Grau en Matemàtiques

2019/2020

Ampliació de software matemàtic en teoria de codis:

Conjunts d'informació de codis q-aris

Pol Piñol Castuera

Directora: Mercè Villanueva Gay

Barcelona, 26/06/2020

Abstract

Code theory is based on studying methods to determine the most economical, fast and secure way to transmit a message through a channel avoiding possible noise interference. To achieve a more reliable and accurate communication, messages are encoded using so-called error-correcting codes. The main goal of this project is to contribute to the process of developing a Magma [1] software library that works with nonlinear codes in an efficient way. We will focus on the design and implementation of five functions related to systematic codes. These are: `IsInformationSet(C, I)`, `IsSystematic(C)`, `InformationSpace(C)`, `InformationSet(C)`, and `AllInformationSets(C)`, where C is a non necessarily linear code and I a set of coordinate positions. We will design different algorithms for each function, we will test theoretical results to improve the efficiency of some algorithms, we will create some tests to approve the functions and finally, we will do a performance study of all the proposed algorithms.

Resum

La teoria de codis es basa en estudiar mètodes per determinar la forma més econòmica, ràpida i segura de transmetre un missatge a través d'un canal evitant les possibles interferències de sorolls. Per aconseguir una comunicació més fiable i exacta, els missatges es codifiquen fent servir els anomenats codis correctors d'errors. L'objectiu principal d'aquest treball és contribuir en el procés de desenvolupament d'una llibreria del software Magma [1] que treballa amb codis no lineals d'una forma eficient. Ens centrarem en el disseny i implementació de cinc funcions relacionades amb els codis sistemàtics. Aquestes són: `IsInformationSet(C,I)`, `IsSystematic(C)`, `InformationSpace(C)`, `InformationSet(C)` i `AllInformationSets(C)`, on C és un codi no necessàriament lineal i I un conjunt de posicions de coordenades. Dissenyarem diferents algorismes per cada funció, provarem resultats teòrics per millorar l'eficiència d'alguns algorismes, crearem alguns tests per provar les funcions i finalment, farem un estudi de rendiment de tots els algorismes plantejats.

Agraïments a la Mercè Villanueva Gay, tutora del treball de fi de grau des de principis d'estiu de l'any 2019, pel seu constant suport, per les seves correccions , per l'orientació i la confiança d'acompanyar-me durant aquest últim projecte de matemàtiques.

A la meva parella Mireia, per estimar-me tant i acompanyar-me en els últims tres anys i mig de la meva vida.

A la meva família: els meus pares Olga i Joan, i al meu germà Jan, amb el seu constant suport i comprensió.

Índex

1 Introducció

L'any 1948, Claude Elwood Shannon va publicar l'article "A mathematical theory of communication" [2], on presentava la seva proposta teòrica anomenada teoria de la informació o també coneguda com a teoria matemàtica de la comunicació.

Sense entrar en gaires detalls, el model proposat per Shannon descriu el viatge d'un senyal enviat per un emissor a un receptor. El senyal originalment parteix d'una font d'informació que, gràcies a un transmissor, viatjarà a través d'un canal fins a un receptor. Durant el transcurs del recorregut, Shannon descriu la possibilitat d'uns certs sorolls que podrien alterar el senyal i també d'un espia que vulgui interceptar el missatge enviat.

En el cas de la possibilitat d'un espia es pot realitzar un procés de xifratge que ens garantiria la seguretat del missatge. L'altre cas que descriu Shannon, és quan el missatge original és alterat degut a certs sorolls fent que apareguin errors i interferències en el senyal inicial. En conseqüència, el receptor no rebria el mateix missatge que s'havia enviat originalment. Per aconseguir una comunicació més fiable i exacta, els missatges es codifiquen fent servir els anomenats codis correctors d'errors. Aquest últim procés està contingut dins l'anomenada teoria de la codificació o teoria de codis, la qual anirem desenvolupant durant el transcurs d'aquest treball.

Resumint, l'objectiu de la teoria de codis es basa en estudiar mètodes per determinar la forma més econòmica, ràpida i segura de transmetre el missatge evitant les possibles interferències de sorolls [3].

L'objectiu princial d'aquest treball és contribuir en el procés de desenvolupament d'una llibreria del software Magma que treballa amb codis no lineals d'una forma eficient [4]. Més concretament, ens centrarem en el disseny i implementació de cinc funcions relacionades amb codis sistemàtics. Aquestes són: `IsInformationSet(C,I)`, `IsSystematic(C)`, `InformationSpace(C)`, `InformationSet(C)` i `AllInformationSets(C)`, on C és un codi no necessàriament lineal i I un conjunt de posicions de coordenades.

El nostre treball està dividit en tres etapes.

1. En la primera etapa del treball ens centrarem en l'estudi i aprenentatge de la teoria de codis. Concretament, estudiarem totes les característiques generals de la teoria de codis i aprendrem a representar un codi d'una forma més eficient, amb la representació del codi a través del kernel i els seus representants de les classes.

2. En la segona etapa del treball farem èmfasi en la secció de la teoria de codis que parla sobre codis sistemàtics i conjunts d'informació. Prepararem les bases per entendre i dissenyar les funcions que volem implementar a la llibreria de Magma. A més, aquí provarem alguns resultats per millorar l'eficiència d'alguns algorismes per implementar.

3. En la tercera etapa del treball desenvoluparem diferents algorismes per les funcions a implementar i els seus tests respectivament, per provar el bon funcionament de cada un d'ells. A més, estudiarem el rendiment dels diferents algorismes en funció de la dimensió del kernel, per així obtenir unes conclusions i acabar amb el projecte.

2 Introducció a la teoria de codis

D'entre tots els codis, els lineals són els més estudiats a causa de la propietat de linealitat, que permet representar-los usant una matriu generadora i així computar els seus paràmetres (longitud, nombre de paraules codi i distància mínima) d'una forma més eficient. A més, ens permeten descriure mètodes eficients per construir nous codis lineals a partir d'antics i poder codificar i descodificar d'una forma més senzilla. D'altra banda, els codis no lineals són menys estudiats per culpa de la seva falta de linealitat, encara que alguns d'aquests siguin millors que altres lineals amb les mateixes característiques.

En aquest capítol exposarem unes definicions i propietats sobre codis lineals que ens ajudaran a entendre el treball. En general, hem utilitzat les referències [5], [6], [7], [8], [9], [10] i [11] per escriure aquesta introducció a la teoria de codis.

2.1 Conceptes bàsics

Definició 1. *Anomenarem paraula a una seqüència de números o dígits. La longitud d'una paraula correspon al nombre de dígits d'aquesta.*

Definició 2. *Definirem un codi C com un conjunt de paraules. Cada paraula pertanyent a aquest codi l'anomenarem paraula codi.*

Siguin a i b dues paraules de longitud n. Les denotem en forma de vector com:

$$a = (a_1, a_2, a_3, \ldots, a_n) \qquad b = (b_1, b_2, b_3, \ldots, b_n)$$

o d'una forma més simplificada:

$$a = a_1a_2a_3 \ldots a_n \qquad b = b_1b_2b_3 \ldots b_n.$$

Definició 3. *Definim la paraula $\mathbf{0}$ com la paraula de longitud n tal que tots els seus dígits són* 0.

Definició 4. *Definim la distància de Hamming entre dues paraules a i b com el nombre de posicions o coordenades en que difereixen. És a dir,*

$$d(a, b) = \#\{i \quad | \quad 1 \leq i \leq n \ , \ a_i \neq b_i\}.$$

Definició 5. *Definim el pes de Hamming d'una paraula a com la distància de Hamming entre a i la paraula $\mathbf{0}$. És a dir,*

$$W(a) = d(a, \mathbf{0}).$$

Aquestes dues últimes definicions ens permeten introduir la distància mínima d_C i el pes mínim W_C d'un codi C que són les que usarem més sovint.

Definició 6. *Donat un codi C definim la distància mínima d_C com:*

$$d_C = \min\{d(a, b) \quad | \quad a \neq b \ , \ a, b \in C\}$$

i el pes mínim W_C com:

$$W_C = \min\{W(a) \quad | \quad a \neq \mathbf{0} \ , \ a \in C\}.$$

Per acabar hem de saber que la capacitat correctora d'errors d'un codi C, que és el nombre d'errors que sóm capaços de corregir per cada n dígits enviats a través d'un canal, es calcula a través de la següent fórmula:

$$e = \left\lfloor \frac{\delta - 1}{2} \right\rfloor.$$

2.2 Exemple de codi detector i codi corrector

En aquesta secció, descriurem alguns exemples de codis correctors i detectors d'errors. Farem esment als més importants que utilitzem diàriament sense adonar-nos. Els més habituals d'utilitzar són els codis detectors d'errors com el codi del DNI, el ISBN, el IBAN, el CCC i el EAN. A las targetes de crèdit i als bitllets d'avions també en podem trobar. Tots aquests codis tenen en comú que estan basats en l'aritmètica modular i que són capaços de detectar errors però no corregir-los. Per més informació sobre aquesta secció es pot consultar [12].

Exemple 1. Anem a veure el codi DNI com està estructurat. Suposem que tenim el següent DNI 34658731P. La lletra del DNI és una redundància per verificar si els dígits són els correctes. Per tant, aplicant l'algorisme pertinent a aquest codi detector d'errors, hauríem de saber calcular la lletra a través dels dígits.

La lletra s'assigna segons el valor que resulta de calcular el número del DNI a $\mathbb{Z}_{23}$, és a dir, segons el residu que s'obté en dividir-lo per 23, d'acord amb les equivalències de la següent taula.

0	1	2	3	4	5	6	7	8	9	10	11
T	R	W	A	G	M	Y	F	P	D	X	B
12	13	14	15	16	17	18	19	20	21	22	
N	J	Z	S	Q	V	H	L	C	K	E	

Taula 1: Assignació de lletres per al DNI

Comprovem-ho:

$$34658731 = 8 \pmod{23} = P$$

El codi detector d'errors del DNI t'assegura detectar si hi ha només un error en un dígit o si dos dígits consecutius s'han intercanviat la posició.

Els codis que estudiarem i desenvoluparem en aquest treball són els correctors d'errors, que seran capaços de detectar l'error i corregir-lo amb una alta probabilitat d'encertar.

Exemple 2. Sigui 123014 la informació que nosaltres volem transmetre a través de dos telèfons mòbils que utilitzen un canal insegur i amb interferències. Un mecanisme per aconseguir una millor fiabilitat podria ser afegir redundància per a cada dígit, és a dir

$$123014 \xrightarrow{codificar} 1111.2222.3333.0000.1111.4444$$

Així si durant el procés d'enviar el missatge sorgís algun error, el més probable és que sigués fàcil de detectar. Si el missatge que ens arriba té algun dígit dels 4 canviats suposarem que només aquell dígit és l'imprecís ja que és el cas més probable. Però com observem a la següent línia aquest procés de codificació no és del tot precís.

$$1111.2222.3333.0000.1111.4444 \xrightarrow{canal} 1111.2222.33\mathbf{5}3.000\mathbf{1}.1\mathbf{2}11.\mathbf{333}4 \xrightarrow{descodificar} 12301\mathbf{3}$$

2.3 Cossos finits

A partir d'ara els codis que considerem estaran definits sobre cossos finits, per això és oportú fer un breu repàs a la part d'àlgebra que pertoca.

Definició 7. *Un grup és una estructura algebraica formada per un conjunt G dotat d'una operació binària interna que està ben definida i compleix les propietats associativa, element neutre i element simètric. A més, si G compleix la propietat commutativa direm que G és un grup abelià.*

Definició 8. *Un anell és una estructura algebraica formada per un grup abelià R amb la suma, el qual les operacions normalment anomenades suma i multiplicació estan ben definides i compleixen les propietats associativa i distributiva de la multiplicació respecte de la suma. A més, si R compleix la propietat commutativa respecte al producte direm que R és un anell commutatiu. Si existeix l'invers per la multiplicació, direm que R és un anell unitari.*

Definició 9. *Un cos és un anell commutatiu i unitari en que tot element no nul és invertible.*

Definició 10. *Un cos finit és un cos definit sobre un conjunt finit d'elements.*

Tots els cossos finits tenen un nombre d'elements $q = p^n$ per algun p primer i algun n enter.

Teorema 1. *Sigui p primer i $n \geq 1$. Aleshores existeix un únic cos finit amb $q = p^n$ elements (tret d'isomorfisme). El denotem per $\mathbb{F}_{p^n}$ o $GF(p^n)$.*

Quan parlem del cos binari i el cos ternari estarem referint-nos a $\mathbb{F}_2 = \{0, 1\}$ i $\mathbb{F}_3 = \{0, 1, 2\}$ respectivament, amb la suma i el producte habituals mòdul 2 i 3.

Definició 11. *Sigui K un cos, definim la característica de K com l'enter positiu n més petit tal que*

$$\underbrace{1_K + \cdots + 1_K}_{n \text{ sumands}} = 0,$$

on 1_K és l'element neutre de la multiplicació. Si no existeix aquest enter positiu n direm que la característica de K és 0. Denotem la característica d'un cos K com $char(K)$.

