$ i el polinomi $c(x) = c_0 + c_1x + \cdots + c_{n-1}x^{n-1}$ en $\mathbb{F}_q[x]$ de grau màxim $n-1$. Com és habitual, farem servir les dues representacions indistintament, ignorant la bijecció que apliquem entre elles.

Notem que si $c(x) = c_0 + c_1x + \cdots + c_{n-1}x^{n-1}$, llavors $xc(x) = c_{n-1}x^n + c_0x + \cdots + c_{n-2}x^{n-1}$, que és el polinomi corresponent a la paraula $\mathbf{c}' = c_{n-1}c_0c_1 \dots c_{n-2}$ si assignen el valor de x^n a 1. Més formalment, que un codi cíclic $\mathcal{C}$ sigui invariant sota el desplaçament cíclic implica que si $c(x) \in \mathcal{C}$ llavors $xc(x)$ també ho és si fem el producte en mòdul $x^n - 1$. Per tant, estudiarem els codis cíclics en l'anell quocient

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

Utilitzant la representació polinomial, obtenim una definició alternativa de codi cíclic.

Proposició 3.2. *Un codi lineal $\mathcal{C} \in \mathbb{F}_q^n$ és cíclic si és un ideal de $\mathcal{R}_n$.*

3.1 Polinomi generador

Teorema 3.3. *Sigui $\mathcal{C}$ un codi cíclic no trivial (o sigui, diferent de zero) de $\mathcal{R}_n$, existeix un polinomi $g(x) \in \mathcal{R}_n$ amb les següents propietats:*

(i) $g(x)$ és l'únic polinomi mònic de grau mínim de $\mathcal{C}$.

(ii) $\mathcal{C} = \langle g(x) \rangle$.

(iii) $g(x) \mid x^n - 1$.

Sigui $k = n - \mathrm{gr}(g(x))$ i $g(x) = \sum_{i=0}^{n-k} g_i x^i$, on $g_{n-k} = 1$. Llavors:

(iv) La dimensió de $\mathcal{C}$ és k i una base de $\mathcal{C}$ és $\{g(x), xg(x), \dots, x^{k-1}g(x)\}$.

(v) Cada element de $\mathcal{C}$ es pot escriure de forma única com el producte $g(x)f(x)$, on $f(x) = 0$ o $\mathrm{gr}(f(x)) < k$.

(vi)

$$G = \begin{bmatrix} g_0 & g_1 & g_2 & \cdots & g_{n-k} & & 0 \\ 0 & g_0 & g_1 & \cdots & g_{n-k-1} & g_{n-k} & \\ & \cdots & \cdots & \cdots & \cdots & \cdots & \\ 0 & & g_0 & & \cdots & & g_{n-k} \end{bmatrix}$$

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

és una matriu generadora de $\mathcal{C}$.

(vii) Si α és una arrel primitiva n-èsima de la unitat en algun cos extensió de $\mathbb{F}_q$, llavors

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

on el producte es fa sobre un subconjunt de representants de les classes q-ciclotòmiques mòdul n.

Demostració. Sigui $g(x)$ un polinomi mònic de grau mínim de $\mathcal{C}$, que existeix per ser $\mathcal{C}$ no trivial. Si $c(x) \in \mathcal{C}$, llavors $c(x) = g(x)h(x) + r(x)$, on $r(x) = 0$ o $\text{gr}(r(x)) < \text{gr}(g(x))$. Com que $\mathcal{C}$ és un ideal de $\mathcal{R}_n$, tenim que $r(x) \in \mathcal{C}$ i com estem suposant que $g(x)$ és de grau mínim, llavors $r(x) = 0$. Això demostra (i) i (ii).

Si dividim $x^n - 1$ per $g(x)$, obtenim que $x^n - 1 = g(x)h(x) + r(x)$, on altre cop $r(x) = 0$ o $\text{gr}(r(x)) < \text{gr}(g(x))$. Com que $x^n - 1$ correspon a la paraula 0 en $\mathcal{C}$ i $\mathcal{C}$ és un ideal de $\mathcal{R}_n$, tenim que $r(x) \in \mathcal{C}$ i necessàriament $r(x) = 0$, per tant, provant (iii).

Suposem que $\text{gr}(g(x)) = n - k$. Per (ii) i (iii), si $c(x) \in \mathcal{C}$ amb $c(x) = 0$ o $\text{gr}(c(x)) < n$, llavors $c(x) = g(x)f(x)$. Si $c(x) = 0$, tenim que $f(x) = 0$, i si $c(x) \neq 0$, llavors $\text{gr}(f(x)) < k$. Per tant, veiem que

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

Per tant, $\mathcal{C}$ té dimensió com a molt k i $\{g(x), xg(x), \ldots, x^{k-1}g(x)\}$ és un conjunt de generadors de $\mathcal{C}$. Com que aquests k polinomis són de graus diferents, són linealment independents en $\mathbb{F}_q[x]$. I com que tenen com a grau màxim $n-1$, continuen sent linealment independents en $\mathcal{R}_n$, demostrant (iv) i (v).

L'apartat (vi) és directe de la definició de matriu generadora utilitzant la base anterior i (vii) segueix del Teorema 1.13. □

Notem que de l'apartat (ii) del teorema, també hem vist que $\mathcal{R}_n$ és un anell d'ideals principals.

Corol·lari 3.4. *Sigui $\mathcal{C}$ un codi cíclic no trivial de $\mathcal{R}_n$, són equivalents:*

(i) $g(x)$ és el polinomi mònic de grau mínim de $\mathcal{C}$.

(ii) $\mathcal{C} = \langle g(x) \rangle$, $g(x)$ és mònic i $g(x) \mid x^n - 1$.

Demostració. (i) $\Rightarrow$ (ii) ja s'ha demostrat al Teorema 3.3.

Suposem el cas (ii) i sigui $g'(x)$ el polinomi mònic de grau mínim de $\mathcal{C}$. Pel mateix Teorema 3.3, tenim que $g'(x) \mid g(x)$ i $\mathcal{C} = \langle g'(x) \rangle$. Com que $g'(x) \in \mathcal{C} = \langle g(x) \rangle$, tenim

que $g'(x) \equiv g(x)a(x) \pmod{x^n-1}$ i per tant $g'(x) = g(x)a(x) + (x^n-1)b(x)$ en $\mathbb{F}_q[x]$. Com que $g(x) \mid x^n-1$, llavors $g(x) \mid g(x)a(x) + (x^n-1)b(x)$, per tant $g(x) \mid g'(x)$. Com que $g(x)$ i $g'(x)$ són mònics, es divideixen l'un a l'altre i $\mathrm{gr}(g'(x)) \leq \mathrm{gr}(g(x))$, tenim que $g(x) = g'(x)$, com volíem demostrar. □

El Teorema 3.3 ens diu que existeix un polinomi mònic $g(x)$ que divideix x^n-1 i que genera $\mathcal{C}$, mentre que el corol·lari ens diu que aquest polinomi és únic. Aquest polinomi és el que anomenem *polinomi generador* de $\mathcal{C}$. Pel corol·lari, aquest polinomi és tant el polinomi mònic de grau mínim de $\mathcal{C}$ com el polinomi mònic de $\mathcal{C}$ que divideix x^n-1. Per tant, tenim una correspondència bijectiva entre els codis cíclics no buits i els divisors de x^n-1 diferents de x^n-1. Per a què aquesta correspondència sigui entre tots els codis cíclics, definim el polinomi generador del codi zero a x^n-1. Directe d'aquesta bijecció, obtenim el següent resultat.

Corol·lari 3.5. *El nombre de codis cíclic en $\mathcal{R}_n$ és 2^m, on m és el nombre de classes q-ciclotòmiques mòdul n. A més a més, les dimensions dels codis cíclics en $\mathcal{R}_n$ són totes les possibles sumes de les mides de les classes q-ciclotòmiques mòdul n.*

3.2 Zeros d'un codi cíclic

En aquesta secció veurem que amb les arrels del polinomi x^n-1 tenim una caracterització dels codis cíclics en $\mathcal{R}_n$ diferent de la donada pels polinomis generadors.

Amb les notacions de la secció 1.3, tenim que $\mathbb{F}_{q^t}$, on $t = ord_n(q)$, és el cos de descomposició de x^n-1. Per tant, $\mathbb{F}_{q^t}$ conté una arrel primitiva n-èsima de la unitat α, amb la descomposició en factors lineals $x^n-1 = \prod_{i=0}^{n-1}(x-\alpha^i)$ sobre $\mathbb{F}_{q^t}$ i la descomposició en factors irreductibles $x^n-1 = \prod_s M_{\alpha^s}(x)$ sobre $\mathbb{F}_q$, on s recorre un conjunt de representants de les classes q-ciclotòmiques mòdul n.

Sigui $\mathcal{C}$ un codi cíclic de $\mathcal{R}_n$ amb polinomi generador $g(x)$. Pel Teorema 1.13 i el Teorema 3.3, tenim que $g(x) = \prod_s M_{\alpha^s}(x) = \prod_s \prod_{i\in C_s}(x-\alpha^i)$, on s recorre un subconjunt de representants de les classes q-ciclotòmiques mòdul n. Sigui $T = \bigcup_s C_s$ la unió d'aquestes classes ciclotòmiques, llavors definim les arrels de la unitat $\mathcal{Z} = \{\alpha^i \mid i \in T\}$ com els *zeros* del codi cíclic $\mathcal{C}$. També definim T com el *conjunt de definició* de $\mathcal{C}$, notem que T depèn de l'arrel primitiva n-èsima de la unitat α que escollim.

Se segueix del Teorema 3.3 que $c(x)$ pertany a $\mathcal{C}$ si i només si $c(\alpha^i) = 0$ per a tot $i \in T$. Notem també que tant el conjunt de definició com el conjunt de zeros determinem el polinomi generador $g(x)$. Pel mateix teorema, veiem que la dimensió de $\mathcal{C}$ és $n - |T|$, ja que $g(x)$ és de grau $|T|$.

En conseqüència, hem demostrat el següent resultat:

Teorema 3.6. *Sigui α una arrel primitiva n-èsima de la unitat en algun cos extensió de $\mathbb{F}_q$. Sigui $\mathcal{C}$ un codi cíclic de longitud n sobre $\mathbb{F}_q$ amb conjunt de definició T i polinomi generador $g(x)$. Llavors:*

(i) T és una unió de classes q-ciclotòmiques mòdul n.

(ii) $g(x) = \prod_{i\in T}(x-\alpha^i)$.

(iii) $c(x) \in \mathcal{R}_n$ és una paraula del codi $\mathcal{C}$ si i només si $c(\alpha^i) = 0$ per a tot $i \in T$.

(iv) $\mathcal{C}$ és un codi cíclic de dimensió $n - |T|$.

3.3 Codificació

Hi ha dues maneres de codificar un missatge amb un codi cíclic, el mètode sistemàtic i el mètode no sistemàtic. Diem que una codificació és *sistemàtica* si la paraula codificada és una extensió del missatge original, és a dir, en un codi lineal de longitud n i dimensió k, si existeixen coordenades $i_1, i_2, \ldots, i_k$ de manera que el missatge es troba en aquestes k posicions de la paraula codificada.

Considerem un codi cíclic $\mathcal{C}$ de longitud n sobre $\mathbb{F}_q$ amb polinomi generador $g(x)$ de grau $n-k$.

El mètode no sistemàtic se segueix de la codificació que s'ha descrit al capítol de codis lineals. Sigui G la matriu generadora de $\mathcal{C}$, llavors codifiquem el missatge $\mathbf{m} \in \mathbb{F}_q^k$ com $\mathbf{c} = \mathbf{m}G$. Utilitzant el Teorema 3.3, si agafem $m(x)$ i $c(x)$, els polinomis de $\mathbb{F}_q[x]$ corresponents a $\mathbf{m}$ i $\mathbf{c}$ respectivament, llavors $c(x) = m(x)g(x)$.

Veiem ara el mètode sistemàtic. Sigui $m(x)$ el polinomi associat al missatge $\mathbf{m}$, tenim que aquest polinomi té grau màxim $k-1$. Per tant, el polinomi $x^{n-k}m(x)$ té grau màxim $n-1$ amb el $n-k$ primers coeficients iguals a 0. Llavors, el missatge està contingut en els coeficients de $x^{n-k}, x^{n-k+1}, \ldots, x^{n-1}$. Si el dividim per $g(x)$, tenim que

$$x^{n-k}m(x) = q(x)g(x) + r(x), \text{on } \mathrm{gr}(r(x)) < n-k \text{ o } r(x) = 0.$$

Si agafem $c(x) = x^{n-k}m(x) - r(x)$, com que és múltiple de $g(x)$, tenim que $c(x) \in \mathcal{C}$. Veiem com $c(x)$ difereix de $x^{n-k}m(x)$ només en els coeficients $1, x, \ldots, x^{n-k-1}$ perquè tenim $\mathrm{gr}(r(x)) < n-k$. Per tant, $c(x)$ conté el missatge $\mathbf{m}$ en els coeficients dels termes de grau major o igual a $n-k$.

4 Codis BCH

En aquest capítol introduirem els codis BCH, que és una de les famílies de codis cíclics més importants. Aquests codis són descoberts el 1959 per Alexis Hocquenghem [Hoc59] i, independentment, el 1960 per Raj Chandra Bose i D.K. Ray-Chaudhuri [BRC60]. El nom BCH ve de les inicials dels seus descobridors, Bose-Chaudhuri-Hocquenghem. Aquests codis són utilitzats en moltes aplicacions, com ara comunicació amb sondes espacials[CP88], unitats d'estat sòlid[MME12] o codis QR[ISO00].