Exemple 3. Per fer-nos una idea podem pensar en la característica de cossos habituals com $\mathbb{R}$ i $\mathbb{Z}_3$:

$$char(\mathbb{R}) = 0 \quad i \quad char(\mathbb{Z}_3) = 3.$$

Proposició 1. *Sigui K un cos finit amb char(K)=p. Llavors K té p^n elements per algun n enter tal que $n \geq 1$. La implicació al revés també és vàlida.*

Definició 12. *Sigui K un cos, definim $K[X]$ com el conjunt de polinomis sobre K format pels elements:*

$$\sum_{i=0}^{n} a_i X^i = a_0 + a_1 X + a_2 X^2 + \cdots + a_n X^n.$$

Observem que $K[X]$ és un anell commutatiu i unitari.

Definició 13. *Siguin G i G' dos grups, definim un morfisme de grups com l'aplicació $f : G \to G'$ tal que:*

$$f(xy) = f(x)f(y), \quad \text{per tot } x, y \in G.$$

Definició 14. *Definim com a isomorfisme de grups aquell morfisme de grups bijectiu. A més diem que dos grups G i G' són isomorfs i denotem $G \cong G'$ si existeix un isomorfisme de grups $f : G \to G'$.*

Teorema 2. *Suposem que K és un cos finit amb p^n elements. Llavors existeix un polinomi irreductible $f(x)$ de grau n a $\mathbb{Z}_p[x]$ tal que*

$$\mathbb{Z}_p[x]/(f(x)) \cong K.$$

Exemple 4. El cos binari $\mathbb{F}_2$ pot ser construït a partir de $\mathbb{Z}_2[x]/(x)$, que conté dos elements, $\mathbb{F}_2 = \{0, 1\}$.

Exemple 5. El polinomi $f(x) = x^2 + x + 1$ és irreductible a $\mathbb{Z}_2[x]$. Pel teorema anterior podem construir el cos $\mathbb{F}_4$ a través de $\mathbb{Z}_2[x]/(f(x))$. Això vol dir que $\mathbb{F}_4 = \{0, 1, \alpha, 1 + \alpha\}$ on $\alpha = [x]$ és la classe de x mòdul $f(x)$.

2.4 Codis lineals

Sigui $\mathbb{F}_q$ el cos finit $\mathbb{F}_q = GF(q)$, on q és una potència d'un primer p. Considerem un codi C de longitud n amb M paraules codi sobre $\mathbb{F}_q$. A partir d'ara parlarem de codis q-aris per emfatitzar que tenim un codi C sobre el cos finit $\mathbb{F}_q$, i no sobre $\mathbb{F}_2$ com és habitual.

Definició 15. *Un codi q-ari C és anomenat lineal si $\lambda a + \mu b \in C$ per tot $a, b \in C$ i $\lambda, \mu \in \mathbb{F}_q$.*

Definició 16. *Equivalentment, direm que un codi q-ari C de longitud n sobre un cos finit $\mathbb{F}_q$ és lineal si és un subespai vectorial de dimensió k de $\mathbb{F}_q^n$. Denotarem el codi lineal com $C[n, k, d]$ on d és la distància mínima.*

Per tant, per a cada codi lineal $C[n, k, d]$ existeix un morfisme $\phi : \mathbb{F}^k \longrightarrow \mathbb{F}^n$ tal que $\phi(\mathbb{F}^k) = C$.

Exemple 6. Sigui $C_1 = \{000, 111\}$ un codi sobre el cos finit $\mathbb{F}_2$. Observem que,

$$000 + 000 = 000, \;\; 000 + 111 = 111, \;\; 111 + 000 = 111, \;\; 111 + 111 = 000.$$

Per tant C_1 és un codi lineal. Agafem ara el codi $C_2 = \{000, 101, 111\}$ sobre el cos finit $\mathbb{F}_2$. Observem que $101 + 111 = 010$ no pertany a C_2. Per tant C_2 no és un codi lineal.

Definició 17. *Donat un codi lineal $C[n, k, d] \subseteq \mathbb{F}_q^n$, podem definir una matriu generadora de C com $G = \phi^T$. És a dir, la matriu generadora d'un codi lineal és la matriu transposada de la matriu definida per $\phi : \mathbb{F}^k \longrightarrow \mathbb{F}^n$ tal que $\phi(\mathbb{F}^k) = C$. Per definició, observem que k és exactament el nombre de files i n és el de columnes de G.*

Exemple 7. Sigui $C_3 = \{1010, 0111, 1101, 0000\}$ un codi lineal sobre el cos finit $\mathbb{F}_2$. Una matriu generadora de C_3 és:

$$G_3 = \begin{pmatrix} 1010 \\ 0111 \end{pmatrix}.$$

Suposem ara que volem transmetre una seqüència de dígits, per exemple 11011011. Agafem els dígits en blocs de $k = 2$ (11 01 10 11) i els codifiquem usant la matriu generadora G_3.

$$(11) \cdot \begin{pmatrix} 1010 \\ 0111 \end{pmatrix} = (1101)$$

$$(01) \cdot \begin{pmatrix} 1010 \\ 0111 \end{pmatrix} = (0111)$$

$$(10) \cdot \begin{pmatrix} 1010 \\ 0111 \end{pmatrix} = (1010)$$

Llavors els dígits que faríem servir per transmetre la informació serien:

$$1101 \;\; 0111 \;\; 1010 \;\; 1101.$$

Definició 18. *Sigui G la matriu generadora d'un codi lineal C. Si la matriu identitat I_k de dimensió k és una submatriu de G, direm que C és un codi sistemàtic en les coordenades de les columnes de la matriu identitat.*

Definició 19. *Sigui G la matriu generadora d'un codi lineal C, direm que G està en forma estàndard si es pot escriure com $G = (I_k \mid A)$. Llavors el codi C és sistemàtic en les k primeres coordenades.*

Proposició 2. *Tot codi lineal C és sistemàtic.*

Demostració. Anem a demostrar que tot codi lineal C és sempre sistemàtic. Sigui G una matriu generadora de C de rang k, llavors existeixen k columnes on es pot aplicar el mètode de Gauss i així obtenir un nova matriu pel mateix codi que tingui la matriu identitat I_k com a submatriu. Aquestes transformacions no impliquen permutacions en les columnes i per tant, la nova matriu generadora amb I_k de submatriu, genera exactament el mateix codi. Aleshores, C és un codi sistemàtic. □

Si es vol que la identitat I_k estigui en les primeres coordenades, aleshores potser implica una permutació de coordenades i la matriu en forma estàndard no té perquè ser una matriu generadora del codi C original.

L'avantatge d'utilitzar codis amb una matriu generadora en forma estàndard és que quan la informació està codificada, aquesta apareix sencera en les primeres coordenades de la paraula codi. La resta de coordenades diferents de la informació és la part de redundància necessària per a la correcció d'errors.

Exemple 8. Agafem el codi lineal C_3 de l'exemple anterior. Observem que aquest codi lineal té una matriu generadora en forma estàndard, ja que té la matriu identitat en les dues primeres columnes.

$$G = \begin{pmatrix} \mathbf{10}10 \\ \mathbf{01}11 \end{pmatrix}$$

Això permet que en enviar la seqüència 11 01 10 11 aquesta es transmeti com

$$\mathbf{11}01\ \ \mathbf{01}11\ \ \mathbf{10}10\ \ \mathbf{11}01,$$

on els dígits en negreta corresponen a la informació original.

En general, podem dir que usant un codi sistemàtic en forma estàndard, les paraules codi mantenen els k dígits de la informació a les primeres k coordenades, i els dígits restants $n-k$ corresponen als de la redundància.

La importància d'usar codis sistemàtics apareix després de la fase de correcció d'errors, en descodificar el vector rebut, ja que només és necessari eliminar la redundància per obtenir la informació.

Per treballar millor en codis lineals sempre ens interessarà que la seva matriu generadora estigui en forma estàndard, ja que així es reduiran molt els càlculs en el procés de descodificar. Llavors ens formulem la següent pregunta: és possible poder trobar un codi lineal equivalent al nostre inicial amb una matriu generadora en forma estàndard?

Definició 20. *Direm que dos codis lineals C i C', definits respectivament com $\phi : \mathbb{F}^k \longrightarrow \mathbb{F}^n$ i $\phi' : \mathbb{F}^k \longrightarrow \mathbb{F}^n$ són equivalents si existeix un ismorfisme $f : \mathbb{F}^k \longrightarrow \mathbb{F}^k$ i una permutació $\pi : \mathbb{F}^n \longrightarrow \mathbb{F}^n$ tal que $\pi \circ \phi' \circ f = \phi$.*

A la pràctica direm que dos codis lineals diferents són equivalents si permutant les columnes i fent combinacions lineals de les files de G es pot arribar a la matriu G', on G i G' són les seves matrius generadores.

Gràcies a la següent proposició podem respondre al dubte plantejat anteriorment sobre si existia la possibilitat de transformar sempre un codi lineal C a un codi lineal equivalent amb una matriu generadora en forma estàndard.

Proposició 3. *Sigui $C_1[n,k,d]$ un codi q-ari lineal amb matriu generadora G_1. Sempre existeix un codi q-ari lineal $C_2[n,k,d]$ equivalent a C_1 amb matriu generadora G_2 i en forma estàndard.*

Demostració. Si G_1 està en forma estàndard ja estaríem. Suposem llavors que G_1 no està en forma estàndard. Com que G_1 té rang k, mitjançant el mètode de Gauss i permutant les columnes que siguin necessàries, podem aconseguir una matriu esglaonada i per tant una matriu identitat en les k primeres columnes. Obtenim així una matriu generadora $G_2 = (I_k \,|\, A)$ amb A una matriu de k files i $n-k$ columnes. □

Exemple 9. Sigui G una matriu generadora d'un codi lineal $C[6,2,3]$ sobre $\mathbb{F}_3$,

$$G = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 2 & 1 \end{pmatrix}.$$

Busquem primer, efectuant permutacions en les columnes, la matriu generadora G' en forma estàndard d'un codi equivalent.

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 2 & 1 \end{pmatrix} \longrightarrow \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 2 & 1 \end{pmatrix} = G'.$$

Sabem que la matriu G' genera un codi lineal $C'[6, 2, 3]$ sobre $\mathbb{F}_3$ equivalent a C. Observem les paraules codi de C' a la Taula 1.

C'
000000
100101
011021
111122
200202
022012
222211
122110
211220

Taula 2: Paraules codi de C'

Utilitzant la matriu generadora G del codi C, si la informació que volem enviar és 21 i 20, enviarem

$$(2, 1)G \text{ } i \text{ } (2, 0)G \quad \rightarrow \quad 221120 \text{ } i \text{ } 220002.$$

En canvi, si utilitzem la matriu generadora G' del codi C' obtenim

$$(2, 1)G' \text{ } i \text{ } (2, 0)G' \quad \rightarrow \quad \mathbf{21}1220 \text{ } i \text{ } \mathbf{20}0202.$$

Observem que la informació es troba a les dues primeres coordenades de les paraules codi, és a dir, la matriu generadora G' està en forma estàndard.

Definició 21. *Donats $x, y \in \mathbb{F}_q^n$ tals que $x = (x_1, \ldots, x_n)$ i $y = (y_1, \ldots, y_n)$, es defineix el producte escalar com:*

$$<x, y> = \sum_{i=1}^{n} x_i y_i.$$

Definició 22. *Donat un codi lineal C sobre un cos $\mathbb{F}_q$ de longitud n i dimensió k, definim el codi ortogonal com*

$$C^{\perp} = \{v \in \mathbb{F}_q^n \mid \forall a \in C, <v, a> = 0\}.$$

Observem que el codi ortogonal $C^{\perp}$ és un codi lineal per definició, fins i tot si C no és lineal.

Definició 23. *La matriu generadora H del codi lineal ortogonal $C^{\perp}$ és l'anomenada matriu de control del codi lineal C.*

A la pràctica aquesta matriu de control H s'encarrega de verificar si una paraula pertany al codi o no. Per tant, si H és una matriu de control d'un codi lineal C de longitud n sobre $\mathbb{F}_q$, llavors C és format per totes les paraules codi $a \in \mathbb{F}_q^n$ tal que $Ha^T = \mathbf{0}$.

Proposició 4. *Donat un codi lineal amb matriu generadora $G = (I_k \mid P)$, on I_k és la matriu identitat $k \times k$ i P una matriu la $k \times (n - k)$, llavors la matriu de control és $H = (-P^T \mid I_{n-k})$ on P^T és la matriu transposada de P.*

Demostració. Sigui $G = (I_k \mid P)$ la matriu generadora d'un codi lineal C. El codi lineal generat per H és un codi $D[n, n - k, d]$. Obtenim:

$$GH^T = (I_k \mid P_{k \times n-k})(-P_{k \times n-k} \mid I_{n-k})^T = -P + P = 0.$$

Observem que totes les paraules codi de C són ortogonals al codi D que genera H. Per tant H és la matriu generadora de $C^{\perp} = D$. □

Exemple 10. Sigui G una matriu generadora d'un codi lineal C sobre $\mathbb{F}_2$,

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

Busquem primer, permutant la primera columna amb la segona i després la segona amb la cinquena, la matriu generadora G' en forma estàndard d'un codi equivalent.

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

Ara apliquem la proposició anterior per trobar la matriu de control H' del codi lineal amb matriu en forma estàndard equivalent.

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

Per trobar la matriu de control H del nostre codi lineal C hem de desfer les permutacions en les columnes efectuades al primer pas.

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

Observem ara si $a_1 = 01011$ i $a_2 = 11111$ pertanyen al codi C utilitzant la matriu de control H. Comprovem-ho:

$$Ha_1^T = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}.$$

Per tant, com que $Ha_1^T = \mathbf{0}$ tenim que $a_1 \in C$. Ara bé,

$$Ha_2^T = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}.$$

Com que $Ha_2^T \neq \mathbf{0}$, llavors $a_2 \notin C$.

2.5 Codis no lineals

Recordem que en teoria de codis, els codis lineals són els més eficients, estudiats i utilitzats per la seva simplicitat de representació gràcies a que es poden representar a través de la seva matriu generadora. En canvi, els codis no lineals tenen una major dificultat de representació. En aquesta secció explicarem com podem representar els codis no necessàriament lineals mitjançant una unió de classes d'un subcodi lineal anomenat kernel.

Definició 24. *Donat un codi $C \subseteq \mathbb{F}_q^n$ direm que és un codi q-ari no lineal si C no és un subespai vectorial de $\mathbb{F}_q^n$.*

Quan tenim un codi no necessàriament lineal el denotem com $C(n, M, d)$, on n és la longitud, M és la quantitat de paraules codi i d la distància mínima. Si no coneixem la seva distància mínima d, el denotem com $C(n, M)$.

Exemple 11. Siguin $C_1(5, 9)$ i $C_2(5, 9)$ dos codis q-aris sobre el cos finit $\mathbb{F}_3$ formats per la següents paraules donades per la Taula 3.

C_1	C_2
00000	00000
22220	02111
00210	01222
10221	11001
22100	22002
00120	10112
22010	21110
10101	12220
10011	20221

Taula 3: Paraules codi de C_1 i C_2

Com es pot observar, el codi C_2 és lineal. En canvi, C_1 no és lineal ja que $10101 + 10011 = 20112 \notin C_2$.

Anem a veure ara com representar un codi q-ari no necessàriament lineal $C(n, M)$ com una part lineal anomenada kernel i els seus representants de classe.

Definició 25. *Donat un codi $C \subseteq \mathbb{F}_q^n$, definim el kernel com el conjunt de vectors que deixen C invariant al fer translacions, és a dir,*

$$K_C = \{x \in \mathbb{F}_q^n \mid \mu x + C = C, \ \forall \mu \in \mathbb{F}_q\}.$$

Observem que K_C forma un codi lineal. Equivalentment, el kernel també es pot definir com el codi lineal més gran tal que C es pot dividir en classes disjuntes de K_C. A més si C és un codi q-ari lineal, llavors el kernel és igual al codi C, és a dir $K_C = C$.

Un codi q-ari C es pot dividir en classes disjuntes, és a dir, podem representar el nostre codi C com:

$$C = \bigcup_{i=0}^{t} (K_C + c_i) \ \text{ i } \ q^k(t+1) = M$$

on $K_C + c_i$ són les classes i el conjunt $L = \{c_0, c_1, \ldots, c_t\}$ conté els representants de cada classe. A més, k és la dimensió del kernel K_C.

Exemple 12. Sigui $C = \{000, 011, 100, 110, 101, 001\}$ un codi q-ari no lineal sobre $\mathbb{F}_2$ ja que $100 + 110 = 010 \notin C$. Busquem el kernel de C.

$$\begin{aligned}
000 + C &= \{000, 011, 100, 110, 101, 001\} = C \\
011 + C &= \{011, 000, 111, 101, 110, 010\} \neq C \\
100 + C &= \{100, 111, 000, 010, 001, 101\} \neq C \\
110 + C &= \{110, 101, 010, 000, 011, 111\} \neq C \\
101 + C &= \{101, 110, 001, 011, 000, 100\} = C \\
001 + C &= \{001, 010, 101, 111, 100, 000\} \neq C \\
t + 1 &= 6/2 = 3
\end{aligned}$$

Amb aquests càlculs obtenim que $K_C = \{000, 101\}$ és el kernel de C i a més forma un codi lineal. Llavors, per definició, hem de buscar c_0, c_1, c_2 que compleixin la següent equació:

$$C = (\{000, 101\} + c_0) \cup (\{000, 101\} + c_1) \cup (\{000, 101\} + c_2)$$

Finalment obtenim que $c_0 = 000$, $c_1 = 110$ i $c_2 = 001$. Així podem formar el codi no lineal C a través del kernel K_C i els representants c_0, c_1, c_2.

Observem que si tenim un codi q-ari C amb $\mathbf{0} \in C$, llavors $\mathbf{0} \in K_C$ i podem escollir el primer representant com $c_0 = \mathbf{0}$.

Proposició 5. *Sigui C un codi q-ari. Si $\mathbf{0} \notin C$, tenim que $K_C \cap C = \emptyset$.*

Demostració. Suposem que existeix $c_1 \in C$ tal que $c_1 \in K_C$. Tenim que $c_1 + C = C$. Per tant, existeix un $c_2 \in C$ tal que $c_1 + c_2$. Llavors $c_2 = \mathbf{0}$ arribant a contradicció. □

Proposició 6. *Sigui $C(n, M)$ un codi q-ari no lineal on $q = 2$ i $\mathbf{0} \in C$. Llavors $t \geq 2$. Recordem que $t + 1$ és el nombre de representants de les classes de C.*

Demostració. Quan tenim $q = 2$, obtenim un codi binari C on el kernel està definit com $K_C = \{x \in \mathbb{F}_2^n \mid x + C = C\}$. Per tant, si $t = 1$ aleshores $C = K_C \cup (K_C + c_1)$. Observem ara:

- Per tot $k_1, k_2 \in K_C$ tenim $k_1 + k_2 \in K_C$.
- Per tot $k \in K_c$ i $x \in K_c + c_1$ tenim $k + x = k + (k_1 + c_1) = (k + k_1) + c_1 \in K_C + c_1$ per $k_1 \in K_C$.
- Per tot $x, y \in K_C + c_1$ tenim $x + y = k_1 + c_1 + k_2 + c_1 = k_1 + k_2 \in K_C$ per $k_1, k_2 \in K_C$.

Finalment, per tota parella d'elements $x, y \in C$ tenim que $x + y \in C$, i per tant C seria lineal i $K_C = C$. Per tant és necessari que $t \geq 2$. □

En aquesta secció, hem vist que podem expressar un codi q-ari utilitzant el kernel i els representants de les classes de K_C en C. Això ens permet emmagatzemar d'una forma més compacta i òptima el nostre codi C. En general si $C[n, k]$ és un codi q-ari lineal, necessitem emmagatzemar k vectors de longitud n d'una matriu generadora del codi. Si $C(n, M)$ és un codi q-ari no lineal, necessitem emmagatzemar M vectors de longitud n, en comptes de $log_q(M)$ com el cas lineal. En canvi, si s'utilitza la representació del kernel i els representants de classe, podem emmagatzemar el codi només amb k vectors d'una base del kernel i els $t + 1$ representants de les classes.

Exemple 13. Sigui $C(17, 3^6)$ un codi sobre $\mathbb{F}_3$, necessitem $17 \cdot 3^6 = 12393$ bits per guardar totes les 3^6 paraules codi. Si C no és lineal amb un kernel amb dimensió 4, llavors seríem capaços de guardar les paraules codi fent servir 4 paraules codi i $t + 1 = M/q^k = 3^6/3^4 = 3^2$ representants de classe. Així, només farem servir $(4 + 3^2) \cdot 17 = 221$ bits, en comptes de 12393.

3 Codis sistemàtics i conjunts d'informació

En l'anterior secció hem fet un repàs general de la teoria de cossos finits i hem introduït la teoria de codis sobre cossos finits en termes generals. Ara ens centrarem en alguns detalls no aprofundits anteriorment. La intenció d'aquesta secció és aprendre la definició d'un codi q-ari C sistemàtic quan el codi no és necessàriament lineal.

En aquest capítol exposarem unes definicions i propietats sobre codis sistemàtics, que ens ajudaran a entendre les implementacions de les funcions de les quals després parlarem. En general, hem utilitzat les referències [4], [13] i [14].

Recordem la definició de sistemàtic i forma estàndard quan tenim un codi q-ari lineal C.

Definició 26. *Sigui G la matriu generadora d'un codi lineal C. Si la matriu identitat I_k de dimensió k és una submatriu de G, direm que C és un codi sistemàtic en les coordenades de les columnes de la matriu identitat.*

Definició 27. *Sigui G la matriu generadora d'un codi lineal C, direm que G està en forma estàndard si es pot escriure com $G = (I_k \mid A)$. Llavors el codi C és sistemàtic en les k primeres coordenades.*

Observem que aquestes dues definicions no són aplicables a codis q-aris no lineals al no tenir una matriu generadora. Per tant, hem de donar una definició de codi sistemàtic més general, tant per codis lineals com no lineals. Per això necessitem definir abans que és una codificació sistemàtica.

Definició 28. *Sigui $C(n, M, d)$ un codi q-ari no necessàriament lineal amb una codificació que assigna el missatge $u_1, \ldots, u_k$ a la paraula codi $c_1, \ldots, c_n$. Aquesta codificació s'anomena sistemàtica si existeixen coordenades $i_1, \ldots, i_k$ tal que $u_1 = c_{i_1}, u_2 = c_{i_2}, \ldots, u_k = c_{i_k}$.*

Exemple 1. Tenim ara la següent codificació:

missatge	paraula codi
00	000
01	010
10	101
11	111

Taula 4: Exemple d'una codificació sistemàtica

Observem que agafant $i_1 = 0$ i $i_2 = 1$ tenim una codificació sistemàtica.
En canvi, amb aquesta codificació:

missatge	paraula codi
00	000
01	111
10	101
11	010

Taula 5: Exemple d'una codificació no sistemàtica

Observem que no és una codificació sistemàtica al no trobar-se el missatge en les paraules codi.

Anem a generalitzar la noció de codificació sistemàtica per codis no lineals. Necessitarem parlar abans sobre que és un conjunt d'informació per un codi q-ari C.

Definició 29. *Un conjunt $I \subseteq \{1, \ldots, n\}$ de nombres enters és un conjunt d'informació per un codi q-ari $C(n, M)$ si $M = \#C_I$, on $C_I = \{v_I : v \in C\}$ i v_I és la restricció del vector v en les coordenades de I.*

Com ja sabem, podem representar un codi q-ari no lineal amb una base del kernel i els seus representants de classe. Si el codi C està emmagatzemat fent servir el kernel i els representants de classe, aleshores podem calcular d'una forma més directa si una seqüència I és un conjunt d'informació.

Definició 30. *Un conjunt $I \subseteq \{1, \ldots, n\}$ és un conjunt d'informació per un codi q-ari $C(n, M)$ si $M = \#C_I$, on $C_I = \{k_I + v_I : v \in L \, i \, k \in K_C\}$ i v_I, k_I són la restricció dels vectors v, k respectivament en les coordenades de I. Recordem que L és el conjunt de representants de classe i K_C el kernel de C.*

Ara que ja tenim la definició de conjunt d'informació, podem definir quan un codi q-ari C no necessàriament lineal és sistemàtic.

Definició 31. *Sigui C un codi q-ari no lineal, direm que C és sistemàtic si existeix un conjunt d'informació I pel codi C.*

Observem que coincideix amb la definició de codi sistemàtic si C és un codi lineal, ja que com que $M = \#C_I$, la matriu generadora de C es pot triar tal que tingui la matriu identitat en les columnes corresponent al conjunt I.

Exemple 2. Sigui $C(5, 9)$ un codi q-ari sobre un cos $\mathbb{F}_3$.

C
00000
22220
00210
10221
22100
00120
22010
10101
10011

Taula 6: Paraules codi de C

Observem que el codi C és no lineal ja que $10101 + 10011 = 20112 \notin C$. Anem a veure si C és sistemàtic i per això hem de buscar si existeix algun conjunt d'informació per C. Calculem primer quantes coordenades té un possible conjunt d'informació I:

$$q^t = \#C \iff 3^t = 9 \iff t = 2.$$

Per trobar un possible conjunt d'informació s'ha de complir que $M = q^t = \#C_I$, on $C_I = \{v_I : v \in C\}$ i v_I és la restricció del vector v en les coordenades de I. Per trobar I fem subconjunts del conjunt $\{1, \ldots, n\}$ de 2 elements. Provem $I_1 = \{1, 2\}$ i observem si $\#C_{I_1} = 9$.

$$C_{I_1} = \{00, 22, 00, 10, 22, 00, 22, 10, 10\} = \{00, 22, 10\}.$$

Observem que $\#C_{I_1} = 3$. Per tant I_1 no és un conjunt d'informació. Provem ara $I_2 = \{1, 3\}$.

$$C_{I_2} = \{00, 22, 02, 12, 21, 01, 20, 11, 10\}.$$

Observem que $\#C_{I_2} = 9$. Per tant I_2 és un conjunt d'informació i C és un codi sistemàtic.

Exemple 3. Sigui $C(3, 4)$ un codi q-ari sobre un cos $\mathbb{F}_2$ tal que $C = \{000, 100, 010, 001\}$. Observem que el codi C és no lineal ja que $100 + 010 = 110 \notin C$. Anem a veure si C és sistemàtic. Calculem primer quantes coordenades té un possible conjunt d'informació I:

$$q^t = \#C \iff 2^t = 9 \iff t = 2.$$

Per trobar un possible conjunt d'informació s'ha de complir que $M = q^t = \#C_I$, on $C_I = \{v_I : v \in C\}$ i v_I és la restricció del vector v en les coordenades de I. Per trobar I fem subconjunts del

conjunt $\{1,\ldots,n\}$ de 2 elements. Provem per tots els possibles conjunts d'informació: $I_1=\{1,2\}$, $I_1=\{1,3\}$ i $I_1=\{2,3\}$.

$$C_{I_1}=\{00,10,01,00\}=\{00,10,01\}.$$

$$C_{I_2}=\{00,10,00,01\}=\{00,10,01\}.$$

$$C_{I_3}=\{00,00,10,01\}=\{00,10,01\}.$$

Observem que $\#C_{I_1}=\#C_{I_2}=\#C_{I_3}=3\neq\#C$. Per tant, no existeix cap conjunt d'informació per C i aleshores podem afirmar que C no és un codi sistemàtic.