4.1 Fita BCH

Per a qualsevol codi, és important determinar la seva distància mínima, ja que aquest valor ens dona informació sobre la capacitat de correcció d'errors del codi. Per tant ens interessa trobar fites per a la distància mínima dels codis, sobretot les fites inferiors. En aquesta secció veurem la fita Bose-Chaudhuri-Hocquenghem, coneguda com fita BCH, que ens dona una fita inferior per a la distància mínima dels codis cíclics i que ens serà fonamental per a la construcció dels codis BCH.

Abans de donar la fita, hem de recordar el concepte de matriu de Vandermonde, que serà útil per a la demostració de la fita BCH. Siguin $\alpha_1, \ldots, \alpha_s$ elements d'un cos $\mathbb{F}$, anomenem *matriu de Vandermonde* a la matriu $V = [v_{i,j}]$ de mida $s \times s$ amb $v_{i,j} = \alpha_j^{i-1}$. Notem que en algunes definicions de matriu de Vandermonde, la matriu és la transposada de la qual hem definit nosaltres, però el resultat següent és el mateix.

Lema 4.1. *Sigui V una matriu de Vandermonde de mida $s \times s$, tenim que el seu determinant és* $\det(V) = \prod_{1 \leq i < j \leq s} (\alpha_j - \alpha_i)$. *En particular, V és invertible si els elements $\alpha_1, \ldots, \alpha_s$ són diferents.*

Demostració. Tenim la matriu de Vandermonde

$$V = \begin{bmatrix} 1 & 1 & \ldots & 1 \\ \alpha_1 & \alpha_2 & \ldots & \alpha_s \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_1^{s-1} & \alpha_2^{s-1} & \ldots & \alpha_s^{s-1} \end{bmatrix}.$$

Per propietats del determinant, podem afegir a una fila de la matriu una combinació lineal de les altres files sense canviar el determinant. Per tant, podem restar a cada fila l'anterior multiplicada per α_1, exceptuant la primera. Això ens dona la matriu

$$\begin{bmatrix} 1 & 1 & \ldots & 1 \\ 0 & \alpha_2 - \alpha_1 & \ldots & \alpha_s - \alpha_1 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \alpha_2^{s-2}(\alpha_2 - \alpha_1) & \ldots & \alpha_s^{s-2}(\alpha_s - \alpha_1) \end{bmatrix}.$$

Ara podem desenvolupar el determinant per la primera columna, i obtenim que

$$\det(V) = \begin{vmatrix} \alpha_2 - \alpha_1 & \ldots & \alpha_s - \alpha_1 \\ \vdots & \ddots & \vdots \\ \alpha_2^{s-2}(\alpha_2 - \alpha_1) & \ldots & \alpha_s^{s-2}(\alpha_s - \alpha_1) \end{vmatrix}.$$

Com que cada columna j de la matriu anterior té el factor $\alpha_j - \alpha_1$, podem treure aquest factor de la columna j i posar-lo com a factor comú de tot el determinant. Això ens dona

$$\det(V) = (\alpha_2 - \alpha_1)\dots(\alpha_s - \alpha_1)\begin{vmatrix} 1 & \dots & 1 \\ \vdots & \ddots & \vdots \\ \alpha_2^{s-2} & \dots & \alpha_s^{s-2} \end{vmatrix} = \prod_{1<j\leq s}(\alpha_j - \alpha_1)\det(V'),$$

on V' és la matriu de Vandermonde de mida $(s-1)\times(s-1)$ amb els elements $\alpha_2,\dots,\alpha_s$. Per tant, iterant aquest procés, obtenim que $\det(V) = \prod_{1\leq i<j\leq s}(\alpha_j - \alpha_i)$.

Per a la segona afirmació, si els elements $\alpha_1,\dots,\alpha_s$ són diferents, llavors $\det(V) \neq 0$ i per tant, V és invertible. □

Per a veure la fita BCH, assumirem que $\mathcal{C}$ és un codi cíclic de longitud n sobre $\mathbb{F}_q$ i que α és una arrel primitiva n-èsima de la unitat en $\mathbb{F}_{q^t}$, on $t = ord_n(q)$. Sigui T el conjunt de definició de $\mathcal{C}$, diem que T conté un conjunt de s elements consecutius si existeix un conjunt S tal que

$$\{b, b+1, \dots, b+s-1\} \bmod n = S \subseteq T.$$

Teorema 4.2 (Fita BCH)**.** *Sigui $\mathcal{C}$ un codi cíclic de longitud n sobre $\mathbb{F}_q$ i sigui T el seu conjunt de definició. Sigui d la distància mínima de $\mathcal{C}$. Si T conté $\delta - 1$ elements consecutius, llavors $d \geq \delta$.*

Demostració. Com que T és el conjunt de definició, tenim que els zeros de $\mathcal{C}$ inclouen $\alpha^b, \alpha^{b+1}, \dots, \alpha^{b+\delta-2}$. Sigui $c(x)$ una paraula no nul·la de $\mathcal{C}$ amb pes w:

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

Suposem el contrari, que $w < \delta$. Com que $c(\alpha^i) = 0$ per a tot $b \leq i \leq b+\delta-2$, ho podem escriure en forma matricial com $M\mathbf{u}^T = \mathbf{0}$, on

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

i $\mathbf{u} = (c_{i_1}, c_{i_2}, \dots, c_{i_w})$. Com que $\mathbf{u} \neq \mathbf{0}$, tenim que $\det(M) = 0$. Però $\det(M) = \alpha^{(i_1+i_2+\dots+i_w)b}\det(V)$, on V és la matriu de Vandermonde

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

Com que els elements $\alpha^{i_1}, \alpha^{i_2}, \dots, \alpha^{i_w}$ són diferents, pel lema 4.1 tenim que $\det(V) \neq 0$, llavors $\det(M) \neq 0$, que és una contradicció. Per tant, necessàriament $w \geq \delta$. □

4.2 Codis BCH

Els codis BCH són els codis cíclics dissenyats aprofitant-se de la fita BCH. Volem construir un codi cíclic $\mathcal{C}$ de longitud n sobre $\mathbb{F}_q$ que simultàniament tingui una distància

mínima gran i dimensió gran. Per al requisit de distància mínima gran, per la fita BCH, simplement hem d'escollir un conjunt de definició T per a $\mathcal{C}$ que contingui molts elements consecutius. Respecte al segon requisit, com que la dimensió de $\mathcal{C}$ és $n - |T|$, pel Teorema 3.6, volem que $|T|$ sigui el més petit possible.

Per tant, si volem que el codi $\mathcal{C}$ tingui una distància mínima d'almenys δ, hem d'escollir el conjunt de definició més petit possible que contingui $\delta - 1$ elements consecutius. Això ens dona la següent definició.

Definició 4.3. *Sigui δ un enter $2 \leq \delta \leq n$. Un codi BCH sobre $\mathbb{F}_q$ de longitud n i distància designada δ és un codi cíclic amb conjunt de definició*

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

on C_i és la classe q-ciclotòmica mòdul n que conté i.

Teorema 4.4. *Un codi BCH amb distància designada δ té una distància mínima d'almenys δ.*

Demostració. Per definició, el conjunt de definició conté $\delta - 1$ elements consecutius, per tant, el resultat segueix directament de la fita BCH. □

Variant el valor de b, podem obtenir una varietat de codis amb, possiblement, diferents distàncies mínimes i dimensions. Quan $b = 1$, diem que és un codi BCH en *sentit estricte*.

4.3 Descodificació

En aquesta secció presentarem tres algoritmes per a descodificar els codis BCH. Totes tres tècniques consisteixen en un procediment de quatre passos on només varia el segon, que també és el més complicat.

4.3.1 Algoritme Peterson-Gorenstein-Zierler

Aquest algoritme és desenvolupat per als codis BCH binaris per Peterson el 1960 i generalitzat un any més tard per als codis BCH no binaris per Gorenstein i Zierler.

Sigui $\mathcal{C}$ un codi BCH de longitud n sobre $\mathbb{F}_q$ amb distància designada δ. Com que la distància mínima de $\mathcal{C}$ és d'almenys δ, $\mathcal{C}$ pot corregir fins a $t = \lfloor(\delta - 1)/2\rfloor$ errors. L'algoritme de descodificació Peterson-Gorenstein-Zierler pot corregir fins a t errors.

Encara que l'algoritme funciona per a qualsevol codi BCH, per simplicitat, assumirem que $\mathcal{C}$ és en sentit estricte. Per tant, el conjunt de definició T de $\mathcal{C}$ conté $\{1, 2, \ldots, \delta - 1\}$, on utilitzem α com a arrel primitiva n-èsima de la unitat en el cos extensió $\mathbb{F}_{q^m}$ de $\mathbb{F}_q$, amb $m = ord_n(q)$, per determinar aquest conjunt.

Suposem que rebem $y(x)$, on assumim que difereix de la paraula enviada $c(x)$ en t o menys posicions. Llavors $y(x) = c(x) + e(x)$, on $c(x) \in \mathcal{C}$ i $e(x)$ és el vector error amb pes $\nu \leq t$. Si suposem que les coordenades on es produeixen els errors són $k_1, k_2, \ldots, k_\nu$, llavors podem escriure el vector error com

$$e(x) = e_{k_1} x^{k_1} + e_{k_2} x^{k_2} + \cdots + e_{k_\nu} x^{k_\nu}. \tag{4.1}$$

Un cop s'ha determinat $e(x)$, que consisteix a trobar les localitzacions dels errors k_j i les magnituds dels errors e_{k_j}, podem descodificar el vector rebut com $c(x) = y(x) - e(x)$.

Recordem que pel Teorema 3.6, $c(x) \in \mathcal{C}$ si i només si $c(\alpha^i) = 0$ per a tot $i \in T$. En particular, $y(\alpha^i) = c(\alpha^i) + e(\alpha^i) = e(\alpha^i)$ per a tot $1 \leq i \leq 2t$, ja que $2t \leq \delta - 1$. Per a $1 \leq i \leq 2t$, definim la *síndrome* S_i de $y(x)$ com l'element $S_i = y(\alpha^i)$ en $\mathbb{F}_{q^m}$.

El primer pas d'aquest algoritme consisteix a computar les síndromes $S_i = y(\alpha^i)$ per a $1 \leq i \leq 2t$ del vector rebut. Per a realitzar aquest pas ens pot ajudar la següent proposició.

Proposició 4.5. *$S_{iq} = S_i^q$ per a tot $i \geq 1$ amb $iq \leq 2t$.*

Demostració. Tenim que $S_i^q = \left(\sum_{j=1}^{\nu} e_{k_j}(\alpha^i)^{k_j}\right)^q = \sum_{j=1}^{\nu} e_{k_j}^q \left((\alpha^i)^{k_j}\right)^q$, on $e_{k_j} \in \mathbb{F}_q$, llavors $e_{k_j}^q = e_{k_j}^{q-1} e_{k_j} = e_{k_j}$, i per tant, $S_i^q = \sum_{j=1}^{\nu} e_{k_j}(\alpha^{iq})^{k_j} = S_{iq}$. □

D'aquestes síndromes volem obtenir un sistema d'equacions amb incògnites les localitzacions i les magnituds dels errors. Veiem que a partir de (4.1) que les síndromes satisfan

$$S_i = y(\alpha^i) = \sum_{j=1}^{\nu} e_{k_j}(\alpha^i)^{k_j} = \sum_{j=1}^{\nu} e_{k_j}(\alpha^{k_j})^i, \tag{4.2}$$

per a $1 \leq i \leq 2t$. Per simplificar la notació, per a $1 \leq j \leq \nu$, denotem $E_j = e_{k_j}$ com la magnitud de l'error en la coordenada k_j i $X_j = \alpha^{k_j}$ com el *nombre de localització de l'error* corresponent a k_j. Per la proposició 1.5, si $\alpha^i = \alpha^k$ per a i i k entre 0 i $n-1$, llavors $i = k$. Per tant, el valor de X_j determina unívocament la localització de l'error k_j. Fent servir aquesta notació, podem escriure (4.2) com

$$S_i = \sum_{j=1}^{\nu} E_j X_j^i, \text{ per a } 1 \leq i \leq 2t, \tag{4.3}$$

que ens porta al següent sistema d'equacions:

$$\begin{cases} S_1 = E_1X_1 + E_2X_2 + \cdots + E_\nu X_\nu, \\ S_2 = E_1X_1^2 + E_2X_2^2 + \cdots + E_\nu X_\nu^2, \\ S_3 = E_1X_1^3 + E_2X_2^3 + \cdots + E_\nu X_\nu^3, \\ \qquad\qquad \vdots \\ S_{2t} = E_1X_1^{2t} + E_2X_2^{2t} + \cdots + E_\nu X_\nu^{2t}. \end{cases} \tag{4.4}$$

Aquest sistema és no lineal en les X_j amb coeficients desconegudes E_j. L'estratègia que es fa servir és utilitzar (4.3) per a crear un nou sistema lineal, sobre unes noves variables $\sigma_1, \sigma_2, \ldots, \sigma_\nu$, que ens permeti trobar directament els nombres de localització d'error. Un cop aquests valors són coneguts, tornem a (4.4), on llavors ja és un sistema lineal en les E_j que podem resoldre per a trobar les magnituds dels errors.