Podem descriure un algorisme per saber si un codi q-ari C és sistemàtic, utilitzant el següent resultat.

Proposició 7. *Sigui C un codi q-ari de longitud n amb cardinal q^k i amb kernel K_C de dimensió κ. Sigui I_κ un conjunt d'informació per K_C. Llavors existeix un conjunt de representants $c_0,\ldots,c_t$ tal que $(c_i)_{I_\kappa}=0$ per tot $i\in\{0,1,\ldots,t\}$, on $t=q^{k-\kappa}-1$. A més, $I=I_\kappa\cup J$, where $J\subseteq\{1,\ldots,n\}\backslash I_\kappa$ and $|J|=k-\kappa$, és un conjunt d'informació de C si i només si $(c_i)_J\neq(c_j)_J$ per tot $i,j\in\{0,1,\ldots,t\}$, $i\neq j$* [14].

Demostració. Com que I_κ és un conjunt d'informació per K_C, podem considerar la matriu generadora G de K_C en forma estàndard tenint la matriu identitat Id_κ a les columnes corresponents al conjunt I_κ. Usant la matriu generadora, sempre podem canviar els representants de classe de C per uns nous representants de classe $c_0,c_1,\ldots,c_t$ tals que $(c_i)_{I_\kappa}=\mathbf{0}$ per tot $i\in\{0,1,\ldots,t\}$. Com que la cardinalitat de C és q^k i K_C té dimensió κ, el nombre de representants de classe és $t+1=q^{k-\kappa}$. Sigui J un subconjunt de $k-\kappa$ coordenades de posició de $\{1,\ldots,n\}\backslash I_\kappa$, sigui $I=I_\kappa\cup J$, i $B=\{(c_i)_J\mid i\in\{0,1,\ldots,t\}\}$. Llavors, si I és un conjunt d'informació per C, necessitem provar que $B=\mathbb{F}_q^{k-\kappa}$. Assumint que $B\neq\mathbb{F}_q^{k-\kappa}$, tenim $i,j\in\{0,1,\ldots,t\}$, $i\neq j$, tal que $(c_i)_J=(c_j)_J$. Per tant $(K_C+c_i)_I=(K_C+c_j)_I$, ja que $(c_i)_{I_\kappa}=(c_j)_{I_\kappa}=\mathbf{0}$. En conseqüència, I no és un conjunt d'informació, obtenint una contradicció. Recíprocament, si $B=\mathbb{F}_q^{k-\kappa}$, llavors $(K_C+c_i)_I\cap(K_C+c_j)_I=\emptyset$ per tot $i,j\in\{0,1,\ldots,t\}$, $i\neq j$, ja que $(c_i)_{I_\kappa}=(c_j)_{I_\kappa}=\mathbf{0}$. Per tant, I és un conjunt d'informació per C. □

Mitjançant la Proposició 7, podem definir un algorisme per comprovar si un conjunt de k nombres enters forma un conjunt d'informació per a un codi q-ari C de longitud n i de cardinal q^k amb el kernel K_C, de manera que també es comprova si C és sistemàtic o no.

4 Software matemàtic per a codis en Magma

4.1 Introducció a Magma

Magma és un gran paquet de software dedicat a computar operacions i càlculs d'àlgebra, geometria, teoria de grups i combinatòria algebraica [18]. Magma es diferencia d'altres softwares ja que:

- Té un llenguatge de programació dissenyat pel seu propi propòsit.
- Té un interprete que permet càlculs interactius.
- Té un extens sistema d'ajuda i documentació.
- Proporciona algorismes i implementacions altament eficients.
- Proporciona un ambient matemàtic rigorós.
- Té un interprete que permet càlculs interactius.

Existeixen altres software matemàtics gratuïts que treballen amb codis lineals i no lineals. Aquests són GAP i SAGE. En el cas del software GAP [15], no és tant eficient en temps i memòria, ja que no emmagatzema les paraules codi en l'estructura kernel i els seus representants de classe. Aleshores, no és una opció vàlida pel nostre projecte. L'altre software és el SAGE [16], que és un software molt potent i eficient, però només té una llibreria per codis lineals.

Magma comparteix característiques amb els dos softwares ja mencionats: té una llibreria ja implementada per codis lineals i a més, té en procés una llibreria per codis no lineals [4] que utilitza l'estructura kernel i els seus representants de classe per la representació d'un codi. Per tant, l'opció més viable per treballar en aquest projecte és el software Magma.

En general, hem consultat informació en [6], [13] i [17].

4.2 Objectius del projecte

L'objectiu principal d'aquest projecte és contribuir en el desenvolupament de la llibreria mencionada de Magma que treballa per codis q-aris no lineals d'una forma eficient [4] i amb la representació de kernel i els seus representants de classe. Concretament, hem desenvolupat cinc funcions relacionades amb la teoria donada de codis sistemàtics. En general, hem dissenyat diferents algorismes per cada funció, hem provat resultats teòrics per millorar l'eficiència d'alguns algorismes, hem creat alguns tests per provar les funcions i finalment, hem fet test de rendiment de tots els algorismes plantejats.

Hem dividit el projecte en els següents objectius:

1. Estudiar les definicions i propietats bàsiques sobre teoria de codis i més en concret, sobre codis q-aris sistemàtics i conjunts d'informació.
2. Aprendre en general el llenguatge de programació del software Magma, centrant-nos en les funcions que inclouen codis lineals i en les funcions de codis no lineals ja implementades en la llibreria que estem ampliant.
3. Escollir codis q-aris adequats tals que s'adaptin i permetin dissenyar un conjunt de tests per validar les noves funcions.
4. Implementar diferents algorismes per les funcions per codis no necessàriament lineals sobre codis sistemàtics, aprofitant, en la mesura que es pugui, funcions ja implementades a Magma.
5. Validar tots els algorismes amb els diferents codis escollits i tests de caixa negra.
6. Fer un estudi de rendiment, per determinar quin algorisme és més òptim, en la majoria dels casos, per cada funció.

4.3 Funcions de Magma

Abans d'entrar a parlar sobre les noves funcions a implementar, anem a entendre per sobre el llenguatge de programació amb el qual treballarem i entrarem en detall en algunes funcions sobre codis lineals (ja implementades en la versió actual de Magma) que més tard utilitzarem. Per aquesta secció hem utilitzat informació de [18].

4.3.1 Accions simples i expressions

- Per assignar variables escrivim: `nom variable := valor variable;`.
- Per mostrar per pantalla escrivim: `print "text";`.
- Alguns operadors bàsics són:

`lt, gt` $(<,>)$ | `eq, ne` $(=,\neq)$ | `ge, le` $(\geq,\leq)$ | `in, notin` $(\in,\notin)$ | `subset` $(\subset)$

4.3.2 Composició d'acccions

- Per utilitzar una composició condicional escrivim:

```
if ..  then ..  elif ..  then ..  else ..  end if;.
```

- Per utilitzar una composició iterativa tenim diverses formes. Alguns exemples són:

```
for i:= ..  to ..  by ..  do ..  end for;

while ..  do ..  end while;.
```

4.3.3 Conjunts, seqüències i tuples

- Tant els conjunts com les seqüències són col·leccions d'objectes pertanyents a una mateixa estructura algebraica. Un conjunt no és ordenat i no existeix repetició d'elements. Una seqüència, en canvi, és ordenada i existeix la repetició d'elements. Els conjunts s'escriuen entre $\{\}$. Les seqüències s'escriuen entre [ ].
- Algunes operacions bàsiques són: `C join D`, `C meet D` i `C diff D` per conjunts i `S cat T` per seqüències.
- Per col·leccionar objectes de diferent tipus junts, Magma té les tuples dels productes cartesians. Aquestes tenen la forma `< a1, a2, ...  , at >`.
- Tenim la comanda `#(C)` que ens dona el cardinal de C. Funciona tant per seqüències, com conjunts, com tuples i codis.
- Tenim la comanda `Sort(S)` per ordenar seqüències.
- Tenim la comanda `Append(S,x)` per afegir x al final de S per seqüències i tuples.

4.3.4 Funcions i procediments

- Es pot definir funcions simples que donats n arguments retornin un valor.

```
f:= func < x1, ..  , xn | ..  >;.
```

- La declaració general d'una funció de n arguments per la que s'hagi de fer una serie d'operacions és:

`f:= function (x1, .., xn)` *operacions* `return ..; end function;.`

- Les funcions que no retornen cap valor s'anomenen procediments. Per a escriure procediments tenim les següents formes:

```
f:= proc < x1, ..  , xn | ..  >;
```

```
f:= procedure (x1, .., xn) ..  end procedure;.
```

- Perquè una de les variables que donem d'entrada a un procediment es modifiqui dins el procediment l'hem de passar per referencia. Això es fa escrivint $\sim$ davant de la variable.

4.3.5 Àlgebra lineal

- Per crear cossos finits de q o p^n elements escrivim `K := GF(q)` o bé `K := GF(p,n)`.
- Donat un cos K, denotem el conjunt de vectors de K^n per `VectorSpace(K,n)`. Per crear un vector utilitzarem l'operador de coerció !.
- Donat un anell R, denotem el conjunt de matrius quadrades $n \times n$ per `MatrixRingMatrixRing(R,n)` i el conjunt de les matrius $m \times n$ per `KMatrixSpace(R,m,n)`.
- Algunes operacions bàsiques amb matrius són: `Rank(A)`, `Determinant(A)` i `Trace(A)`.

4.3.6 Codis lineals

- Per la construcció d'un codi lineal hi ha diverses opcions [18]. La forma més senzilla és construir un codi lineal a partir de la seva matriu generadora G amb la comanda `LinearCode(G)`.
- També es poden construir diversos codis trivials a partir d'un anell R i la longitud n del codi. Alguns exemples són: `ZeroCode(R, n)` i `UniverseCode(R, n)`.
- Una comanda que usarem sovint és `Set(C)` que serveix per trobar totes les paraules codi d'un codi C.
- Tenim la comanda `IsInformationSet(C,I)` que comprova si la seqüència I donada és un conjunt d'informació pel codi lineal C.
- També tenim la comanda `AllInformationSets(C)` per trobar tots els conjunts d'informació d'un codi lineal C.

4.4 Funcions implementades

Recordem que aquest projecte es basa en la implementació d'una llibreria per codis no lineals en la qual nosaltres hem desenvolupat cinc funcions. Aquestes són: `IsSystematic(C)`, `IsInformationSet(C,I)`, `IsInformationSpace(C)`, `InformationSet(C)` i `AllInformationSets(C)`. Anem a veure una breu definició per a cada una de les funcions treballades [4].

`IsInformationSet(C,I)`

Retorna *true* si i només si la seqüència $I \subset \{1, \ldots, n\}$ de coordenades és un conjunt d'informació pel codi q-ari C de longitud n. El conjunt I pot ser donat com una seqüència d'enters o com una tupla $< I_k, I_r >$ de dues seqüències d'enters. En l'últim cas, la funció retorna *true* si i només si I_k és un conjunt d'informació del kernel K_C i $I = I_k \cup I_r$ és un conjunt d'informació pel codi C.

`IsSystematic(C)`

Retorna *true* si i només si el codi q-ari C de longitud n és sistemàtic. La funció també retorna un conjunt d'informació I, com una seqüència d'enters; i una tupla $< I_k, I_r >$ de dues seqüències d'enters tals que I_k és un conjunt d'informació del kernel K_C i $I = I_k \cup I_r$.

InformationSpace(C)

Donat un codi q-ari C sistemàtic de cardinalitat q^t, retorna l'espai vectorial $U = \mathbb{F}_q^t$, on U és l'espai de vectors d'informació del codi C.

InformationSet(C)

Donat un codi q-ari C sistemàtic de longitud n, retorna un conjunt d'informació I de C. La funció també retorna una tupla $< I_k, I_r >$ de dues seqüències d'enters tals que I_k és un conjunt d'informació del kernel K_C i $I = I_k \cup I_r$ és un conjunt d'informació pel codi C.

AllInformationSets(C)

Donat un codi q-ari C sistemàtic de longitud n, retorna tots els possibles conjunts d'informació de C com una seqüència de seqüències ordenades.

4.5 Algorismes

Cada funció implementada pot contenir dos tipus d'algorismes: KC y BF.

- L'algorisme BF executa les funcions calculant totes les paraules codi o comprovant totes les opcions possibles. És l'anomenat algorisme $BruteForce$. En general és un algorisme lent i poc òptim.
- L'algorisme KC executa les funcions fent servir alguns resultats teòrics i la representació del codi a través del kernel i els seus representants de les classes. És a dir, en la majoria de casos la funció no calcularà totes les paraules codi o no totes les opcions possibles. És l'anomenat algorisme $KernelCosets$.

Recordem que la finalitat d'aquest projecte és implementar d'una forma òptima i eficient les funcions ja proposades. Per això és necessari un estudi dels dos models d'algorismes KC i BF fent diferents tests de rendiment per veure en quins casos, quin model s'executa més ràpid. En les següents seccions entrarem més en detall en cada funció i veurem tots els models amb diferents algorismes proposats.

4.5.1 Funció IsInformationSet(C,I)

Primer de tot destacar que per alleugerir alguns càlculs i ampliar la llibreria de Magma per codis lineals hem millorat la funció `IsInformationSet(C,I)` que actua només quan C és lineal. Només hem fet una funció que retorna *true* si el rang de la matriu generadora de C restringida a les columnes corresponents als valors de I és igual a la dimensió del codi donat.

Recordem ara que segons la definició que hem donat a la funció, el conjunt d'informació I pot ser donat com una seqüència d'enters o com una tupla.

Pel cas que I sigui donat com una seqüencia d'enters hem creat tres algorismes diferents.

IsInformationSetV3(C, I)

Aquest algorisme *versió 3* es tracta d'un $BruteForce$ que es realitza restringint totes les paraules del codi utilitzant la funció de `Set(I)` en les coordenades que dona I.

IsInformationSetV2(C, I)

Aquest algorisme *versió 2* es tracta d'una versió intermitja entre $BruteForce$ i $KernelCosets$ ja que es realitza restringint totes les paraules del codi en les coordenades que dona I però obtenint aquestes paraules codi utilitzant el kernel i els seus representants.

IsInformationSetV1(C, I)

Aquest algorisme *versió 1* es tracta d'un $KernelCosets$ que agafarà tots els conjunts d'informació del kernel que es trobin dins de la seqüència I i utilitzarà la teoria i demostració explicada a la Proposició 7 per comprovar si I és un conjunt d'informació. Destacar que si la dimensió del kernel és 0, llavors seria equivalent a fer servir el segon algorisme $BruteForce$.

Pel cas que I sigui donat com una tupla d'enters $< I_k, I_r >$ de dues seqüències d'enters, hem creat dos algorismes diferents.

IsInformationSetV2(C, I)

Aquest algorisme *versió 2* comprovarà si la seqüència I_k és un conjunt d'informació pel kernel utilitzant la funció IsInformationSet per codis lineals i després comprovarà si la unió $I = I_k \cup I_r$ és un conjunt d'informació utilitzant l'algorisme *versió 1* anteriorment descrit.

IsInformationSetV1(C, I)

Aquest algorisme *versió 1* es tracta d'un $KernelCosets$ que després de comprovar que amb el I_k donat es pugui diagonalitzar, i s'utilitzarà teoria i demostració explicada a la Proposició 7 per comprovar si $I = I_k \cup I_r$ forma un conjunt d'informació. En certa forma és molt semblant al primer algorisme però hauria de ser més òptim al no cridar altres funcions i fer només els càlculs necessaris. Destacar que si la dimensió del kernel és 0, llavors seria equivalent a fer servir un $BruteForce$ si la seqüència $I = I_r$ és un conjunt d'informació.

4.5.2 Funció IsSystematic(C)

Implementar d'una forma òptima la funció la qual tracta aquesta secció és la finalitat principal d'aquest projecte. Degut a aquest fet es van crear el màxim d'algorismes diferents per poder-los comparar entre ells i extreure més resultats. Aquests són:

IsSystematicV6(C)

Aquest algorisme *versió 6* es tracta d'una versió intermitja entre $BruteForce$ i $KernelCosets$ i es basa primer en buscar subconjunts I_k de mida k de $\{1, \ldots, n\}$ comprovant si aquests són conjunts d'informació pel kernel, on n és la longitud de les paraules codi de C i k la dimensió del kernel. Després es busca un subconjunt I_r de mida $r = t - k$, amb t el nombre natural tal que $\#C = q^t$, de $\{1, \ldots, n\}$, disjunt de I_k tal que $I_k \cup I_r$ és un conjunt d'informació (utilizant l'algorisme `InformationSetV3(C,I)`). En el moment que es troba un cas, es para el programa i es retorna $true$.

IsSystematicV5(C)

Aquest algorisme *versió 5* es tracta d'una versió intermitja entre $BruteForce$ i $KernelCosets$ i es basa primer en buscar subconjunts I_k de mida k de $\{1, \ldots, n\}$ comprovant si aquests són conjunts d'informació pel kernel, on n és la longitud de les paraules codi de C i k la dimensió del kernel. Després es busca un subconjunt I_r de mida $r = t - k$, amb t el nombre natural tal que $\#C = q^t$, de $\{1, \ldots, n\}$, disjunt de I_k tal que $I_k \cup I_r$ és un conjunt d'informació (utilizant l'algorisme `InformationSetV2(C,I)`). En el moment que es troba un cas, es para el programa i es retorna $true$.

IsSystematicV4(C)

Aquest algorisme *versió 4* es tracta d'una versió intermitja entre $BruteForce$ i $KernelCosets$ i es basa primer en buscar subconjunts I_k de mida k de $\{1, \ldots, n\}$ comprovant si aquests són conjunts d'informació pel kernel, on n és la longitud de les paraules codi de C i k la dimensió del kernel. Després es busca un subconjunt I_r de mida $r = t - k$, amb t el nombre natural tal que $\#C = q^t$, de $\{1, \ldots, n\}$, disjunt de I_k tal que $I_k \cup I_r$ és un conjunt d'informació (utilizant l'algorisme `InformationSetV1(C,I)`). En el moment que es troba un cas, es para el programa i es retorna $true$.

IsSystematicV3(C)

Aquest algorisme *versió 3* es tracta d'un $BruteForce$ i busca si existeix algun conjunt d'informació I comprovant tots els subconjunts del conjunt $\{1, \ldots, n\}$ de t elements on n és la longitud de les paraules del codi C i t el nombre natural tal que $\#C = q^t$. Calcularà si el subconjunt I és un conjunt d'informació restringint totes les paraules del codi en les coordenades que dona I però obtenint aquestes paraules codi utilitzant el kernel i els seus representants.

IsSystematicV2(C)

Aquest algorisme *versió 2* es tracta d'un $BruteForce$ i busca si existeix algun conjunt d'informació I comprovant tots els subconjunts del conjunt $\{1, \ldots, n\}$ de t elements on n és la longitud de les paraules del codi C i t el nombre natural tal que $\#C = q^t$. Calcularà si el subconjunt I és un conjunt d'informació restringint totes les paraules del codi utilitzant la funció de `Set(I)` en les coordenades que dona I. En el moment que es troba un I que sigui conjunt d'informació es para el programa i es retorna $true$.

IsSystematicV1(C)

Aquest algorisme *versió 1* es tracta d'un $KernelCosets$ i es basa en agafar conjunts d'informació I_k del kernel amb la funció `AllInformationSets(C‘Kernel)` ja implementada a Magma. Després es busca un subconjunt I_r de mida $r = t - k$, amb t el nombre natural tal que $\#C = q^t$, de $\{1, \ldots, n\}$ disjunt de I_k i utilitzant la teoria i demostració explicada a la Proposició 7 per comprovar si la tupla forma un conjunt d'informació. Si alguna d'aquestes tuples forma un conjunt d'informació es para el programa i es retorna $true$. Destacar que té moltes similituds amb l'algorisme *versió 4*, però aquest podria ser més òptim al no cridar la funció `InformationSet(C,I)` i fer només els càlculs necessaris pel resultat buscat.

4.5.3 Funció InformationSpace(C)

L'algorisme implementat per aquesta funció és molt senzill. En aquest cas no es tracta ni d'un $BruteForce$ ni d'un $KernelCosets$, només es basa en l'ús d'una de les funcions de Magma. La funció retorna `VectorSpace(GF(q), t)` d'un codi q-ari C de cardinal q^t. Aquesta funció no comprova si el codi donat és sistemàtic.

4.5.4 Funció InformationSet(C)

L'algorisme implementat per aquesta funció no és més que el més òptim de la funció `IsSystematic(C)`. Com que primer es calcula si el codi C és sistemàtic, s'aprofiten els càlculs per trobar un conjunt d'informació i una tupla.

4.5.5 Funció AllInformationSets(C)

La funció que tracta aquesta secció és la més complicada d'avaluar, ja que el fet de buscar tots els conjunts d'informació d'un codi C és una feina feixuga. Introduirem tres algorismes diferents:

AllInformationSetsV4(C)

Aquest algorisme *versió 4* es tracta d'un $BruteForce$ i busca tots els conjunts d'informació I fent els subconjunts del conjunt $\{1, \ldots, n\}$ de t elements on n és la longitud de les paraules del codi C i t el nombre natural tal que $\#C = q^t$. Calcularà si el subconjunt I és un conjunt d'informació restringint totes les paraules del codi utilitzant la funció `Set(I)` en les coordenades que dona I.

AllInformationSetsV3(C)

Aquest algorisme *versió 3* es tracta d'un $BruteForce$ i busca tots els conjunts d'informació I fent els subconjunts del conjunt $\{1, \ldots, n\}$ de t elements on n és la longitud de les paraules del codi C i t el nombre natural tal que $\#C = q^t$. Calcularà si el subconjunt I és un conjunt

d'informació restringint totes les paraules del codi en les coordenades que dona I i obtenint aquestes paraules codi a través del kernel i els seus representants.

AllInformationSetsV2(C)

Aquest algorisme *versió 2* es tracta d'un $KernelCosets$ i busca tots els conjunts d'informació I fent els subconjunts del conjunt $\{1, \ldots, n\}$ de t elements on n és la longitud de les paraules del codi C i t el nombre natural tal que $\#C = q^t$ i comprovant-ho per la funció més òptima de `IsInformationSet(C,I)`.

AllInformationSetsV1(C)

Aquest algorisme *versió 1* es tracta d'un $KernelCosets$ i es basa en agafar tots els conjunts d'informació I_k del kernel amb la funció `AllInformationSets(C'Kernel)` ja implementada a Magma. Després es busca un subconjunt I_r de mida $r = t - k$, amb t el nombre natural tal que $\#C = q^t$, de $\{1, \ldots, n\}$ disjunt de I_k i utilitzant la teoria i demostració explicada a la Proposició 7 per comprovar si la tupla forma un conjunt d'informació. En el cas que la dimensió del kernel és 0, seria equivalent a fer servir l'algorisme $BruteForce$. Destacar que en general té moltes similituds amb l'algorisme anterior però aquest podria de ser més òptim al no cridar altres funcions i fer només els càlculs necessaris.

4.6 Tests de caixa negra

Un cop implementades les funcions, han de passar uns tests anomenats tests de caixa negra o BB ($BlackBox$). Aquests tests són els encarregats de validar si les funcions implementades són correctes i de trobar possibles errors que puguin portar.

Hem realitzat un total de 13 tests per les funcions que volem implementar. En cada un d'aquests tests hem testejat les funcions per un codi determinat. Per fer una bona comprovació de les funcions hem hagut de buscar 13 codis q-aris diferents i amb propietats variades. En general, hem volgut provar tests de diferents mides, cardinalitat, linealitat, dimensió del kernel i nombre de paraules codi. A la Taula 7 podem veure les propietats de cada codi testejat.

		GF(q)	n	kerDim	#C	Linear	Systematic	$0 \in C$
C_1	UniverseCode(GF(3), 10)	GF(2)	10	10	59049	sí	sí	sí
C_2	RepetitionCode(GF(4), 12)	GF(2)	12	1	4	sí	sí	sí
C_3	ZeroCode(GF(8), 13)	GF(8)	13	0	1	sí	sí	sí
C_4	Cq3n13ker8c_seq	GF(3)	13	8	59049	no	sí	sí
C_5	Cq3n13ker9a_seq	GF(3)	13	9	59049	no	sí	sí
C_6	CHadq4n16ker1b_seq	GF(4)	16	1	64	no	sí	sí
C_7	Cq4n16ker0_seq	GF(4)	16	0	16	no	no	sí
C_8	Cq8n20ker3_seq	GF(8)	20	3	2560	no	no	sí
C_9	CHadq4n16ker3_seq	GF(4)	16	3	64	sí	sí	sí
C_{10}	Cq2n20ker0_seq	GF(2)	20	0	127	no	no	no
C_{11}	CHamq2n16ker4t_seq	GF(2)	16	4	2048	no	sí	no
C_{12}	QaryCodeCker0q2	GF(2)	22	0	10	no	no	sí
C_{13}	CyclicCodeCker10q2	GF(2)	10	10	1024	sí	sí	sí

Taula 7: Codis q-aris per l'estudi dels tests de caixa negra

A continuació mostrarem un test de caixa negra utilitzant el codi trivial Univers per testejar totes les funcions.

```
print "test 1: Universe code of length 10 over GF(3)";
universeCode := UniverseCode(GF(3), 10);
C := QaryCode(universeCode);

I1 := [1..Length(C)];
I2 := [1..Length(C)] cat [3,4];
I3 := [1,2,6];
I4 := [1,2,7,7];

expectedOutputAllInformationSets := [ [1..10] ];

expectedOutputIsSystematic := true;
expectedOutputIsInformationSet1 := true;
expectedOutputIsInformationSet2 := true;
expectedOutputIsInformationSet3 := false;
expectedOutputIsInformationSet4 := false;
expectedInformationSpace := VectorSpace(GF(3), 10);

OutputAllInformationSets := AllInformationSets(C);
OutputIsSystematic, OutputI, OutputTup := IsSystematic(C);
OutputInformationSet, OutputInformationSetTup := InformationSet(C);
OutputIsInformationSet1 := IsInformationSet(C, I1);
OutputIsInformationSet2 := IsInformationSet(C, I2);
OutputIsInformationSet3 := IsInformationSet(C, I3);
OutputIsInformationSet4 := IsInformationSet(C, I4);

assert expectedOutputIsSystematic eq OutputIsSystematic;
assert IsInformationSet(C, OutputI);
assert IsInformationSet(C, OutputTup);
assert IsInformationSet(QaryCode(C`kernel), OutputTup[1]);
assert OutputI eq Sort(OutputTup[1] cat OutputTup[2]);
assert IsInformationSet(C, OutputInformationSet);
assert IsInformationSet(C, OutputInformationSetTup);
assert IsInformationSet(QaryCode(C`kernel), OutputInformationSetTup[1]);
assert OutputInformationSet eq Sort(OutputTup[1] cat OutputTup[2]);
assert #expectedOutputAllInformationSets eq #OutputAllInformationSets;
assert Set(expectedOutputAllInformationSets) eq Set(OutputAllInformationSets);
assert expectedInformationSpace eq InformationSpace(C);
assert expectedOutputIsInformationSet1 eq OutputIsInformationSet1;
assert expectedOutputIsInformationSet2 eq OutputIsInformationSet2;
assert expectedOutputIsInformationSet3 eq OutputIsInformationSet3;
assert expectedOutputIsInformationSet4 eq OutputIsInformationSet4;
```

Anem a explicar el test mostrat.