Per aquest motiu, es defineix el *polinomi localitzador d'error* com

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

Per la primera igualtat, veiem que les arrels de $\sigma(x)$ són les inverses dels nombres de localització d'error i, per tant,

$$\sigma(X_j^{-1}) = 1 + \sigma_1 X_j^{-1} + \sigma_2 X_j^{-2} + \cdots + \sigma_\nu X_j^{-\nu} = 0,$$

per a $1 \leq j \leq \nu$. Si multipliquem per $E_j X_j^{i+\nu}$ obtenim que

$$E_j X_j^{i+\nu} + \sigma_1 E_j X_j^{i+\nu-1} + \cdots + \sigma_\nu E_j X_j^i = 0,$$

per a tot i. Ara sumem aquestes equacions per a $1 \leq j \leq \nu$ i obtenim que

$$\sum_{j=1}^{\nu} E_j X_j^{i+\nu} + \sigma_1 \sum_{j=1}^{\nu} E_j X_j^{i+\nu-1} + \cdots + \sigma_\nu \sum_{j=1}^{\nu} E_j X_j^i = 0.$$

Veiem que mentre $1 \leq i$ i $i + \nu \leq 2t$, les sumes són exactament les síndromes que hem definit a (4.3). Per tant, ho podem reescriure com

$$S_{i+\nu} + \sigma_1 S_{i+\nu-1} + \sigma_1 S_{i+\nu-2} + \cdots + \sigma_\nu S_i = 0$$

o, equivalentment,

$$\sigma_1 S_{i+\nu-1} + \sigma_1 S_{i+\nu-2} + \cdots + \sigma_\nu S_i = -S_{i+\nu},$$

vàlida per a $1 \leq i \leq \nu$, ja que sabem que $\nu \leq t$. Això ens dona un sistema lineal de ν equacions en les σ_k que podem escriure en forma matricial com

$$\begin{bmatrix} S_1 & S_2 & S_3 & \dots & S_{\nu-1} & S_\nu \\ S_2 & S_3 & S_4 & \dots & S_\nu & S_{\nu+1} \\ S_3 & S_4 & S_5 & \dots & S_{\nu+1} & S_{\nu+2} \\ & & & \vdots & & \\ S_\nu & S_{\nu+1} & S_{\nu+2} & \dots & S_{2\nu-2} & S_{2\nu-1} \end{bmatrix} \begin{bmatrix} \sigma_\nu \\ \sigma_{\nu-1} \\ \sigma_{\nu-2} \\ \vdots \\ \sigma_1 \end{bmatrix} = \begin{bmatrix} -S_{\nu+1} \\ -S_{\nu+2} \\ -S_{\nu+3} \\ \vdots \\ -S_{2\nu} \end{bmatrix}. \tag{4.5}$$

Resoldre aquest sistema per a trobar els valors de $\sigma_1, \dots, \sigma_\nu$ és el segon pas de l'algoritme.

Una dificultat que tenim és que no sabem el valor de ν, ja que no coneixem el nombre d'errors que hi ha en el vector rebut, i per tant no sabem la mida del sistema (4.5). Recordem de la secció 2.3 que la nostra solució té el menor valor de ν possible, i per a això, farem ús del següent lema.

Lema 4.6. *Sigui $\mu \leq t$ i*

$$M_\mu = \begin{bmatrix} S_1 & S_2 & \dots & S_\mu \\ S_2 & S_3 & \dots & S_{\mu+1} \\ S_3 & S_4 & \dots & S_{\mu+2} \\ & & \vdots & \\ S_\mu & S_{\mu+1} & \dots & S_{2\mu-1} \end{bmatrix},$$

llavors M_μ és invertible si $\mu = \nu$ i singular si $\mu > \nu$, on ν és el nombre d'errors en el vector rebut.

Demostració. Si $\mu > \nu$, tenim que $X_{\nu+1} = X_{\nu+2} = \cdots = X_\mu = 0$ i $E_{\nu+1} = E_{\nu+2} = \cdots = E_\mu = 0$, i per tant $S_i = \sum_{j=1}^{\mu} E_j X_j^i$, per a $1 \leq i \leq 2t$. Llavors si agafem les matrius

$$A_\mu = \begin{bmatrix} 1 & 1 & \dots & 1 \\ X_1 & X_2 & \dots & X_\mu \\ & & \vdots & \\ X_1^{\mu-1} & X_2^{\mu-1} & \dots & X_\mu^{\mu-1} \end{bmatrix} \text{ i } B_\mu = \begin{bmatrix} E_1 X_1 & 0 & \dots & 0 \\ 0 & E_2 X_2 & \dots & 0 \\ & & \vdots & \\ 0 & 0 & \dots & E_\mu X_\mu \end{bmatrix},$$

tenim que $M_\mu = A_\mu B_\mu A_\mu^T$, per càlcul directe. Aplicant el determinant a ambdues bandes, tenim que $\det(M_\mu) = \det(A_\mu)\det(B_\mu)\det(A_\mu)$.

Si $\mu > \nu$, $\det(B_\mu) = 0$, ja que B_μ és una matriu diagonal amb zeros a la diagonal, i per tant $\det(M_\mu) = 0$ i M_μ és singular.

Si $\mu = \nu$, $\det(B_\mu) \neq 0$, ja que B_μ no té cap zero a la diagonal. Observem que A_μ és una matriu de Vandermonde i pel lema 4.1 tenim que $\det(A_\mu) \neq 0$, ja que els elements $X_1, \ldots, X_\mu$ són diferents. Per tant, $\det(M_\mu) \neq 0$ i M_μ és invertible. □

Per a executar aquest segon pas, intentem esbrinar el nombre d'erros ν. Per a això, comencem amb l'estimació $\mu = t$ i anem decreixent μ fins que trobem una matriu M_μ invertible, aprofitant la notació del lema. Un cop trobada, sabem que $\nu = \mu$ i podem resoldre el sistema (4.5) per a determinar el polinomi localitzador d'error $\sigma(x)$.

El tercer pas de l'algoritme consisteix a trobar les arrels de $\sigma(x)$ i després invertir-les per a trobar els nombres de localització d'error. Això normalment es fa amb una cerca exhaustiva, provant $\sigma(\alpha^i)$ per a tot $1 \leq i \leq n$.

L'últim pas és resoldre el sistema lineal (4.4) un cop substituïts els valors de X_j i obtenim les magnituds dels errors E_j. De fet, només hem de considerar les ν primeres equacions de (4.4), ja que

$$\det \begin{bmatrix} X_1 & X_2 & \ldots & X_\nu \\ X_1^2 & X_2^2 & \ldots & X_\nu^2 \\ & & \vdots & \\ X_1^\nu & X_2^\nu & \ldots & X_\nu^\nu \end{bmatrix} = X_1 X_2 \ldots X_\nu \det \begin{bmatrix} 1 & 1 & \ldots & 1 \\ X_1 & X_2 & \ldots & X_\nu \\ & & \vdots & \\ X_1^{\nu-1} & X_2^{\nu-1} & \ldots & X_\nu^{\nu-1} \end{bmatrix},$$

on la matriu de la dreta és una matriu de Vandermonde i, per tant, el seu determinant és diferent de zero per ser les X_j diferents.

Per tant, l'algoritme Peterson-Gorenstein-Zierler per a la descodificació de codis BCH consisteix en els següents passos:

I. Calcular les síndromes $S_i = y(\alpha^i)$ per a $1 \leq i \leq 2t$.

II. Comencem amb $\mu = t$ i fem decréixer μ fins el primer valor per al qual M_μ és invertible. Tenim llavors que $\nu = \mu$ i resolem (4.5) per a determinar $\sigma(x)$.

III. Trobar les arrels de $\sigma(x)$ amb el càlcul $\sigma(\alpha^i)$ per a $1 \leq i \leq n$. Invertim aquestes arrels per a obtenir els nombres de localització d'error X_j.

IV. Resoldre les ν primeres equacions de (4.4) per a trobar les magnituds dels errors E_j.

Veiem ara algunes observacions sobre l'algoritme.

- Un cop s'han trobat els errors i el vector rebut és corregit, s'ha de comprovar que aquest vector resultant és del codi. Si no ho és, llavors el vector rebut conté més errors dels que podem corregir i l'algoritme no pot corregir el vector rebut.
- Si el codi BCH és binari, llavors totes les magnituds dels errors són 1. Per tant, ens podem saltar el quart pas de l'algoritme.
- Si totes les síndromes són zero, llavors el vector rebut és del codi i considerem que no s'ha produït cap error.

Analitzem ara l'eficiència d'aquest algoritme pas a pas:

I. En aquest pas avaluem el polinomi $y(x)$ de grau màxim n en $2t$ punts. Per tant, el cost per a cada punt és de n multiplicacions i n sumes, utilitzant, per exemple, el mètode de Horner[Pan66]. Si considerem que el cost d'aquestes operacions és constant en el cos finit, el cost total d'aquest pas és de $2t \cdot 2n$, que podem escriure com $O(tn)$.

II. En aquest punt, el pitjor cas és quan $\nu = 1$ o quan $\nu > t$, on no es podria corregir l'error. En tots dos casos, es fan t iteracions, on cada iteració consisteix a calcular el determinant d'una matriu de mida $\mu \times \mu$, on μ decreix en cada iteració. La complexitat de calcular el determinant d'una matriu depèn de l'algoritme que fem servir, si fem servir el mètode de reducció de Gauss, el cost és de $O(\mu^3)$[Far18]. Per tant, el cost de les iteracions és de $O(t^4)$. Un cop fet això, hem de resoldre un sistema lineal de mida $\nu \times \nu$, però com que ja tenim la matriu triangular superior, del mètode de Gauss, el cost d'aquesta operació és de $O(\nu^2)$, en el pitjor cas, $O(t^2)$. Per tant, el cost total d'aquest pas és de $O(t^4)$.

III. Aquí hem de trobar les arrels de $\sigma(x)$, que és un polinomi de grau ν. Com que l'algoritme per a trobar aquestes arrels ja s'ha definit, la cerca exhaustiva, tenim un cas similar a I, s'ha d'avaluar un polinomi de grau màxim t en n punts, amb un cost de $O(tn)$. Recordem que invertir les arrels és una operació de cost constant en el cos finit. Per tant, el cost total d'aquest pas és de $O(tn)$.

IV. Finalment, en aquest pas, hem de resoldre un sistema lineal de mida $\nu \times \nu$, on ν. Seguint el mateix raonament que en el pas II, el cost d'aquesta operació és de $O(\nu^3)$, que en el pitjor cas podem escriure com $O(t^3)$.

Per tant, la complexitat total de l'algoritme Peterson-Gorenstein-Zierler és de $O(tn + t^4)$.

Com podem observar, per a la variable t, el pas més complicat és el segon, ja que quan la capacitat de correcció d'error del codi és molt gran, les matrius que hem de considerar també ho són. Per a això, en les següents seccions veurem altres mètodes per a realitzar aquest pas de manera més eficient.

4.3.2 Algoritme Berlekamp-Massey

L'algoritme de Berlekamp-Massey utilitza una tècnica iterativa per a trobar el polinomi localitzador d'error $\sigma(x)$, que és el segon pas de l'algoritme Peterson-Gorenstein-Zierler. Encara que aquest algoritme també funciona per a qualsevol codi BCH, per simplicitat, assumirem que estem treballant en codis binaris. Farem servir la mateixa notació que en la secció anterior.

Aquest algoritme, en comptes de fer servir les equacions definides a (4.5), utilitza les identitats o fórmules de Newton, enunciades i demostrades a [Tra20]. Aquí només les veurem, enunciat i demostració, aplicades en aquest context [Cha98].

Teorema 4.7. *Sigui $\sigma(x)$ el polinomi localitzador d'error, $X_1, X_2, \ldots, X_\nu$ els nombres de localització d'error, si totes les magnituds d'error són 1, llavors els coeficients de $\sigma(x)$ i*

les síndromes S_i estan relacionades per les identitats de Newton:

$$\begin{aligned}
&S_1 + \sigma_1 = 0,\\
&S_2 + \sigma_1 S_1 + 2\sigma_2 = 0,\\
&S_3 + \sigma_1 S_2 + \sigma_2 S_1 + 3\sigma_3 = 0,\\
&\qquad\vdots\\
&S_\nu + \sigma_1 S_{\nu-1} + \cdots + \sigma_{\nu-1} S_1 + \nu\sigma_\nu = 0,
\end{aligned}$$

i per a $j \geq \nu$:

$$S_j + \sigma_1 S_{j-1} + \cdots + \sigma_\nu S_{j-\nu} = 0.$$

Demostració. Recordem de (4.3) que $S_i = \sum_{j=1}^{\nu} E_j X_j^i$, per a $1 \leq i \leq 2t$, si tenim que les magnituds d'error E_j són totes 1, llavors $S_i = \sum_{j=1}^{\nu} X_j^i$.