- A la línia 1 escrivim la informació del codi q-ari C.
- De la línia 2 a la línia 3 generem el codi q-ari C.
- De la línia 4 a la línea 8 escrivim els conjunts d'informació que provarem.
- A la línia 10 escrivim tots els conjunts d'informació del codi C.
- A la línia 12 escrivim *true*, que és el primer resultat esperat de la funció `IsSystematic(C)`.

- De la línia 13 a la línia 16 escrivim els resultats esperats dels quatre conjunts d'informació.
- A la línia 17 escrivim l'espai de vectors d'informació del codi C.
- De la línia 19 a la línia 25 calculem les funcions `AllInformationSets(C)`, `IsSystematic(C)`, `InformationSet(C)` i `IsInformationSet(C,`I_i`)` amb $i = 1, \ldots, 4$ i guardem tots els resultats en variables.
- De la línia 27 a la línia 35 fem un `assert` per comprovar que tots els resultats de la funció `IsSystematic(C)` siguin correctes.
- De la línia 36 a la línia 37 fem un `assert` per comprovar que la funció `AllInformationSets(C)` sigui correcta.
- A la línia 38 fem un `assert` per comprovar que la funció `InformationSpace(C)` sigui correcta.
- De la línia 39 a la línia 42 fem un `assert` per comprovar que la funció `IsInformationSet(C,`I_i`)` sigui correcta per cada I_i amb $i = 1, \ldots, 4$.

4.7 Tests de rendiment

Un cop testejades i modificades les funcions, el següent pas és estudiar el rendiment per determinar quin de tots els diferents mètodes i algorismes creats per cada funció actuarà d'una forma més òptima en la majoria de casos. Per aquesta secció hem utilitzat informació de [13] i [17].

Per analitzar el rendiment dels diferents algorismes proposats només canviant la dimensió del kernel i els seus representants de classe, hem construït nous codis q-aris "artificials" a partir d'un codi q-ari fixat. Comencem amb un codi q-ari amb un valor alt de la dimensió del kernel k. Aleshores, triem un subespai del kernel de dimensió $k-1$ com a kernel del nou codi.

Calculem els seus representants de classe del nou codi, de manera que el nou codi tingui les mateixes paraules codi que el codi inicial. Llavors, tots dos codis tenen exactament els mateixos paràmetres i només difereixen en la dimensió del kernel i els seus representants de classe.

S'han dissenyat i implementat diferents funcions de forma modular per executar aquestes proves. Construeixen els codis "artificials" amb diferents dimensions del kernel, executant cada funció 20 vegades per estimar un temps d'execució i validar les sortides de les funcions.

Expliquem ara com hem fet aquestes particions del kernel. Donat un codi q-ari, tenim que $q^k = \frac{M}{t+1}$ on k és la dimensió del kernel, $t+1$ el número de representants i M el nombre de paraules codi de C. Quan realitzem particions al kernel, el codi resultant no es comprimeix del tot, però conserva les mateixes paraules codi. Observem que es compleix la següent igualtat:

$$q^{k-i} = \frac{M}{q^i(t+1)}, \quad \textit{para} \quad i = 0, \ldots, k.$$

En general, cada cop que reduïm la dimensió del kernel es multiplica el nombre de representants per q.

Per realitzar aquest estudi aprofitarem els diferents codis de la Taula 7. La nostra intenció es poder veure diferències de rendiment en els algorismes quan el kernel varia fixant les paraules codi i els paràmetres.

En aquesta secció analitzarem els test de rendiment per totes les funcions i algorismes a excepció de:

1. Els algorismes `IsInformationSetV1(C, I)` i `IsInformationSetV2(C, I)` quan I és donat com una tupla, per falta de temps. No es pot utilitzar el mateix test de rendiment implementat i s'hauria de crear un nou per aquesta funció en concret.

2. Els algorismes `IsSystematicV6(C)` i `IsSystematicV5(C)`, ja que no té sentit analitzar dos algorismes que sabem que són més lents en comparació a `IsSystematicV3(C)`, després de fer el test de rendiment per la funció `IsInformationSet(C, I)`.

3. L'algorisme `AllInformationSetsV4(C)`, ja que és un *BruteForce* i a més, no utilitza una estructura kernel i representants de classe. En temps de rendiment hauria de ser semblant a `AllInformationSetsV3(C)`.

4. La funció `InformationSpace(C)`, ja que només té un algorisme directe.

5. La funció `InformationSet(C)`, ja que ve donada dels resultats de `IsSystematic(C)`.

4.7.1 Funció IsInformationSet(C,I)

Recordem que segons la definició que hem donat a la funció, el conjunt d'informació I pot ser donat com una seqüència d'enters o com una tupla.

Pel cas que I sigui donat com una seqüencia d'enters teniem tres algorismes diferents.

1. `IsInformationSetV3(C, I)`. Marcat amb el color verd en els següents gràfics i denotat com $v3$.

2. `IsInformationSetV2(C, I)`. Marcat amb el color vermell en els següents gràfics i denotat com $v2$.

3. `IsInformationSetV1(C, I)`. Marcat amb el color blau en els següents gràfics i denotat com $v1$.

A continuació, hem generat quatre gràfics que mostren els resultats més significatius. L'eix x de cada gràfic representa les diferents dimensions del kernel i l'eix y el temps en segons que tarda a executar-se la funció. A més, hem generat dues taules on apareixen els resultats de tots els codis estudiats. Una taula representa l'estudi de la funció quan li donem un conjunt d'informació *true* i l'altre taula per un conjunt d'informació *false*. Aquestes taules es poden trobar a la Figura 11 i la Figura 12 dels Annexes.

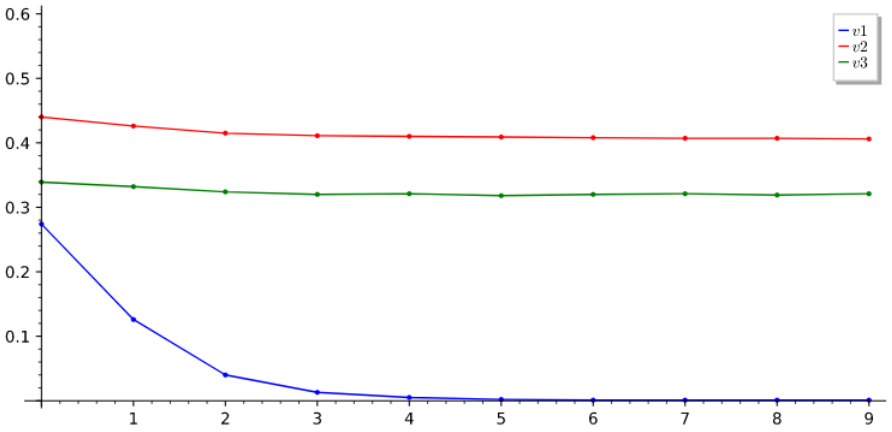

Figura 1: Temps de càlcul dels algorismes de IsInformationSet per C_5 i el conjunt d'informació $I = \{1, 2, 3, 4, 5, 6, 7, 8, 10, 11\}$

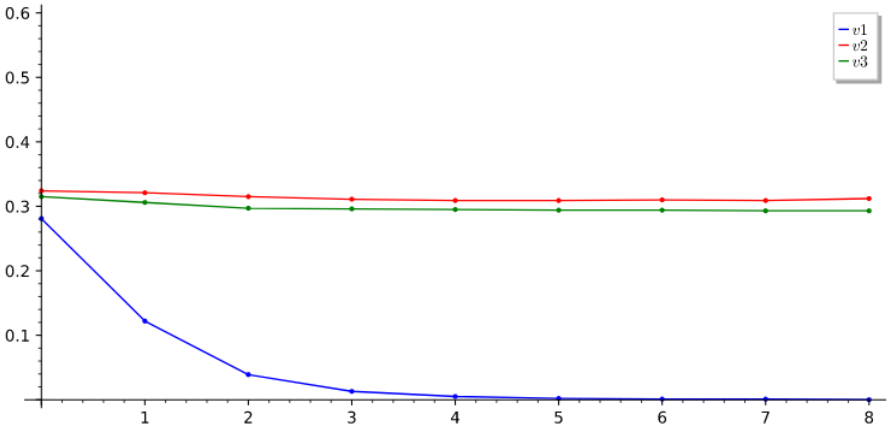

Figura 2: Temps de càlcul dels algorismes de IsInformationSet per C_4 i el conjunt d'informació $I = \{1, 2, 3, 4, 5, 6, 7, 8, 10, 11\}$

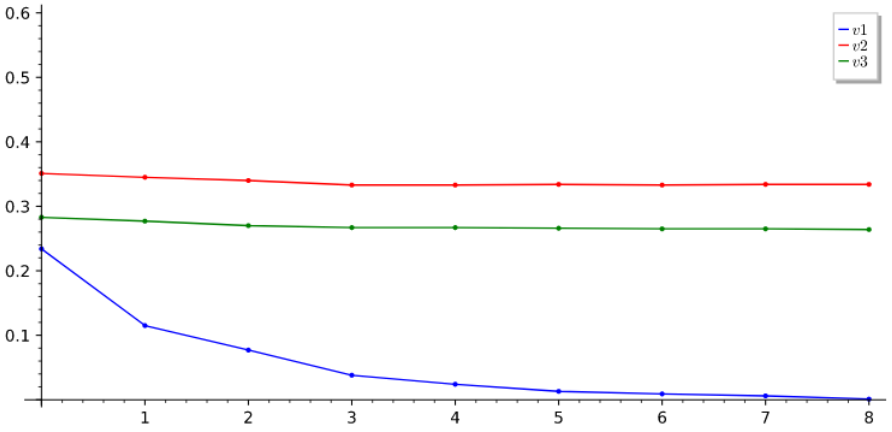

Figura 3: Temps de càlcul dels algorismes de IsInformationSet per C_4 i el conjunt d'informació $I = \{1, \ldots, 10\}$

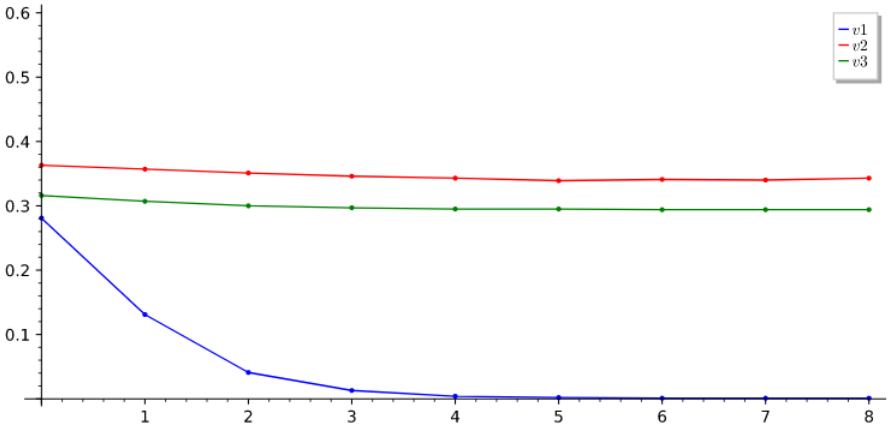

Figura 4: Temps de càlcul dels algorismes de IsInformationSet per C_4 i el conjunt d'informació $I = \{1, 2, 3, 4, 5, 6, 7, 8, 10, 12\}$

En general, si observem les 4 figures anteriors, notem que l'algorisme $v1$ és més eficient que els algorismes $v2$ i $v3$ per diferents codis, per diferents conjunts d'informació i per diferents dimensions del kernel. A més, quan la dimensió del kernel creix, l'algorisme $v1$ cada cop és més eficient respecte els altres dos algorismes que no varien el temps d'execució respecte a la dimensió del kernel.

A partir d'aquests resultats, s'ha decidit deixar la implementació de la funció seguint l'algorisme $v1$ i en un futur es preveu demostrar que aquest comportament és sempre així de forma més analítica analitzant la complexitat dels diferents algorismes.