Veiem primer la derivada de $\sigma(x) = \prod_{i=1}^{\nu}(1 - xX_i)$, la fórmula de derivació d'un producte ens diu que el derivat del producte $\prod_{i=1}^{\nu}(1-xX_i)$ és la suma, sobre l'índex i, del derivat del factor $1 - xX_i$, que és $-X_i$, multiplicat pel producte dels altres factors, que és $\sigma(x)/(1 - xX_i)$. Per tant, tenim que

$$\frac{\sigma'(x)}{\sigma(x)} = \sum_{i=1}^{\nu} \frac{-X_i}{1 - xX_i}.$$

Si agafem el denominador com a sèrie formal de potències, obtenim $(1 - xX_i)^{-1} = \sum_{l=0}^{\infty} x^l X_i^l$. Llavors, podem escriure

$$\frac{\sigma'(x)}{\sigma(x)} = \sum_{i=1}^{\nu} -X_i \sum_{l=0}^{\infty} x^l X_i^l = \sum_{l=0}^{\infty} \left(\sum_{i=1}^{\nu} -X_i^{l+1} \right) x^l = \sum_{l=0}^{\infty} -S_{l+1} x^l.$$

Si ara escrivim el polinomi localitzador d'erros com $\sigma(x) = \sum_{i=0}^{\nu} \sigma_i x^i$, per tant, amb $\sigma_0 = 1$, passant multiplicant, tenim que

$$\sigma'(x) = \left(\sum_{l=0}^{\infty} -S_{l+1} x^l \right) \left(\sum_{i=0}^{\nu} \sigma_i x^i \right).$$

Si desenvolupem el producte de la dreta, obtenim que

$$\sigma'(x) = \sum_{j=0}^{\infty} \left(\sum_{l+i=j} -S_{l+1}\sigma_i \right) x^j.$$

A continuació, si agafem la derivada de $\sigma(x)$ vista com $\sigma(x) = 1 + \sum_{i=1}^{\nu} \sigma_i x^i$, obtenim que

$$\sum_{j=1}^{\nu} j\sigma_j x^{j-1} = \sum_{j=0}^{\infty} \left(\sum_{l+i=j} -S_{l+1}\sigma_i \right) x^j.$$

Finalment, igualant coeficients, obtenim que

$$j\sigma_j = -\sum_{i=0}^{j-1} S_{j-1}\sigma_i, \text{ per a } \ j \leq \nu,$$

$$0 = \sum_{i=0}^{\nu} S_{j-1}\sigma_i, \text{ per a } j > \nu,$$

que és el resultat que volíem demostrar un cop hem substituït $\sigma_0 = 1$. □

Notem que estem treballant en un de cos de característica 2, per tant, $i\sigma_i = 0$ quan i és parell i $i\sigma_i = \sigma_i$ quan i és senar. Llavors veiem que la segona identitat és $S_2+\sigma_1 S_1+2\sigma_2 = S_2+\sigma_1 S_1 = 0$, on no hi ha cap σ_2 i, per tant, no ens aporta nova informació. En particular, si apliquem la proposició 4.5, tenim que $S_2 + \sigma_1 S_1 = S_1^2 + \sigma_1 S_1 = 0$ que és equivalent a $S_1 + \sigma_1 = 0$, la primera identitat. Això ens permet reduir el nombre d'identitats de Newton que hem de considerar a la meitat, que enumerem i reescribim:

(1) $S_1 + \sigma_1 = 0$

(2) $S_3 + \sigma_1 S_2 + \sigma_2 S_1 + \sigma_3 = 0$

(3) $S_5 + \sigma_1 S_4 + \sigma_2 S_3 + \sigma_3 S_2 + \sigma_4 S_1 + \sigma_5 = 0$

$\vdots$

(μ) $S_{2\mu-1} + \sigma_1 S_{2\mu-2} + \sigma_2 S_{2\mu-3} + \cdots + \sigma_{2\mu-2} S_1 + \sigma_{2\mu-1} = 0$

Definim una seqüència de polinomis $\sigma^{(\mu)}(x)$ de grau d_μ com:

$$\sigma^{(\mu)}(x) = 1 + \sigma_1^{(\mu)} x + \sigma_2^{(\mu)} x^2 + \cdots + \sigma_{d_\mu}^{(\mu)} x^{d_\mu}.$$

Els polinomis $\sigma^{(\mu)}(x)$ es calculen com el polinomi de grau mínim tal que els seus coeficients, $\sigma_1^{(\mu)}, \sigma_2^{(\mu)}, \ldots, \sigma_{d_\mu}^{(\mu)}$, satisfan les μ primeres identitats de Newton. Associem a cada polinomi la seva *discrepància* Δ_μ, que mesura com de lluny està $\sigma^{(\mu)}(x)$ de satisfer la següent identitat:

($\mu+1$) $\Delta_\mu = S_{2\mu+1} + \sigma_1^{(\mu)} S_{2\mu} + \sigma_2^{(\mu)} S_{2\mu-1} + \cdots + \sigma_{2\mu}^{(\mu)} S_1 + \sigma_{2\mu+1}^{(\mu)}$.

Comencem amb els polinomis $\sigma^{(-1/2)}(x) = 1$ i $\sigma^{(0)}(x) = 1$ i llavors generem els polinomis $\sigma^{(\mu)}(x)$ inductivament depenent de la discrepància. Per convenció, definim $\Delta_{-1/2} = 1$. Llavors substituïm els coeficients de $\sigma^{(0)}(x)$ en la identitat (1) i obtenim que la discrepància de $\sigma^{(0)}(x)$ és $\Delta_0 = S_1$, ja que $\sigma_1^{(0)} = 0$.

Procedim de la següent manera per a generar els polinomis que continuen: Notant la discrepància $\Delta_0 = S_1$, si $\sigma^{(0)}(x)$ tingués un terme addicional $S_1 x$, els coeficients d'aquest polinomi $\sigma^{(0)}(x) + S_1 x = 1 + S_1 x$ satisfaria la identitat (1) ja que $S_1 + S_1 = 0$. Per tant, agafem $\sigma^{(1)}(x) = 1 + S_1 x$. Substituint els coeficients de $\sigma^{(1)}(x)$ en la identitat (2), obtenim que la discrepància de $\sigma^{(1)}(x)$ és $\Delta_1 = S_3 + \sigma_1^{(1)} S_2 = S_3 + S_1 S_2$. Llavors tenim les següents possibilitats:

- Si $\Delta_1 = 0$, $\sigma^{(1)}(x)$ també satisfà (2).
- Si $\Delta_1 \neq 0$ i $S_1 \neq 0$, llavors $\sigma^{(2)}(x) = \sigma^{(1)}(x)+(S_3+S_1 S_2)S_1^{-1}x^2 = \sigma^{(1)}(x)+\Delta_1\Delta_0^{-1}x^2$ satisfà les igualtats (1) i (2).
- Si $\Delta_1 \neq 0$ i $S_1 = 0$, llavors $\sigma^{(1)}(x) = 1$ i $\Delta_1 = S_3$, per tant, el polinomi de grau més petit que satisfà (1) i (2) és $\sigma^{(2)}(x) = \sigma^{(1)}(x) + S_3 x^3 = \sigma^{(1)}(x) + \Delta_1 x^3$.

Aquesta és la idea bàsica d'aquest algoritme, un pot deduir que les possibilitats es tornen més complicades a mesura que augmenta μ, però, afortunadament, aquest algoritme redueix cada iteració en una o dues opcions. Veiem ara el pas iteratiu en el cas general demostrat a [Ber15].

L'algoritme de Berlekamp-Massey calcula el polinomi localitzador d'error per a codis BCH binaris amb un procés iteratiu que comença amb $\mu = 0$ i acaba quan es troba $\sigma^{(t)}(x)$:

I. Si $\Delta_\mu = 0$, llavors
$$\sigma^{(\mu+1)}(x) = \sigma^{(\mu)}(x).$$

II. Si $\Delta_\mu \neq 0$, trobem el valor $-(1/2) \leq \rho < \mu$ tal que $\Delta_\rho \neq 0$ i $2\rho - d_\rho$ sigui el més gran possible. Llavors:
$$\sigma^{(\mu+1)}(x) = \sigma^{(\mu)}(x) + \Delta_\mu \Delta_\rho^{-1} x^{2(\mu-\rho)} \sigma^{(\rho)}(x).$$

On el polinomi localitzador d'error és $\sigma(x) = \sigma^{(t)}(x)$. Si el grau d'aquest polinomi és més gran que t, llavors el vector rebut conté més errors dels que podem corregir i l'algoritme no pot corregir el vector rebut. Un cop s'ha determinat $\sigma(x)$, continuem com s'ha descrit a l'algoritme Peterson-Gorenstein-Zierler.

Analitzem ara l'eficiència d'aquest algoritme. Veiem primer que el nombre d'iteracions que s'ha de realitzar és t. Dins de cada iteració, el pitjor cas és quan tenim $\Delta_\mu \neq 0$. Per a trobar el polinomi $\sigma^{(\mu+1)}(x)$, primer fem un producte per a obtenir $\Delta_\mu \Delta_\rho^{-1}$ i llavors un producte d'un polinomi d'un coeficient amb un polinomi de grau màxim t, $\Delta_\mu \Delta_\rho^{-1} x^{2(\mu-\rho)} \sigma^{(\rho)}(x)$, que correspon a t operacions. Finalment, sumem dos polinomis de grau màxim t, altre cop t operacions. Per tant, el cost d'una iteració és de $O(t)$ i el cost total de l'algoritme és de $O(t^2)$. Notem que falten els costos dels altres tres passos que s'ha definit a l'algoritme Peterson-Gorenstein-Zierler, que són $O(tn)$, $O(tn)$ i $O(t^3)$. Per tant, la complexitat total de l'algoritme de Berlekamp-Massey és de $O(tn + t^3)$.

4.3.3 Algoritme Sugiyama

L'algoritme de Sugiyama és un altre mètode per a trobar el polinomi localitzador d'error, per tant, una altra forma de realitzar el pas dos de l'algoritme Peterson-Gorenstein-Zierler.

Recordem que definim el polinomi localitzador d'error com $\sigma(x) = \prod_{j=1}^{\nu}(1 - xX_j)$. Definim el *polinomi avaluador d'error* com

$$\omega(x) = \sum_{j=1}^{\nu} E_j X_j \prod_{\substack{i=1 \\ i\neq j}}^{\nu} (1 - xX_j) = \sum_{j=1}^{\nu} E_j X_j \frac{\sigma(x)}{1 - xX_j}. \tag{4.6}$$

Notem que $\mathrm{gr}(\sigma(x)) = \nu$ i, per tant, $\mathrm{gr}(\omega(x)) \leq \nu - 1$. A partir de les síndromes S_i, definim el polinomi
$$S(x) = \sum_{i=0}^{2t-1} S_{i+1} x^i,$$
amb $\mathrm{gr}(S(x)) \leq 2t - 1$. Si expandim (4.6) com a sèrie formal de potències i apliquem la definició de síndromes de (4.3) juntament amb la definició de $S(x)$, obtenim que

$$\begin{aligned}
\omega(x) &= \sigma(x) \sum_{j=1}^{\nu} E_j X_j \frac{1}{1 - xX_j} = \sigma(x) \sum_{j=1}^{\nu} E_j X_j \sum_{i=0}^{\infty} (xX_j)^i \\
&= \sigma(x) \sum_{i=0}^{\infty} \left(\sum_{j=1}^{\nu} E_j X_j^{i+1} \right) x^i \equiv \sigma(x) \sum_{i=0}^{2t-1} \left(\sum_{j=1}^{\nu} E_j X_j^{i+1} \right) x^i \pmod{x^{2t}} \\
&\equiv \sigma(x) \sum_{i=0}^{2t-1} S_{i+1} x^i \pmod{x^{2t}} \equiv \sigma(x) S(x) \pmod{x^{2t}}.
\end{aligned}$$

Anomenem aquest resultat com l'*equació clau*
$$\omega(x) \equiv \sigma(x) S(x) \pmod{x^{2t}}.$$

Lema 4.8. *Els polinomis $\sigma(x)$ i $\omega(x)$ són coprimers.*

Demostració. Sabem que les arrels de $\sigma(x)$ són les X_j^{-1} per a $1 \leq j \leq \nu$. Mentre que

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

per tant, $\sigma(x)$ i $\omega(x)$ no tenen cap arrel en comú. □

L'algoritme de Sugiyama utilitza l'algoritme d'Euclides, descrit al Teorema 1.15, per a resoldre l'equació clau de la següent manera:

I. Agafem $f(x) = x^{2t}$ i $s(x) = S(x)$. Assignem $r_{-1}(x) = f(x)$, $r_0(x) = s(x)$, $b_{-1}(x) = 0$ i $b_0(x) = 1$.

II. Trobem els valors de $h_i(x)$, $r_i(x)$ i $b_i(x)$ inductivament per a $i = 1, 2, \ldots, I$ fins que I satisfà $\operatorname{gr}(r_{I-1}(x)) \geq t$ i $\operatorname{gr}(r_I(x)) < t$, utilitzant aquests càlculs:

$$\begin{aligned} r_{i-2}(x) &= r_{i-1} h_i(x) + r_i(x), \quad \text{on } \operatorname{gr}(r_i(x)) < \operatorname{gr}(r_{i-1}(x)), \\ b_i(x) &= b_{i-2}(x) - h_i(x) b_{i-1}(x). \end{aligned}$$

III. $\sigma(x)$ és un múltiple no nul de $b_I(x)$.

Observem que el pas II està ben definit, ja que els $\operatorname{gr}(r_i(x))$ defineixen una successió estrictament decreixent on s'ha començat amb $\operatorname{gr}(r_{-1}(x)) > t$. Observem que per a la descodificació, només ens interessa trobar les arrels de $\sigma(x)$, per tant, és suficient trobar les arrels del polinomi $b_I(x)$ que obtenim al pas II de l'algoritme.