4.7.2 Funció IsSystematic(C)

A continuació, hem generat tres gràfics que mostren els resultats més significatius. L'eix x de cada gràfic representa les diferents dimensions del kernel i l'eix y el temps en segons que tarda a executar-se la funció. A més, hem generat una taula on apareixen els resultats de tots els codis estudiats. Aquesta taula es pot trobar a la Figura 13 dels Annexes. Destaquem que:

1. `IsSystematicV4(C)`. Marcat amb el color taronja en els següents gràfics i denotat com $v4$.
2. `IsSystematicV3(C)`. Marcat amb el color verd en els següents gràfics i denotat com $v3$.
3. `IsSystematicV2(C)`. Marcat amb el color vermell en els següents gràfics i denotat com $v2$.
4. `IsSystematicV1(C)`. Marcat amb el color blau en els següents gràfics i denotat com $v1$.

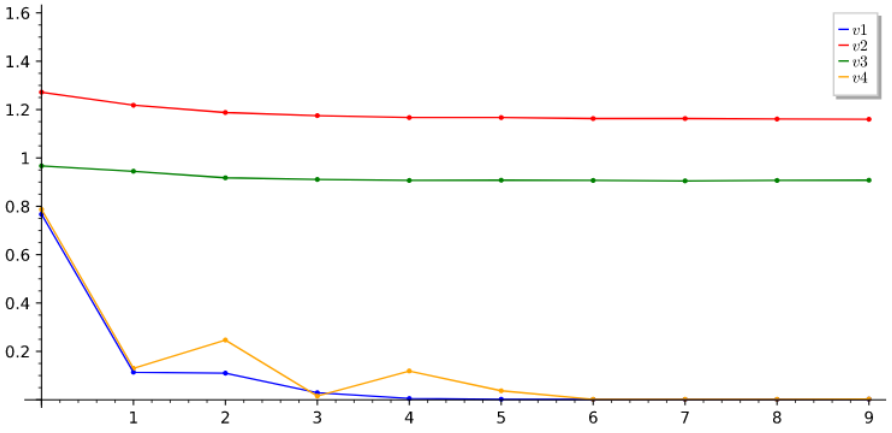

Figura 5: Temps de càlcul dels algorismes de IsSystematic per C_5

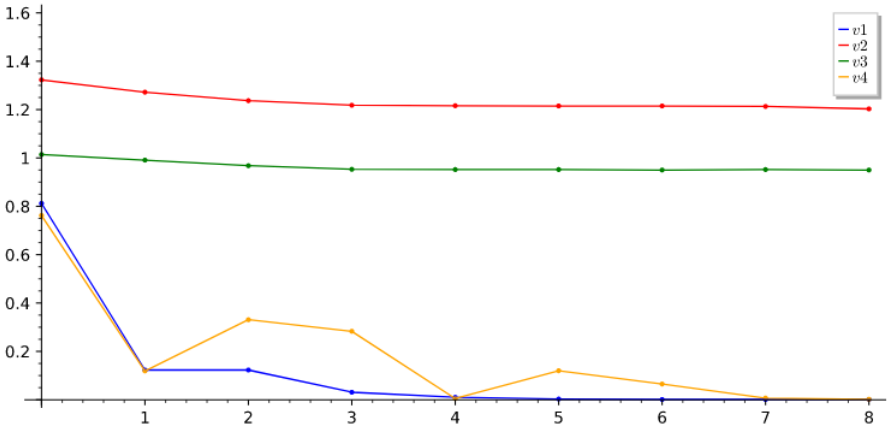

Figura 6: Temps de càlcul dels algorismes de IsSystematic per C_4

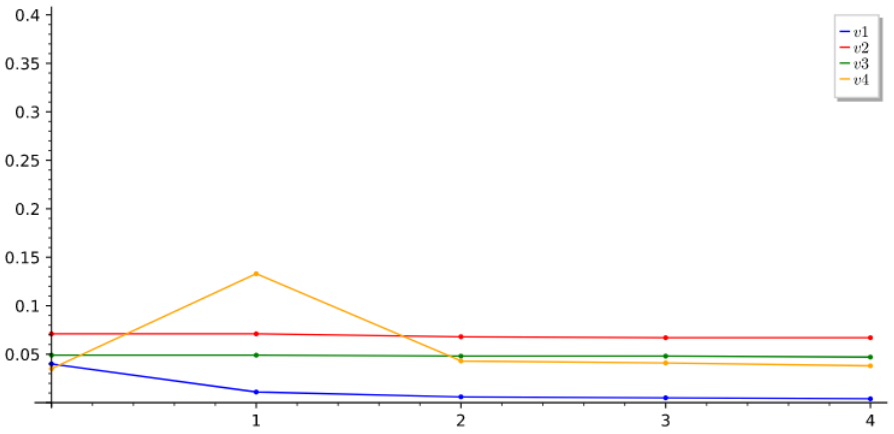

Figura 7: Temps de càlcul dels algorismes de IsSystematic per C_{11}

En general, si observem les 3 figures anteriors, notem que els algorismes $v1$ i $v4$ són més eficient que els algorismes $v2$ i $v3$ per diferents codis i per diferents dimensions del kernel. Observem que quan usem el codi C_{11} (Figura 7), les diferències de rendiment no són tan notòries. Això és degut el fet que el codi estudiat no té tantes paraules codi i per tant a l'hora d'executar els càlculs, els temps són molt ràpids indiferentment de l'algorisme utilitzat.

A més, quan la dimensió del kernel creix, els algorismes $v1$ i $v4$ cada cop són més eficient respecte els altres dos algorismes que no varien el temps d'execució respecte a la dimensió del kernel.

Observem ara les diferències entre els algorismes $v1$ i $v4$. Notem que l'algorisme $v1$ dona uns resultats més fiables respecte a l'algorisme $v4$, ja que a vegades pot tenir un temps d'execució molt eficient i a vegades un temps igual als dos algorismes descartats.

A partir d'aquests resultats, s'ha decidit deixar la implementació de la funció seguint l'algorisme $v1$ i en un futur es preveu demostrar que aquest comportament és sempre així de forma més analítica analitzant la complexitat dels diferents algorismes.

4.7.3 Funció AllInformationSets(C)

A continuació, hem generat tres gràfics que mostren els resultats més significatius. L'eix x de cada gràfic representa les diferents dimensions del kernel i l'eix y el temps en segons que tarda a executar-se la funció. A més hem generat una taula on apareixen els resultats de tots els codis estudiats. Aquesta taula es pot trobar a la Figura 14 dels Annexes. Destaquem que:

1. `AllInformationSetsV3(C)`. Marcat amb el color verd en els següents gràfics i denotat com $v3$.
2. `AllInformationSetsV2(C)`. Marcat amb el color vermell en els següents gràfics i denotat com $v2$.
3. `AllInformationSetsV1(C)`. Marcat amb el color blau en els següents gràfics i denotat com $v1$.

Estudiem primer de tot la taula referida per descartar casos. En la majoria de codis estudiats observem un patró: l'algorisme $v3$ és molt lent respecte als algorismes $v1$ i $v2$. En alguns casos, l'algorisme $v3$ tarda 125 segons aproximadament mentre que els altres dos algorismes no tarden més d'un segon. Per tant, queda descartat aquest algorisme totalment.

Fixem-nos en els tres casos més significatius. Escollim 3 codis diferents amb una dimensió del kernel elevada per poder fer més particions i extreure millors resultats.

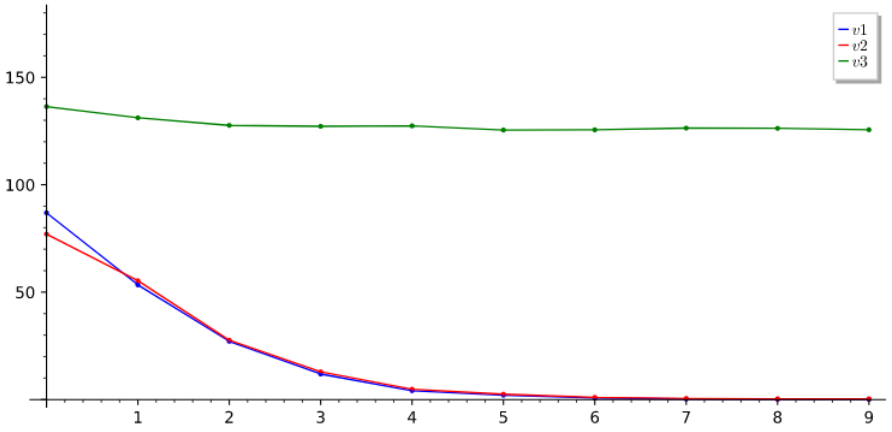

Figura 8: Temps de càlcul dels algorismes de AllInformationSets per C_5

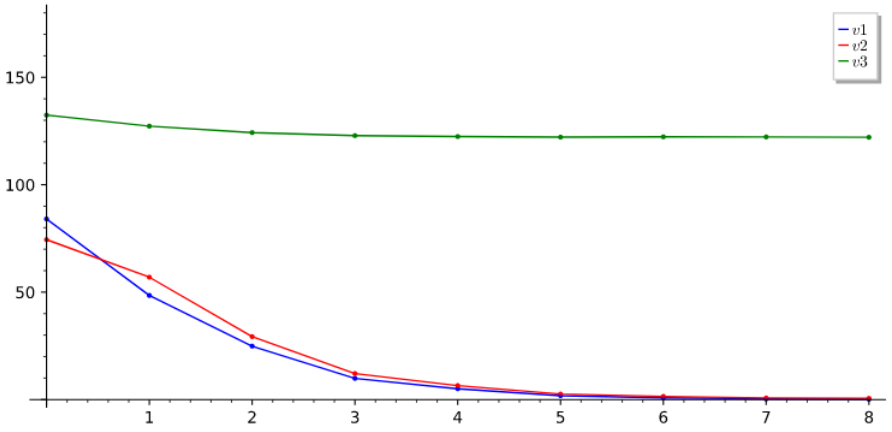

Figura 9: Temps de càlcul dels algorismes de AllInformationSets per C_4

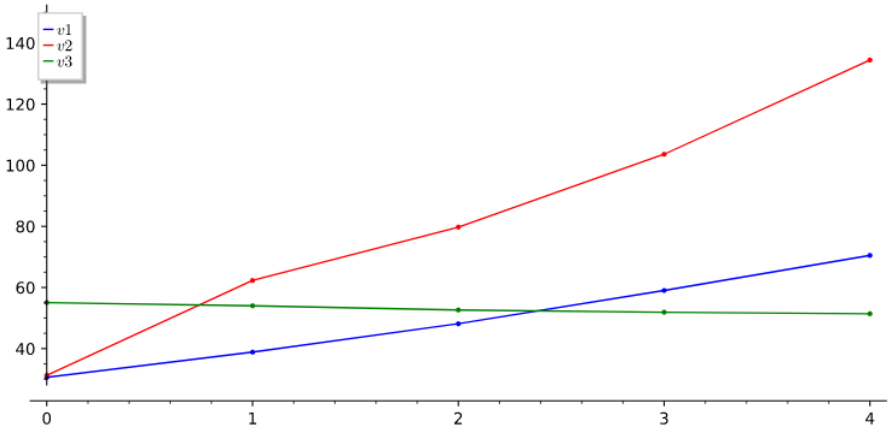

Figura 10: Temps de càlcul dels algorismes de AllInformationSets per C_{11}

En les dues primeres figures reafirmem els resultats esperats: l'algorisme $v3$ no és gens òptim. A més observem que els dos algorismes restants són bastant similars i cada un actua millor segons la dimensió

del kernel però per pocs segons de diferència.

Fixem-nos en la tercera figura. Observem que els tres algorismes tarden molt a l'hora d'executar la funció i això és causat per la quantitat de conjunts d'informació del codi C_{11} (Figura 10), que té un total de 832 conjunts d'informació. Llavors, quan un codi té un nombre elevat de conjunts d'informació que calcular, l'algorisme $v2$ s'executa molt més lent que l'algorisme $v1$.

En general, si observem les 3 figures anteriors i fem una mitja, observem que l'algorisme $v1$ és més eficient que els algorismes $v2$ i $v3$ per diferents codis i per diferents dimensions del kernel.

A partir d'aquests resultats, s'ha decidit deixar la implementació de la funció seguint l'algorisme $v1$ i en un futur es preveu demostrar que aquest comportament és sempre així de forma més analítica analitzant la complexitat dels diferents algorismes.

5 Conclusions

En aquest projecte, hem assolit amb èxit la majoria d'objectius proposats i descrits a la secció 4.2. Hem implementat diferents algorismes per les funcions: `IsInformationSet(C,I)`, `IsSystematic(C)`, `InformationSpace(C)`, `InformationSet(C)` i `AllInformationSets(C)`. Els algorismes donats en aquest projecte han estat provats i demostrats amb resultats teòrics. Tots els tests han estat realitzats en Magma i els algorismes escrits en funcions seguint el llenguatge de programació propi de Magma. A més, tant pels tests com per les funcions s'ha seguit el mateix estil que per la resta de funcions ja implementades del paquet. Aquestes funcions s'incorporaran en una nova llibreria, que serà publicada per treballar amb codis no lineals. Està previst que a finals d'aquest any es publiqui a la web del grup CCSG (Coding Combinatiorial and Security Group) una nova versió (2.0) d'aquest paquet incloent les funcions implementades en aquest projecte [4].

5.1 Resultats principals

Recordem que el nostre treball l'havíem dividit en tres etapes.