Veiem la complexitat d'aquest algoritme. El pas I consisteix en assignacions de polinomis, per tant, el cost d'aquest pas és constant, notem que $r_0(x) = s(x) = S(x) = \sum_{i=0}^{2t-1} S_{i+1} x^i$, on ja tenim les síndromes calculades.

Els pas II són I iteracions, que en el pitjor cas, si $\operatorname{gr}(r_i(x)) = \operatorname{gr}(r_{i-1}(x)) - 1$ per a cada iteració, tenim que $I = t$. Dins de cada cicle, per a trobar $r_i(x)$ realitzem una divisió entera entre dos polinomis que difereix en grau en 1, per tant, el cost d'aquesta operació és de $O(t)$. A continuació, per a trobar $b_i(x)$, fem el producte d'un polinomi de grau 1 amb un polinomi de grau màxim t, seguit per una suma de dos polinomis de grau màxim t, per tant, el cost d'aquesta operació també és de $O(t)$. Doncs, el cost total d'aquest pas és de $O(t^2)$.

Finalment, el pas III, com ja s'ha comentat, no és rellevant per a la descodificació, per tant, el cost total és de $O(t^2)$. Recordem que falten els costos dels altres tres passos que s'ha definit a l'algoritme Peterson-Gorenstein-Zierler, que són $O(tn)$, $O(tn)$ i $O(t^3)$. En conseqüència, la complexitat total de l'algoritme de Sugiyama és de $O(tn + t^3)$.

Per a demostrar que aquest algoritme funciona, primer veiem el següent lema, que utilitza la notació que s'ha fet servir en l'algoritme.

Lema 4.9. *Siguin $a_{-1}(x) = 1$, $a_0(x) = 0$ i $a_i(x) = a_{i-2}(x) - h_i(x) a_{i-1}(x)$ per a $i \geq 1$, són certes les següents afirmacions:*

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

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

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

(iv) $\text{gr}(b_i(x)) + \text{gr}(r_{i-1}(x)) = \text{gr}(f(x))$ per a $i \geq 0$.

Demostració. Demostrarem totes les afirmacions per inducció.

(i) Els casos $i = -1$ i $i = 0$ se segueixen directament de l'assignació inicial feta al pas I, $1f(x) + 0s(x) = f(x) = r_{-1}(x)$ i $0f(x) + 1s(x) = s(x) = r_0(x)$.
Assumim certes per a $i-1$ i $i-2$, llavors

$$\begin{aligned} a_i(x)f(x) + b_i(x)s(x) &= [a_{i-2}(x) - h_i(x)a_{i-1}(x)]f(x) \\ &\quad + [b_{i-2}(x) - h_i(x)b_{i-1}(x)]s(x) \\ &= a_{i-2}(x)f(x) + b_{i-2}(x)s(x) \\ &\quad - h_i(x)[a_{i-1}(x)f(x) + b_{i-1}(x)s(x)] \\ &= r_{i-2}(x) - h_i(x)r_{i-1}(x) \\ &= r_i(x). \end{aligned}$$

(ii) El cas $i = 0$ se segueix directament de l'assignació inicial feta al pas I, $1f(x) - 0s(x) = f(x) = (-1)^0 f(x)$.
Assumim cert per a $i-1$, llavors

$$\begin{aligned} b_i(x)r_{i-1}(x) - b_{i-1}(x)r_i(x) &= [b_{i-2}(x) - h_i(x)b_{i-1}(x)]r_{i-1}(x) - b_{i-1}(x)r_i(x) \\ &= b_{i-2}(x)r_{i-1}(x) - b_{i-1}(x)[h_i(x)r_{i-1}(x) + r_i(x)] \\ &= b_{i-2}(x)r_{i-1}(x) - b_{i-1}(x)r_{i-2}(x) \\ &= (-1)(-1)^{i-1}f(x) \\ &= (-1)^i f(x). \end{aligned}$$

(iii) El cas $i = 0$ se segueix directament de l'assignació inicial feta al pas I, $0 \cdot 0 - 1 \cdot 1 = -1 = (-1)^{0+1}$.
Assumim cert per a $i-1$, llavors

$$\begin{aligned} a_i(x)b_{i-1}(x) - a_{i-1}(x)b_i(x) &= [a_{i-2}(x) - h_i(x)a_{i-1}(x)]b_{i-1}(x) \\ &\quad - a_{i-1}(x)[b_{i-2}(x) - h_i(x)b_{i-1}(x)] \\ &= a_{i-2}(x)b_{i-1}(x) - a_{i-1}(x)b_{i-2}(x) \\ &= (-1)(-1)^i \\ &= (-1)^{i+1}. \end{aligned}$$

(iv) El cas $i = 0$ se segueix directament de l'assignació inicial feta al pas I, $\text{gr}(1) + \text{gr}(f(x)) = \text{gr}(f(x))$.
Assumim cert per a $i-1$, pel pas II sabem que $\text{gr}(r_i(x)) < \text{gr}(r_{i-1}(x)) < \text{gr}(r_{i-2}(x))$, llavors

$$\begin{aligned} \text{gr}(b_{i-1}(x)r_i(x)) &= \text{gr}(b_{i-1}(x)) + \text{gr}(r_i(x)) \\ &< gr(b_{i-1}(x)) + \text{gr}(r_{i-2}(x)) = \text{gr}(f(x)) \\ &< \text{gr}(f(x)). \end{aligned}$$

Ara només s'ha d'aplicar el punt (ii) per a obtenir que $\text{gr}(b_i(x)) + \text{gr}(r_{i-1}(x)) = \text{gr}(b_i(x)r_{i-1}(x)) = \text{gr}(f(x))$.

□

Veiem ara que l'algoritme funciona.

Demostració. De la part (i) del lema, tenim que

$$a_I(x)x^{2t} + b_I(x)S(x) = r_I(x). \tag{4.7}$$

L'equació clau diu que

$$a(x)x^{2t} + \sigma(x)S(x) = \omega(x), \tag{4.8}$$

per a algun polinomi $a(x)$. Multiplicant (4.7) per $\sigma(x)$ i (4.8) per $b_I(x)$, obtenim que

$$a_I(x)\sigma(x)x^{2t} + b_I(x)\sigma(x)S(x) = r_I(x)\sigma(x), \tag{4.9}$$

$$a(x)b_I(x)x^{2t} + \sigma(x)b_I(x)S(x) = \omega(x)b_I(x). \tag{4.10}$$

Si ho agafem mòdul x^{2t}, això equival a

$$r_I(x)\sigma(x) \equiv \omega(x)b_I(x) \pmod{x^{2t}}. \tag{4.11}$$

Com que $\mathrm{gr}(\sigma(x)) \leq t$, per com s'ha escollit I, tenim que $\mathrm{gr}(r_I(x)\sigma(x)) = \mathrm{gr}(r_I(x)) + \mathrm{gr}(\sigma(x)) < t + t = 2t$. Utilitzant la part (iv) del lema i el fet que $\mathrm{gr}(\omega(x)) < t$, obtenim que $\mathrm{gr}(\omega(x)b_I(x)) = \mathrm{gr}(\omega(x)) + \mathrm{gr}(b_I(x)) < t + \mathrm{gr}(b_I(x)) = t + [\mathrm{gr}(x^{2t}) - \mathrm{gr}(r_{I-1}(x))] \leq t + 2t - t = 2t$. Per tant, (4.11) implica que $r_I(x)\sigma(x) = \omega(x)b_I(x)$. Llavors, amb aquesta igualtat, si tornem a (4.9) i (4.10), tenim que

$$a_I(x)\sigma(x) = a(x)b_I(x). \tag{4.12}$$

Del punt (iii) del lema, obtenim que $a_I(x)$ i $b_I(x)$ són coprimers, llavors ha de passar que $a(x) = \lambda(x)a_I(x)$. Si ho substituïm a (4.12), tenim que

$$\sigma(x) = \lambda(x)b_I(x). \tag{4.13}$$

Per tant, aplicant-ho a (4.8), obtenim que $\lambda(x)a_I(x)x^{2t} + \lambda(x)b_I(x)S(x) = \omega(x)$. Finalment, utilitzant (4.7), tenim que

$$\omega(x) = \lambda(x)r_I(x). \tag{4.14}$$

Aquí observem que pel lema 4.8, els polinomis $\sigma(x)$ i $\omega(x)$ són coprimers, per tant, per les equivalències (4.13) i (4.14), tenim que $\lambda(x)$ ha de ser una constant no nul·la i se satisfà el pas III de l'algoritme. □

Observació 4.10. Cal notar que els tres algoritmes de descodificació que hem vist fins ara, Peterson-Gorenstein-Zierler, Berlekamp-Massey i Sugiyama, poden descodificar qualsevol codi cíclic fins a la fita BCH. És a dir, si $\mathcal{C}$ és un codi cíclic amb conjunt de definició T que conté $\delta - 1$ elements consecutius, aquest codi $\mathcal{C}$ és un codi BCH amb distància designada δ. Per tant, tots aquests algoritmes poden corregir fins a $t_\delta = \lfloor(\delta - 1)/2\rfloor$ errors. Però, com que la fita BCH és una fita inferior de la distància mínima del codi, és possible que la distància mínima d del codi $\mathcal{C}$ sigui $d > \delta$, llavors, depenent del valor de d, el codi $\mathcal{C}$ realment pot corregir més de t_δ errors, ja que $t_d = \lfloor(d - 1)/2\rfloor \geq t_\delta$.

5 Implementació

En aquest capítol veurem la implementació dels codis BCH en el llenguatge de programació C. En específic, veurem la codificació i la descodificació amb els diferents algoritmes que s'ha explicat en la secció anterior. En aquesta implementació s'ha restringit a codis BCH binaris, en sentit estricte i de longitud $n = 2^m - 1$ on $2 \leq m \leq 8$, per tant, de longitud màxima $2^8 - 1 = 255$. Encara aquestes restriccions, el programa ens permetrà observar el comportament dels codis BCH en general.

Aquesta implementació utilitza la llibreria FLINT: Fast Library for Number Theory [FLI23], per tant, és necessari tenir-la instalada tant per a compilar el programa com per a executar-ho. Aquesta llibreria de C, disponible sota llicència GNU LGPL, conté moltes eines per a operar amb nombres, polinomis i matrius sota diferents cossos finits que ens seran molt útils per a la implementació dels codis BCH.

En l'explicació, i en el codi en si, s'ha emprat la notació dels símbols que s'ha anat utilitzant durant el treball, amb especial èmfasi amb la utilitzada en la secció anterior de descodificacions.

Per a compilar el programa:

```
$ gcc main.c -lflint -lm -o bch
```

I executar-lo:

```
$ ./bch
```

Notem que en les seccions de codi que s'ha afegit a continuació, s'ha eliminat el codi que no és rellevant per a l'explicació, com ara assignar i alliberar blocs de memòria, on s'ha comprovat que s'alliberen correctament amb l'eina Valgrind [Val23]. El codi complet es pot trobar al fitxer `main.c`.

5.1 Inicialització del codi BCH

Per a poder començar a codificar i descodificar, primer de tot s'ha d'inicialitzar el codi BCH, que en aquest cas és trobar el polinomi generador del codi. Per a això, s'ha creat la funció `init()`, que requereix els valors de `m`, que defineix la longitud del codi amb $n = 2^m - 1$, i `t`, el nombre d'errors que pot corregir el codi.

Primer de tot cal inicialitzar el context de cos finit per FLINT, que ens permetrà operar amb els elements del cos finit $\mathbb{F}_{2^m}$. Notar que FLINT utilitza la representació polinomial dels elements, però ens interessa també la representació com a potències de α, arrel primitiva n-èsima de la unitat, per tant, s'ha de pre-computar les potències de α, que s'ha guardat en el vector `alpha_pows`, on `alpha_pows[i]` és α^i.

```
// Inicialitzar el context de cos finit per FLINT.
fmpz_init_set_ui(p, 2);
fq_ctx_init(ctx, p, m, "X");
fq_one(one, ctx);
n = n_pow(2, m) - 1;
// Precomputar les potencies de alpha.
fq_gen(alpha, ctx);
for (slong i = 0; i < n; i++) {
    fq_pow_ui(alpha_pows[i], alpha, i, ctx);
}
```

Recordem que per a utilitzar la fita BCH, necessitem que T, el conjunt de definició, contingui $d-1$ elements consecutius, on $d = 2t+1$. Com que només estem treballant amb codis en sentit estricte, ha de contenir els elements $1, 2, \ldots, d-1$. Per a la construcció d'aquest conjunt T, pel Teorema 3.6, hem de trobar primer les classes 2-ciclotòmiques modul n.

Per a això, tenim la següent secció de codi, que genera les classes 2-ciclotòmiques modul n i els guarda en la matriu `cosets`, on `cosets[i]` és la classe C_i, i el vector `size`, on `size[i]` és la mida de la classe C_i. També s'ha guardat el nombre de classes en la variable `number_cosets`.

El funcionament és bastant simple, començant amb el classe $C_1 = \{1\}$, anem generant els altres elements de la classe, multiplicant l'últim element per 2 i prenent el resultat mòdul n, fins que tornem a trobar el primer element de la classe. Aleshores, busquem un representant de la següent classe, que serà el primer element que no hagi aparegut en cap de les classes anteriors, i ho afegim com el primer element de la classe C_{i+1}. Això es repeteix fins que tots els valors de 0 a $n-1$ hagin aparegut en alguna de les classes, ja que sabem que les classes formen una partició d'aquests nombres.

```
    // Generar les classes 2-ciclotomiques.
    slong cosets[64][8] = {0}, size[64] = {0}, gen_cosets[64] = {0},
          powers[64] = {0};
    slong i = 1, j, temp, matched, next_repr, number_cosets, redundancy = 0,
          number_gen_cosets;
    cosets[1][0] = 1;
    size[0] = 1;
    size[1] = 1;
    do {
        // Generar la classe C_i.
        j = 0;
        temp = cosets[i][j] * 2 % n;
        while (temp != cosets[i][0]) {
            cosets[i][++j] = temp;
            temp = temp * 2 % n;
            size[i]++;
        }
        // Trobar el seguent representant.
        next_repr = 2;
        do {
            next_repr++;
            matched = 0;
            for (slong ii = 1; ii <= i && !matched; ii++) {
                for (slong jj = 0; (jj < size[ii]) && !matched; jj++) {
                    if (cosets[ii][jj] == next_repr) {
                        matched = 1;
                    }
                }
            }
        } while (matched && next_repr < n);
        // Si s'ha trobat un representant, inicialitzar com el primer
    element.
        if (!matched) {
            cosets[++i][0] = next_repr;
            size[i] = 1;
        }
        // Repetir fins que s'hagin trobat totes les classes.
    } while (next_repr < n);
    number_cosets = i;
```

Un cop tenim les classes, hem de trobar quines classes conté els elements $1, 2, \ldots, d-1$, o, equivalentment, quines classes conté algun d'aquests elements. Per tant, per a cada classe, comprovem si algun dels seus elements és menor que d i, en conseqüència, de la llista. Si és així, guardem l'índex de la classe en el vector `gen_cosets`, i incrementem el nombre de `redundancy` $= |T|$ amb la mida de la classe. També guardem el nombre de classes que contenen algun dels elements en la variable `number_gen_cosets`.

```
    // Trobar quines classes conte els valors 1, 2, ..., d-1.
    temp = 0;
    for (i = 1; i <= number_cosets; i++) {
        matched = 0;
        for (j = 0; (j < size[i]) && !matched; j++) {
            if (cosets[i][j] < d) {
                matched = 1;
                gen_cosets[temp++] = i;
            }
        }
        // Si aquesta classe conte algun valor, ajustar el nombre de
    redundancia.
        if (matched) {
            redundancy += size[i];
        }
    }
    number_gen_cosets = temp;
```

Pel mateix Teorema 3.6, sabem que la dimensió del codi és $k = n - |T|$. Per tant, cal comprovar que $k \geq 1$, ja que si no, la longitud del missatge seria 0, i no es podria construir el codi.

```
    k = n - redundancy;
    if (k < 1) {
        flint_printf("No es pot construir el codi.\n");
        exit(1);
    }
```

Finalment, només ens queda trobar el polinomi generador del codi. Per a això, utilitzem un cop més el Teorema 3.6, que ens diu que $g(x) = \prod_{i \in T} x - \alpha^i$. Per tant, obtenim els valors de T en el vector `powers`, i anem generant els polinomis $x - \alpha^i$ i els multipliquem per $g(x)$, que inicialment és 1. Això ho fem amb la funció void fq_poly_mul(fq_poly_t rop, const fq_poly_t op1, const fq_poly_t op2, const fq_ctx_t ctx) que ens proporciona FLINT. Un cop obtingut el polinomi generador, el guardem en la variable global `g`.

```
    // Obtenir els elements de les classes trobades.
    temp = 0;
    for (i = 0; i < number_gen_cosets; i++) {
        for (j = 0; j < size[gen_cosets[i]]; j++) {
            powers[temp++] = cosets[gen_cosets[i]][j];
        }
    }
    // Obtenir el polinomi generador del codi.
    fq_poly_one(g, ctx);
    for (i = 0; i < redundancy; i++) {
        fq_poly_gen(f, ctx);
        fq_poly_set_coeff(f, 0, alpha_pows[powers[i]], ctx);
        fq_poly_mul(g, g, f, ctx);
    }
```

5.2 Codificació

Per a la codificació del missatge, s'ha utilitzat el mètode sistemàtic descrit en la secció 3.3. Sigui `f` $= f(x)$ el polinomi que representa el missatge, notem que hem fet servir la lletra f en comptes de m, ja que aquesta ja s'ha utilitzat en el codi. Per tant, el missatge codificat serà el polinomi $c(x) = x^{n-k}f(x) - r(x)$, on obtenim $r(x)$ fent la divisió entera de polinomis $x^{n-k}f(x) = q(x)g(x) + r(x)$. Aquesta última operació es realitza amb el mètode void fq_poly_rem(fq_poly_t R, const fq_poly_t A, const fq_poly_t B, const fq_ctx_t ctx) que ofereix FLINT per a obtenir el residu de la divisió de dos polinomis.

El codi complet es troba a `encode()`.

```
    // Obtenir hf(x) = x^{n-k} * f(x).
    fq_poly_t h;
    fq_poly_init(h, ctx);
    fq_poly_set_coeff(h, n - k, one, ctx);
    fq_poly_t hf;
    fq_poly_init(hf, ctx);
    fq_poly_mul(hf, f, h, ctx);

    // Obtenir el residu de dividir hf(x) entre g(x).
    fq_poly_t r;
    fq_poly_init(r, ctx);
    fq_poly_rem(r, hf, g, ctx);

    // Obtenir el missatge codificat c(x) = hf(x) - r(x).
    fq_poly_init(c, ctx);
    fq_poly_sub(c, hf, r, ctx);
```

5.3 Descodificació per Peterson-Gorenstein-Zierler

Veiem a continuació la descodificació implementada amb l'algoritme Peterson-Gorenstein-Zierler, descrit en la secció 4.3.1. El codi complet es troba a `decode()` i `decode_PGZ()`, per al pas 2.

Sigui `y` $= y(x)$ el polinomi que representa el vector rebut, recordem que el primer pas és calcular les síndromes $S_i = y(\alpha^i)$ per a $1 \leq i \leq 2t = d - 1$. Com que ja tenim precomputades les potències de α, només cal avaluar el polinomi $y(x)$ en aquests valors. Això ho fem amb la funció void fq_poly_evaluate_fq(fq_t rop, const fq_poly_t f, const fq_t a, const fq_ctx_t ctx) que ens proporciona FLINT i guardem les síndromes en el vector `syndromes`. També comprovem si totes les síndromes són zero, en aquest cas, no hi ha errors i retornem 0.

```
    // Pas 1: Computar les sindromes.
    fq_t syndromes[d];
    slong error = 0;
    for (slong i = 1; i < d; i++) {
        fq_init(syndromes[i], ctx);
        fq_poly_evaluate_fq(syndromes[i], y, alpha_pows[i], ctx);
        if (!fq_is_zero(syndromes[i], ctx)) {
            error = 1;
        }
    }
    if (!error) {
        flint_printf("No s'ha detectat errors.\n");
        return 0;
    }
```

Per al segon pas, hem de trobar el polinomi localitzador d'error `sigma`= $\sigma(x)$. Seguint l'algoritme, començant amb `mu` $= \mu = t$, hem de comprovar que la matriu M_μ, definida al lema 4.6, sigui invertible, si no ho és, decreixem μ i tornem a comprovar. Com que totes aquestes matrius són submatrius de M_t, l'inicialitzem primer a `M`, llavors podem obtenir les matrius M_μ amb el diferents valors de μ amb void fq_mat_window_init(fq_mat_t window, const fq_mat_t mat, slong r1, slong c1, slong r2, slong c2, const fq_ctx_t ctx), que ens permet obtenir una submatriu de `M`, amb les files i columnes especificades.

Per a comprovar si una matriu és invertible, FLINT ens proporciona la funció int fq_mat_inv(fq_mat_t B, fq_mat_t A, const fq_ctx_t ctx), que ens retorna 1 si la matriu és invertible, i 0 si no ho és. Alhora, si la matriu és invertible, ens retorna la inversa que guardem a `M_inv`. D'aquí obtenim el valor de ν, el nombre d'errors que s'ha produït, notem que pot ocórrer que $\nu > t$, en aquest cas cap de les matrius M_μ serà invertible i, per tant, no es pot corregir l'error.

Observació 5.1. És interessant veure com troba FLINT la inversa de la matriu. Com que la documentació no en fa menció, hem hagut de mirar en el codi font de la llibreria. Observem llavors, que internament, ignorant casos trivials, utilitza un altre mètode per a resoldre l'equació matricial $AB = I$, on I és la matriu identitat, que és exactament la definició de matriu inversa. Per resoldre aquesta equació, FLINT empra la descomposició LU, per tant, la funció fq_mat_inv() té complexitat $O(n^3)$[Far18], on n és la dimensió de la matriu.

Ara només en queda resoldre l'equació matricial descrita a (4.5) per a obtenir $\sigma(x)$. Si considerem que les matrius que apareixen a (4.5) l'escrivim com $MS = V$, llavors tenim que $S = M^{-1}V$, on M^{-1} ja s'ha calculat i V és formada per les síndromes que també tenim. Amb els elements de S, podem construir el polinomi $\sigma(x)$, que serà el polinomi localitzador d'error i el guardem a `sigma`.

```
1     // Pas 2:
2     // Obtenir la matriu M.
3     fq_mat_t M;
4     fq_mat_init(M, t, t, ctx);
5     for (slong i = 0; i < t; i++) {
6         for (slong j = 0; j < t; j++) {
7             fq_mat_entry_set(M, i, j, syndromes[i + j + 1], ctx);
8         }
9     }
10    // Decrementar mu fins que la matriu sigui invertible.
11    slong mu = t + 1, invertible = 0;
12    fq_mat_t M_inv, M_mu;
13    while (!invertible && mu > 1) {
14        mu--;
15        fq_mat_window_init(M_mu, M, 0, 0, mu, mu, ctx);
16        fq_mat_init(M_inv, mu, mu, ctx);
17        invertible = fq_mat_inv(M_inv, M_mu, ctx);
18    }
19    if (!invertible) {
20        flint_printf("No s'ha pogut corregir l'error, ha fallat al pas 2.\n"
      );
21        return 1;
22    }
23    slong nu = mu;
24    // Resoldre l'equacio matricial per obtenir sigma, el polinomi
      localitzador d'error.
25    fq_mat_t V, S;
```

```
    fq_mat_init(V, nu, 1, ctx);
    fq_mat_init(S, nu, 1, ctx);
    for (slong i = 0; i < nu; i++) {
        fq_mat_entry_set(V, i, 0, syndromes[nu + i + 1], ctx);
    }
    fq_mat_mul(S, M_inv, V, ctx);
    fq_poly_set_coeff(sigma, 0, one, ctx);
    for (slong i = 0; i < nu; i++) {
        fq_poly_set_coeff(sigma, i + 1, fq_mat_entry(S, nu - i - 1, 0), ctx)
    ;
    }
```

El tercer pas consisteix a trobar les arrels de $\sigma(x)$ i invertir-les per a obtenir els nombres de localització d'error. Com ja s'ha comentat, això normalment es fa amb una cerca exhaustiva computant $\sigma(\alpha^i)$ per a $1 \leq i \leq n$. Per tant, avaluem $\sigma(x)$ amb els valors de α^i que ja tenim pre-computats, i si el resultat és zero, guardem $n-i$ en el vector `loc`, ja que tenim que $(\alpha^i)^{-1} = \alpha^{n-i}$, excepte en el cas que $i = 0$, ja que $(\alpha^0)^{-1} = 1^{-1} = 1 = \alpha^0$. Si el nombre d'arrels trobades és diferent de ν, el nombre d'errors que hauria de tenir $\sigma(x)$ pel pas 2, llavors el vector rebut tenia més errors dels que es poden corregir i, per tant, no es pot corregir l'error.

```
    // Pas 3: Obtenir els nombres de localitzacio d'error.
    slong nu = fq_poly_degree(sigma, ctx);
    slong loc[nu];
    slong temp = 0;
    for (slong i = 0; i < n && temp < nu; i++) {
        fq_poly_evaluate_fq(result, sigma, alpha_pows[i], ctx);
        if (fq_is_zero(result, ctx)) {
            if (i == 0)
                loc[temp++] = 0;
            else
                loc[temp++] = n - i;
        }
    }
    if (temp != nu) {
        flint_printf("No s'ha pogut corregir l'error, ha fallat al pas 3.\n"
    );
        return 1;
    }
```

Per al pas 4, hem de trobar les magnituds dels errors, però com ja hem vist en les observacions de l'algoritme, com que estem treballant en codis binaris, només ens cal saber quins són els nombres de localització d'error, ja que la magnitud serà sempre 1. Per tant, només ens fa falta corregir l'error.

```
    // Pas 4: Les magnituds d'error son 1 per ser un codi binari.
    for (slong i = 0; i < nu; i++) {
        fq_poly_set_coeff(e, loc[i], one, ctx);
    }
    // Obtenir el vector corregit, c_hat(x) = y(x) - e(x).
    fq_poly_sub(c_hat, y, e, ctx);
```

Com a última comprovació, cal verificar que el vector corregit pertany al codi, que es pot realitzar fàcilment utilitzant el Teorema 3.3, comprovant que $\hat{c}(x)$ és divisible per $g(x)$.

```
    // Comprovar que el vector corregit es del codi.
    fq_poly_rem(r, c_hat, g, ctx);
    if (!fq_poly_is_zero(r, ctx)) {
        flint_printf(
```

```
5             "No s'ha pogut corregir l'error, ha fallat a la comprovacio.\n")
    ;
6         return 1;
7     }
```

Finalment, cal recuperar el missatge original agafant els termes de $\hat{c}(x)$ amb grau igual o major que $n-k$, ja que hem fet servir la codificació sistemàtica. Això ho fem amb la funció void fq_poly_shift_right(fq_poly_t rop, const fq_poly_t op, slong n, const fq_ctx_t ctx), que redueix el grau de tots els termes del polinomi. Llavors tenim el missatge corregit `f_hat`, que és el resultat de la descodificació.

```
1     // Obtenir el missatge original.
2     fq_poly_t f_hat;
3     fq_poly_init(f_hat, ctx);
4     fq_poly_shift_right(f_hat, c_hat, n - k, ctx);
```

5.4 Descodificació per Berlekamp-Massey

La implementació de l'algoritme Berlekamp-Massey, descrit en la secció 4.3.2, es troba a la funció `decode_BM()`. Recordem que aquest algoritme només modifica el pas 2 de l'algoritme PGZ, per tant, per als altres passos, se segueix utilitzant el mateix codi en `decode()`.

De la descripció del pas iteratiu de l'algoritme, observem que si el realitzéssim manualment ens interessaria anar emplenant una taula com la següent:

μ	$\sigma^{(\mu)}(x)$	Δ_μ	d_μ	$2\mu - d_\mu$
$-1/2$	1	1	0	-1
0	1	S_1	0	0
1				
$\vdots$				
t				

On l'última columna és la que ens ajuda a decidir el valor de ρ per al cas II.

Per al programa, només requerim les dues primeres columnes, ja que els valors de d_μ i $2\mu - d_\mu$ els podem obtenir fàcilment. Òbviament, si les guardem en dos vectors, ens interessa que μ sigui l'índex dels valors. Però com que tenim una fila abans de $\mu = 0$, caldrà fer un petit canvi de variables, per tant, utilitzarem `mu` $= \mu + 1$ com a índex dels vectors. A més, s'ha de considerar el cas particular que `mu` $= 0$, que li correspon el valor de $\mu = -1/2$. Doncs, estem omplint dos vectors com la taula següent:

μ	`mu`	$\sigma^{(\mu)}(x)$	Δ_μ
$-1/2$	0	1	1
0	1	1	S_1
1			
$\vdots$			
t	$t+1$		

Observem, però, que no és necessari guardar $\sigma^{(\mu)}(x)$ i Δ_μ per a tots els valors de μ. A cada iteració, només requerim aquests valors per a μ i ρ, ja que la fórmula és $\sigma^{(\mu+1)}(x) = \sigma^{(\mu)}(x) + \Delta_\mu \Delta_\rho^{-1} x^{2(\mu-\rho)} \sigma^{(\rho)}(x)$. El problema que tenim és que no sabem quin és el valor de ρ i, segons l'algoritme, a cada iteració s'ha de calcular com $-(1/2) \leq \rho < \mu$ tal que $\Delta_\rho \neq 0$ i $2\rho - d_\rho$ sigui el més gran possible. La solució ho proporciona la condició de

més gran possible, per tant, podem calcular a cada iteració el valor de ρ per a la següent iteració, en comptes de per a la iteració actual. Això ho podem fer de la següent manera: si $2\mu - d_\mu$ és més gran que `rho_value` $= 2\rho - d_\rho$ i $\Delta_\mu \neq 0$, actualitzem `rho_value` i guardem el valor de μ en `rho`, $\sigma^{(\mu)}(x)$ en `sigma_rho` i Δ_μ en `delta_rho`. Notem que si les condicions no es compleixen, pel valor de μ actual, els $\sigma^{(\mu)}(x)$ i Δ_μ mai s'utilitzaran com a ρ, per tant, només caldrà guardar-les per a la iteració següent i després podran ser descartades.

Amb això, només necessitem dues parelles de variables, `sigma_mu` i `sigma_mu_next` per a guardar els valors de $\sigma^{(\mu)}(x)$ i $\sigma^{(\mu+1)}(x)$, respectivament, i `delta_mu` i `delta_mu_next` per a les discrepàncies.

Per tant, primer de tot, inicialitzem els valors inicials.

```
    // Inicialitzar els polinomis sigma i les discrepancies.
    fq_poly_t sigma_mu, sigma_mu_next, sigma_rho;
    fq_t discrepancy_mu, discrepancy_mu_next, discrepancy_rho, temp;
    slong exponent, degree_mu, rho = 0, rho_value = -1;
    fq_poly_one(sigma_mu, ctx);
    fq_poly_one(sigma_rho, ctx);
    fq_set(discrepancy_mu, syndromes[1], ctx);
    fq_one(discrepancy_rho, ctx);
```

Un cop inicialitzades, s'ha de realitzar el pas iteratiu en ordre, començant a $\mu = 0$ fins a $\mu = t - 1$. Que per a `mu` equival a:

```
    for (slong mu = 1; mu < t + 1; mu++) {
        // Pas iteratiu.
    }
```

Per a cada μ, primer hem de calcular $\sigma^{(\mu+1)}(x)$. Aquí, l'algoritme té dos casos, el cas $\Delta_\mu = 0$ és trivial, per tant, veiem el cas $\Delta_\mu \neq 0$. Com que ja tenim ρ precalculada de la iteració anterior, només cal aplicar la fórmula $\sigma^{(\mu+1)}(x) = \sigma^{(\mu)}(x) + \Delta_\mu \Delta_\rho^{-1} x^{2(\mu-\rho)} \sigma^{(\rho)}(x)$. Això ho fem amb les diverses funcions que ens proporciona FLINT i que ja hem vist en la secció anterior. Notem també que s'ha de tenir en compte amb el que s'ha comentat abans sobre el valor de `mu`, que s'aplica també per a `rho`.

```
        // Obtenir el polinomi sigma^{(mu+1)}.
        if (fq_is_zero(discrepancy_mu, ctx)) {
            fq_poly_set(sigma_mu_next, sigma_mu, ctx);
        } else {
            fq_inv(temp, discrepancy_rho, ctx);
            fq_mul(temp, discrepancy_mu, temp, ctx);
            if (rho == 0) {
                exponent = 2 * (mu - 1) + 1;
            } else {
                exponent = 2 * (mu - 1 - (rho - 1));
            }
            fq_poly_set_coeff(sigma_mu_next, exponent, temp, ctx);
            fq_poly_mul(sigma_mu_next, sigma_mu_next, sigma_rho, ctx);
            fq_poly_add(sigma_mu_next, sigma_mu, sigma_mu_next, ctx);
        }
```

A continuació s'ha de calcular la discrepància $\Delta_{\mu+1}$. Igual que abans, només cal aplicar la fórmula tenint en compte el valor de `mu`. Notem que si $\mu + 1 = t$, ja tenim el polinomi $\sigma^{(t)}(x)$, per tant, no cal calcular la discrepància i podem acabar el bucle.

```
        // Obtenir la discrepancia Delta_{mu+1}.
        if (mu == t) break;
```

```
        for (slong j = 1; j < 2 * mu; j++) {
            fq_poly_get_coeff(temp, sigma_mu_next, j, ctx);
            fq_mul(temp, temp, syndromes[2 * mu + 1 - j], ctx);
            fq_add(discrepancy_mu_next, discrepancy_mu_next, temp, ctx);
        }
        fq_poly_get_coeff(temp, sigma_mu, 2 * mu + 1, ctx);
        fq_add(discrepancy_mu_next, discrepancy_mu_next, temp, ctx);
        fq_add(discrepancy_mu_next, discrepancy_mu_next,
                syndromes[2 * mu + 1], ctx);
```

A continuació, actualitzem, si és aplicable, el valor de ρ per a la següent iteració com ja s'ha comentat abans.

```
        // Actualitzar rho.
        degree_mu = fq_poly_degree(sigma_mu, ctx);
        if (!fq_is_zero(discrepancy_mu, ctx) &&
            2 * (mu - 1) - degree_mu > rho_value) {
            rho = mu;
            rho_value = 2 * (mu - 1) - degree_mu;
            fq_poly_set(sigma_rho, sigma_mu, ctx);
            fq_set(discrepancy_rho, discrepancy_mu, ctx);
        }
```

Dins de bucle, només en queda intercanviar les variables per a preparar la següent iteració.

```
        // Intercanviar els valors per la seguent iteracio.
        fq_poly_set(sigma_mu, sigma_mu_next, ctx);
        fq_poly_zero(sigma_mu_next, ctx);
        fq_set(discrepancy_mu, discrepancy_mu_next, ctx);
        fq_zero(discrepancy_mu_next, ctx);
```

Finalment, cal assignar el polinomi localitzador d'error $\sigma(x)$ a la variable `sigma` per a què es pugui utilitzar en el pas 3 de `decode()`. També cal comprovar que el grau de $\sigma(x)$ sigui correcte.

```
    fq_poly_set(sigma, sigma_mu_next, ctx);
    if (fq_poly_degree(sigma, ctx) > t) {
        flint_printf("No s'ha pogut corregir l'error, ha fallat al pas 2.\n"
    );
        return 1;
    }
```

5.5 Descodificació per Sugiyama

L'algoritme de Sugiyama, programat a la funció `decode_Sugiyama()` i descrit a la secció 4.3.3, és el més senzill quant a implementació.

Observem primer que en el pas iteratiu i de l'algoritme, només s'utilitza els valors computats a les iteracions $i-1$ i $i-2$. Per tant, no cal guardar-se la llista completa dels diferents polinomis $r_i(x)$ i $b_i(x)$, només necessitem les variables `r` i `b` que contenen els polinomis per a $i-1$ i les variables `r_prev` i `b_prev` per a $i-2$.

Seguint l'algoritme, el pas I és inicialitzar les variables als seus valors inicials.

```
    // Inicialitzar r_{-1}(x), r_0(x), b_{-1}(x), b_0(x).
    fq_poly_t r, r_prev, b, b_prev, h;
    fq_poly_set_coeff(r_prev, 2*t, one, ctx);
    for (slong i = 0; i < 2*t; i++) {
        fq_poly_set_coeff(r, i, syndromes[i + 1], ctx);
```

```
    }
    fq_poly_set_coeff(b, 0, one, ctx);
```

En el pas iteratiu II, només cal realitzar els càlculs $r_{i-2}(x) = r_{i-1}h_i(x) + r_i(x)$ i $b_i(x) = b_{i-2}(x) - h_i(x)b_{i-1}(x)$, utilitzant les variables que hem descrit abans. Notem que per trobar a $r_i(x)$ podem sobreescriure `r_prev`, ja que no necessitem més el polinomi $r_{i-2}(x)$, com també per a $b_i(x)$, podem sobreescriure `b_prev`. Només ens caldrà intercanviar els valors de `r` amb `r_prev` i `b` amb `b_prev` al final de cada iteració per a preparar la següent. Això ho repetim fins que $\text{gr}(r_i(x)) < t$.

```
    do {
        // Pas iteratiu.
        // Obtenir r_i(x).
        fq_poly_divrem(h, r_prev, r_prev, r, ctx);
        fq_poly_swap(r_prev, r, ctx);
        // Obtenir b_i(x).
        fq_poly_mul(h, h, b, ctx);
        fq_poly_sub(b_prev, b_prev, h, ctx);
        fq_poly_swap(b_prev, b, ctx);
    } while (fq_poly_degree(r, ctx) >= t);
```

Com s'ha comentat abans, com que només ens interessa les arrels de polinomi localitzador d'error, podem simplement considerar com a $\sigma(x)$ el polinomi $b_I(x)$, per tant, assignem com a `sigma` el polinomi `b`, que conté l'últim $b_i(x)$ que s'ha calculat. Finalment, comprovem que el grau de $\sigma(x)$ sigui correcte.

```
    // Obtenir sigma.
    fq_poly_set(sigma, b, ctx);
    if (fq_poly_degree(sigma, ctx) > t) {
        flint_printf("No s'ha pogut corregir l'error, ha fallat al pas 2.\n"
    );
        return 1;
    }
```

5.6 Comparació dels algoritmes

Per a comparar els algoritmes, s'ha utilitzat la funció `benchmark()`. Aquesta funció, donats els valors de m i t, primer inicialitza el codi BCH amb aquests paràmetres i troba el polinomi generador $g(x)$ del codi, això ho fa amb el mètode `init()` descrit anteriorment.

Després, genera un missatge aleatori de longitud adequada amb `generate_message()` i el codifica amb `encode()`. A continuació, obtenim el nombre d'errors que volem introduir, que serà el mateix que el nombre de bits que volem canviar, aquest valor l'agafarem amb una distribució geomètrica amb probabilitat $p = 0.5$. Això ho fem amb la funció `geometric_random()`. Aquesta funció aprofita el fet que si poden obtenir un valor aleatori entre 0 i 1 amb distribució uniforme, $U \sim U(0,1)$, llavors $X = \lfloor \frac{\ln(U)}{\ln(1-q)} \rfloor$ té distribució geomètrica amb probabilitat q. Per tant, només cal obtenir un valor aleatori entre 0 i 1 i aplicar la fórmula anterior. Això ho fem amb la funció `rand()` que ens proporciona C i dividir-ho amb l'enter màxim `RAND_MAX`.

```
slong geometric_random(slong max) {
    double random = rand() / (double)RAND_MAX;
    slong value = (slong)(log(random) / log(0.5));
    if (value > max) {
        return max;
    }
```

```
    return value;
}
```

Un cop tenim el nombre d'errors, cal decidir quins bits canviarem. Per a això, utilitzem la funció `generate_error()`, que fa servir l'algoritme de Fisher-Yates per a generar les posicions aleatòries on afegir l'error.

```
void generate_error() {
    for (slong i = 0; i < n; i++) {
        error_locations[i] = i;
    }
    // Remenar amb Fisher-Yates.
    for (slong i = n - 1; i > 0; i--) {
        slong j = rand() % (i + 1);
        slong temp = error_locations[i];
        error_locations[i] = error_locations[j];
        error_locations[j] = temp;
    }
}
```

Llavors, introduïm l'error en el missatge codificat amb `introduce_error()`. Repetim aquest procés un nombre determinat de vegades, on guardem el vector rebut en cada iteració. Finalment, descodifiquem tots els missatge amb els tres algoritmes i guardem el temps que triga cada un amb `clock()`.

```
void benchmark(slong iterations, slong max_errors, slong total_time) {
    clock_t start, end;
    slong time_PGZ = 0, time_BM = 0, time_S = 0;
    srand(time(NULL));
    num_errors = t;
    num_messages = iterations;
    init();
    for (slong i = 0; i < iterations; i++) {
        generate_message();
        encode(i);
        if (!max_errors) num_errors = geometric_random(255);
        generate_error();
        introduce_error(i);
    }

    // Peterson-Gorenstein-Zierler
    start = clock();
    for (slong i = 0; i < iterations; i++) {
        decode(Peterson_Gorenstein_Zierler, i);
    }
    end = clock();
    time_PGZ = end - start;

    // Berlekamp-Massey
    start = clock();
    for (slong i = 0; i < iterations; i++) {
        decode(Berlekamp_Massey, i);
    }
    end = clock();
    time_BM = end - start;

    // Sugiyama
    start = clock();
    for (slong i = 0; i < iterations; i++) {
        decode(Sugiyama, i);
```

```
36      }
37      end = clock();
38      time_S = end - start;
39      // Mostrar els resultats.
40      clear();
41  }
```

Notem que per a què el resultat sigui just, tots tres algoritmes intenten descodificar el mateix vector rebut en cada iteració.

Per a aquesta comparació, s'ha utilitzat codis BCH de longitud $n = 255$ i que poden corregir $t = 5$, $t = 10$, $t = 15$, $t = 20$ i $t = 25$ errors. Per als dos primers s'han realitzat $2^{11} = 2048$ iteracions i per als tres últims $2^{16} = 65536$ iteracions, aquests nombres s'han escollit per a què de mitjana hi hagi un error que no es pugui corregir per als casos $t = 10$ i $t = 15$. Els resultats són els següents, on el temps és en mil·lisegons:

t	PGZ	BM	Sugiyama
5	0.579	0.562	0.569
10	1.137	1.080	1.093
15	1.715	1.558	1.576
20	2.361	2.075	2.093
25	3.063	2.606	2.631

L'usuari pot executar aquesta comparació, notem que pot trigar bastant, amb:

```
1   $ ./bch benchmark
```

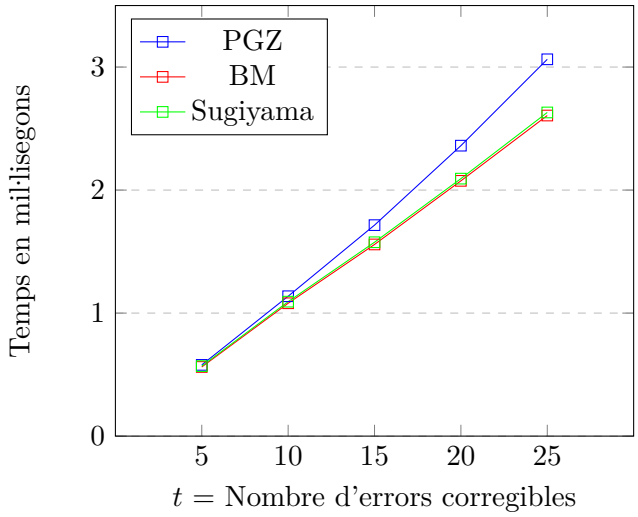

Figura 2: Temps mitjà de descodificació per a n=255 amb nombre d'errors donats per una distribució geomètrica amb probabilitat $p = 0.5$

Recordem de l'anàlisi teòrica de la complexitat dels algoritmes que l'algoritme Peterson-Gorenstein-Zierler té complexitat $O(tn + t^4)$, l'algoritme Berlekamp-Massey $O(tn + t^3)$ i l'algoritme de Sugiyama $O(tn + t^3)$. Però com que estem treballant amb codis binaris, no fem el pas 4 de l'algoritme, per tant, les complexitats són $O(tn + t^4)$, $O(tn + t^2)$ i $O(tn + t^2)$, respectivament.

De la gràfica en la figura 2, podem veure com l'algoritme Peterson-Gorenstein-Zierler és el més lent dels tres, on la diferència creix a mesura que augmenten el nombre d'errors que

es pot corregir, metre que els altres dos són molt similars, amb l'algoritme Berlekamp-Massey lleugerament més eficient que l'algoritme de Sugiyama.

Llavors, el resultat que obtenim és consistent amb l'anàlisi teòrica de la complexitat dels algoritmes que s'ha fet anteriorment, el que no és consistent és la corba que formen, on tots tres semblen tenir una complexitat lineal respecte de t.

Una possible explicació és el fet que en la nostra comparació, on volem un escenari realista en el nombre d'errors que es pot produir, tenim $P(X = 0) = 1/2$, per tant, en la meitat de les iteracions no hi ha cap error i, en conseqüència, tots tres algoritmes només computen les síndromes en el primer pas i acaben, deixant la complexitat a $O(nt)$, on n és la longitud del codi, que sí és lineal respecte a la variable t.

Comparem ara aquests tres algoritmes forçant un escenari on hi hagi errors en tots els missatges, en particular, que tots els vectors rebuts tinguin t errors. Farem servir els mateixos paràmetres $n = 255$ i $t = 5$, $t = 10$, $t = 15$, $t = 20$ i $t = 25$, però, farem 2048 iteracions per a cada cas. Els resultats són els següents, on el temps és en mil·lisegons:

t	PGZ	BM	Sugiyama
5	0.931	0.873	0.906
10	2.024	1.769	1.866
15	3.293	2.650	2.830
20	4.950	3.560	3.882
25	7.378	4.504	4.969

L'usuari pot executar aquesta comparació amb:

```
$ ./bch benchmark 1
```

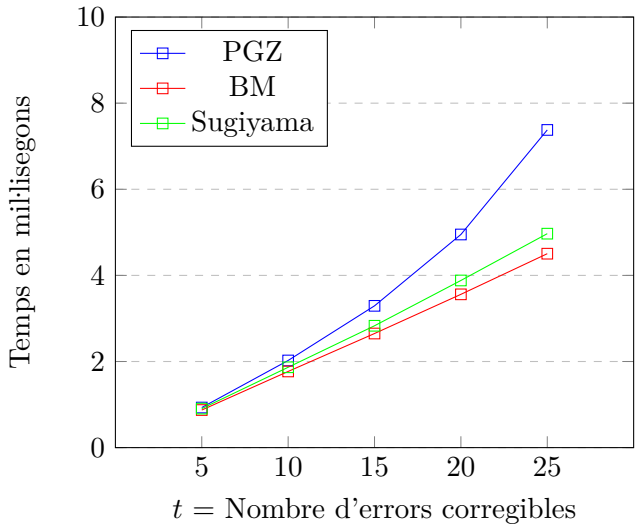

Figura 3: Temps mitjà de descodificació per a n=255 amb t errors

En aquesta comparació, els resultats són similars, però sí que podem observar que l'algoritme Peterson-Gorenstein-Zierler no té una corba lineal respecte de t, però tots tres algoritmes segueixen tenint una corba diferent respecte de la complexitat teòrica.

Una altra explicació és el valor de n que estem utilitzant, repetim la comparació anterior amb $n = 63$ i $t = 2$, $t = 4$, $t = 6$, $t = 10$ i $t = 15$, amb 2048 iteracions per a cada cas. Els resultats són els següents, on el temps és en mil·lisegons:

t	PGZ	BM	Sugiyama
2	0.079	0.073	0.080
4	0.165	0.142	0.159
6	0.266	0.205	0.233
10	0.548	0.336	0.396
15	1.163	0.517	0.638

L'usuari pot executar aquesta comparació amb:

```
$ ./bch benchmark 2
```

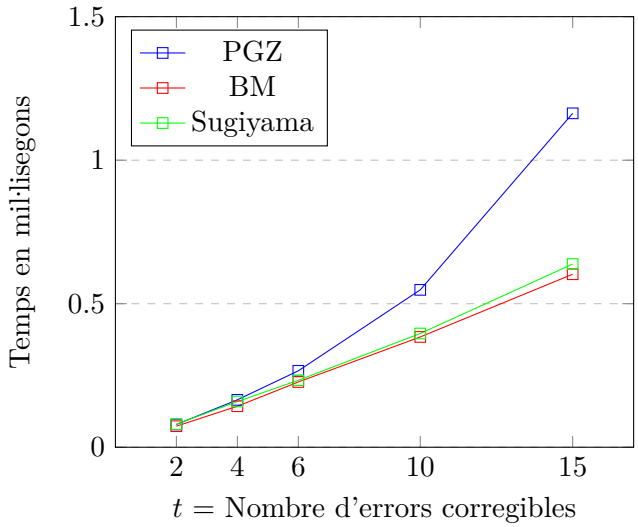

Figura 4: Temps mitjà de descodificació per a n=63 amb t errors

Finalment, fem un cas més pràctic. Ja s'ha comentat que els codis BCH s'utilitzen com a codi corrector d'errors per a unitats d'estat sòlid, veiem si aquesta implementació seria útil per a escoltar una cançó. Si la cançó està guardat en format mp3, encara que depèn de la taxa de bits, podem suposar que un minut de música és 1MB. Considerem llavors que tenim una cançó de tres minuts i, per tant, de 3MB.

Un codi BCH de longitud $n = 255$ i que pot corregir $t = 10$ errors codifica un missatge de 179 bits. Per tant, com que 3MB són $3 \cdot 8 \cdot 10^6$ bits, i tenim que $3 \cdot 8 \cdot 10^6/179 < 134079$, llavors podem guardar aquest fitxer codificat com 134079 missatges. Tornem al cas més realista on introduïm errors amb una distribució geomètrica de probabilitat $p = 0.5$, veiem quant triga cada algoritme en descodificar-ho, temps en segons:

PGZ	BM	Sugiyama
152.7	143.8	145.6

L'usuari pot executar aquesta comparació amb:

```
$ ./bch benchmark 3
```

Com que tres minuts són 180 segons, veiem que amb aquesta implementació dels algoritmes, tots tres algoritmes són viables per a descodificar la cançó en temps real.

Un cop vista totes aquestes comparacions, podem concloure que Berlekamp-Massey és un bon algoritme per a descodificar codis BCH, on podem veure que és l'algoritme que s'ha escollit per a implementar la descodificació de codis BCH en la llibreria interna del nucli de Linux[Ker23].

Referències

[Ber15] E.R. Berlekamp. *Algebraic Coding Theory (Revised Edition)*. World Scientific Publishing Company, 2015.

[BRC60] R.C. Bose and D.K. Ray-Chaudhuri. On a class of error correcting binary group codes. *Information and Control*, 3(1):68–79, March 1960.

[Cha98] P. Charpin. Open problems on cyclic codes. In *Handbook of Coding Theory*. Elsevier, 1998.

[CP88] K.M. Cheung and F. Pollara. Phobos lander coding system: Software and analysis. 1988.

[Cre20] M.T. Crespo. *Estructures Algebraiques*. Universitat de Barcelona, 2020.

[Cre21] M.T. Crespo. *Equacions Algebraiques*. Universitat de Barcelona, 2021.

[Far18] R.W. Farebrother. *Linear Least Squares Computations*. CRC Press, 2018.

[FLI23] *FLINT: Fast Library for Number Theory*, 2023. Version 2.9.0, `https://flintlib.org`.

[HKS21] W.C. Huffman, J.L. Kim, and P. Solé. *Concise Encyclopedia of Coding Theory*. CRC Press, 2021.

[Hoc59] A. Hocquenghem. Codes correcteurs d'erreurs. *Chiffres*, 2:147–156, 1959.

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

[ISO00] ISO. Information technology – automatic identification and data capture techniques – bar code symbology – qr code. Standard ISO 18004, International Organization for Standardization, 2000.

[Ker23] *The Linux Kernel*, 2023. Version 6.7.0, `https://www.kernel.org`.

[MME12] R. Micheloni, A. Marelli, and K. Eshghi. *Inside Solid State Drives (SSDs)*. Springer Series in Advanced Microelectronics. Springer Netherlands, 2012.

[Pan66] V.Y. Pan. Methods of computing values of polynomials. *Russian Mathematical Surveys*, 21(1):105–136, 1966.

[Rom92] S. Roman. *Coding and Information Theory*. Graduate texts in mathematics. Springer-Verlag, 1992.

[Sha48] C.E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 1948.

[Tra20] A. Travesa. *Equacions algebraiques*. Universitat de Barcelona, 2020.

[Val23] *Valgrind*, 2023. Version 3.22.0, `https://valgrind.org`.