En la primera etapa del treball hem après i entès les característiques generals sobre la teoria de codis. Al final, hem estudiat conceptes bàsics sobre teoria de codis lineals i a la vegada repassat cossos finits, hem vist exemples de codis detectors i corrector d'erros, i per acabar, ens hem centrat en els codis no lineals i en la seva representació a través del kernel i els seus representants de classe.

En la segona etapa del treball hem fet èmfasi en la secció de la teoria de codis que parla sobre codis sistemàtics i conjunts d'informació. Hem entès les bases per dissenyar les funcions que volíem implementar a la llibreria de Magma i també hem vist definicions més generals sobre un codi sistemàtic i un conjunt d'informació. A més, hem vist com coincidien amb les definicions ja donades quan teníem un codi lineal. Finalment, hem demostrat un resultat que permet determinar si un conjunt és d'informació de forma més eficient.

En la tercera etapa del treball hem desenvolupat diferents algorismes per les funcions a implementar i els seus tests respectivament. Hem comprovat que tots els algorismes funcionessin amb els tests de caixa negra, per finalment estudiar el seu rendiment.

- Per la funció `IsInformationSet(C,I)` i quan I és donat com una seqüència d'enters, hem vist que en general, l'algorisme `IsInformationSetV1(C,I)` és més eficient que les altres versions per qualsevol dimensió del kernel.
- Per la funció `IsSystematic(C)`, hem vist que en general, l'algorisme `IsSystematicV1(C,I)` és més eficient que les altres versions per qualsevol dimensió del kernel.
- Per la funció `InformationSpace(C)`, no teníem cap altra versió, ja que només té un algorisme directe.
- Per la funció `InformationSet(C)`, no teníem cap altra versió, ja que ve donada dels resultats de `IsSystematic(C)`. Per tant hem escollit l'algorisme més òptim de `IsSystematic(C)`.
- Per la funció `AllInformationSets(C)`, hem vist que en la majoria dels casos, l'algorisme `AllInformationSetsV1(C,I)` és el més eficient per qualsevol dimensió del kernel. Indicar que com que els resultats eren bastant semblants a l'algorisme `AllInformationSetsV2(C,I)` hem implementat que l'usuari que vulgui executar la funció, disposi dels dos algorismes i, per defecte, si l'usuari no escull cap algorisme, s'executarà el que hem vist que sembla més eficient en la majoria dels casos.

5.2 Futures millores i possibles continuacions del treball

A continuació es mostren un seguit de millores que hem considerat per seguir continuant i treballant amb el projecte i així, millorar els resultats obtinguts.

- Continuar amb els test de rendiment de la funció `IsInformationSet(C, I)` i els seus dos algorismes `IsInformationSetV1(C, I)` i `IsInformationSetV2(C, I)`, quan I és donat com una tupla. S'haurien de crear nous tests de rendiment diferents dels de les altres funcions per poder obtenir uns resultats que permetin comparar els dos algorismes.
- Considerar la generació de tots els subconjunts de coordenades d'una mida fixada, de forma iterativa. En les implementacions actuals de les funcions, es genera tot un conjunt de subconjunts i s'analitza cadascun dels subconjunts, per veure si serveix en el procés d'obtenir si un conjunt de coordenades és d'informació o no. En la nova proposta, els conjunts es van creant alhora que es van necessitant, per tant no cal emmagatzemar-los tot en memòria i això pot suposar un estalvi important de memòria.
- Demostrar analíticament que les versions que afirmem que són més eficients a través dels tests de rendiment, sempre (o en la majoria de casos) són més eficients.
- Dissenyar, implementar i testejar noves funcions relacionades amb les implementades. Per exemple, funcions que codifiquin un vector d'informació en un codi sistemàtic, amb l'opció d'especificar on situar les coordenades d'informació fent servir una codificació sistemàtica. També es pot generalitzar el concepte de matriu en forma estàndard, per a codis no lineals i funcions que permetin trobar un codi equivalent que estigui en forma estàndard.
- Entrar més en detall en la teoria de codificació i veure fins a quin cert punt val la pena treballar amb codis no lineals sistemàtics i quins avantatges aporta, en comparació a codis lineals. En particular, també es pot estudiar mètodes de descodificació basats en una codificació sistemàtica i en forma estàndard.

Referències

[1] J. Cannon, W. Bosma, C. Fieker, i A. Steel, *Magma Computational Algebra System*, http://magma.maths.usyd.edu.au/magma/, versió 2.25-5, 2020.

[2] C. E. Shannon, *A mathematical theory of communication* Bell System Technical Journal, vol. 27, no. 10, p. 379–423, 1998.

[3] R. B. Ash, *Information theory*, New York: John Wiley and Sons Inc, 1965.

[4] J. Pujol i M. Villanueva, *Q-ary codes. A MAGMA package*, http://ccsg/uab.cat, versió 1.0, 2017.

[5] F. Heß, *Short course at IHP Paris*, September 2004.

[6] F. Zeng, *Nonlinear codes: representation, contructions, minimum distance computing and decoding*, Tesis, Universitat Autònoma de Barcelona, 2014.

[7] D.G. Hoffman, D.A. Leonard, C.C. Lidner, i K.T. Phelps, *Coding Theory: The Essentials*, Marcel Dekker Inc, 1991.

[8] R. B. Nenno, *An Introduction to the Theory of Nonlinear Error-Correcting Codes*, Tesis, Rochester Institute of Technology, 1987.

[9] M. Villanueva, F. Zeng i J. Pujol, *Efficient representation of binary nonlinear codes: constructions and minimum distance computation*, Designs, Codes and Cryptography 76: 3-21, 2015.

[10] F.J MacWilliams i N.J.A. Sloane, *The Theory of Error-Correcting Codes*, North-Holland Mathematical Library, 1977.

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

[12] M. Villanueva i C. Fernàndez-Córdoba, *Codis detectors i correctors d'errors i algunes de les seves aplicacions a la societat de la informació*, Butlletí De La Societat Catalana De Matemàtiques, p. 53-89, 2019.

[13] R. Montes, *Ampliación de una librería en Magma para la construcción de códigos q-arios*, Treball Final de Grau, Universitat Autònoma de Barcelona, 2019.

[14] J. Pujol i M. Villanueva, *Article en preparació: Q-ary codes*, 2020.

[15] T. G. Group, Group, GAP – Groups, *Algorithms and Programming*, http://www.gap-system.org, versió 4.11, 2019.

[16] W. A. Stein, *Software for Algebra and Geometry Experimentation*, http://www.sagemath.org/, versió 9.1, 2019.

[17] A. Figuerola, *Permutation Automorphism Groups for Q-ary Codes. Development in MAGMA*, Treball Final de Grau, Universitat Autònoma de Barcelona, 2019.

[18] J. Cannon, W. Bosma, C. Fieker, i A. Steel, *Handbook of of MAGMA functions*, http://magma.maths.usyd.edu.au/magma, versió 2.22, 2016.

7 Annexes

```
/*************************************************************/
/*                                                           */
/* Function name: IsSystematic                               */
/* Parameters: C                                             */
/* Function description:  Return true if and only if the     */
/* q-ary code C of length n is systematic, that is, there is */
/* a set of $t$ coordinate positions I C {1,..,n}, called    */
/* information set, such that |C|=|C_I|=q^t, where           */
/* C_I={ v_I : v in C } and v_I denote the restriction of    */
/* the vector v to the coordinates in I.                     */
/* Input parameters description:                             */
/*   - C: A q-ary code                                       */
/* Output parameters description:                            */
/*   - true if I is Systematic, false otherwise              */
/*   - An information set I for C                            */
/*   - A tuple < I_k, I_r > where I_k is an information      */
/*     set for the kernel K(C) and I=I_k U I_r is an         */
/*     information set for C.                                */
/*                                                           */
/* Signature: (<CodeFld> C) -> BoolElt, SeqEnum, Tup         */
/*                                                           */
/*************************************************************/
```

```
intrinsic IsSystematicV1(C::CodeFld) -> BoolElt, SeqEnum, Tup
{Return true if and only if the q-ary code C of length n is systematic, that is,
there is a set of t coordinate positions I C [1,..,n], called information set,
such that |C|=|C_I|=q^t, where C_I=[ v_I : v in C ] and v_I denote the restriction
of the vector v to the coordinates in I.}

    if(C`IsLinear) then
        IK := InformationSet(C`Kernel);
        return true, IK, <IK,[]>;
    end if;

    n := Length(C);
    t, e := ParametersInfoSet(#C, C`BaseField);
    k :=  Dimension(C`Kernel);
    r := t-k;
    V := InformationSpace(C`Kernel);
    numCosetRep := #C`CosetRepresentatives;

    // #C is not a power of q
    if e ne 1 then
        return false, [], <[],[]>;
    end if;

    // #C is 1, so C is the zero code
    if t eq 0 then
        return true, [], <[], []>;
    end if;

    if k ne 0 then
        for IK in AllInformationSets(C`Kernel) do
            GIK := DiagonalizeKernelFromInfoSetK(C`Kernel, IK);
            newCosetRepresentatives := [v - V![v[i] : i in IK] * GIK :
                                                    v in C`CosetRepresentatives];
            for IR in Subsets({1..n} diff Set(IK), r) do
                setRepresentatives := { [v[i] : i in IR] :
                                                  v in newCosetRepresentatives };
                if #setRepresentatives eq numCosetRep then
                    return true, Sort(IK cat Setseq(IR)), <IK, Setseq(IR)>;
                end if;
            end for;
        end for;
    else
        for I in Subsets({1..n}, t) do
            CIcodewords := { [v[i] : i in I] : v in C`CosetRepresentatives };
            if #CIcodewords eq #C then
                newI := Setseq(I);
                return true, newI, <[], newI>;
            end if;
        end for;
    end if;

    return false, [], <[],[]>;

end intrinsic;
```

```
intrinsic IsSystematicV2(C::CodeFld) -> BoolElt, SeqEnum
{}

    n := Length(C);
    t, e := ParametersInfoSet(#C, C`BaseField);

    if e ne 1 then
        return false, [];
    end if;

    if t eq 0 then
        return true, [];
    end if;

    for I in Subsets({1..n}, t) do
        CIcodewords := { [c[i] : i in I] : c in Set(C) };
        if #CIcodewords eq #C then
            return true, Setseq(I);
        end if;
    end for;

    return false, [];

end intrinsic;
```

```
intrinsic IsSystematicV3(C::CodeFld) -> BoolElt, SeqEnum
{}

    n := Length(C);
    t, e := ParametersInfoSet(#C, C`BaseField);

    if e ne 1 then
        return false, [];
    end if;

    if t eq 0 then
        return true, [];
    end if;

    for I in Subsets({1..n}, t) do
        CIcodewords := { [(k+v)[i] : i in I] : k in C`Kernel,
                                               v in C`CosetRepresentatives };
        if #CIcodewords eq #C then
            return true, Setseq(I);
        end if;
    end for;

    return false, [];

end intrinsic;
```

```
intrinsic IsSystematicV4(C::CodeFld) -> BoolElt, SeqEnum, Tup
{}

    n := Length(C);
    t, e := ParametersInfoSet(#C, C`BaseField);
    k := Dimension(C`Kernel);
    r := t-k;
    G := GeneratorMatrix(C`Kernel);

    if e ne 1 then
        return false, [], <[],[]>;
    end if;

    if t eq 0 then
        return true, [], <[], []>;
    end if;

    for IK in Subsets({1..n}, k) do
        if(Rank(Submatrix(G, [1..k], Setseq(IK))) eq k) then
            for IR in Subsets({1..n} diff IK, r) do
                newI := IK join IR;
                if(IsInformationSet(C, newI)) then
                    return true, Setseq(newI), <Setseq(IK), Setseq(IR)>;
                end if;
            end for;
        end if;
    end for;

    return false, [], <[],[]>;

end intrinsic;
```

```
intrinsic IsSystematicV5(C::CodeFld) -> BoolElt, SeqEnum, Tup
{}

    n := Length(C);
    t, e := ParametersInfoSet(#C, C`BaseField);
    k := Dimension(C`Kernel);
    r := t-k;
    G := GeneratorMatrix(C`Kernel);

    if e ne 1 then
        return false, [], <[],[]>;
    end if;

    if t eq 0 then
        return true, [], <[], []>;
    end if;

    for IK in Subsets({1..n}, k) do
        if(Rank(Submatrix(G, [1..k], Setseq(IK))) eq k) then
            for IR in Subsets({1..n} diff IK, r) do
                newI := IK join IR;
                if(IsInformationSetV2(C, newI)) then
                    return true, Setseq(newI), <Setseq(IK), Setseq(IR)>;
                end if;
            end for;
        end if;
    end for;

    return false, [], <[],[]>;

end intrinsic;
```

```
intrinsic IsSystematicV6(C::CodeFld) -> BoolElt, SeqEnum, Tup
{}

    n := Length(C);
    t, e := ParametersInfoSet(#C, C`BaseField);
    k := Dimension(C`Kernel);
    r := t-k;
    G := GeneratorMatrix(C`Kernel);

    if e ne 1 then
        return false, [], <[],[]>;
    end if;

    if t eq 0 then
        return true, [], <[], []>;
    end if;

    for IK in Subsets({1..n}, k) do
        if(Rank(Submatrix(G, [1..k], Setseq(IK))) eq k) then
            for IR in Subsets({1..n} diff IK, r) do
                newI := IK join IR;
                if(IsInformationSetV3(C, newI)) then
                    return true, Setseq(newI), <Setseq(IK), Setseq(IR)>;
                end if;
            end for;
        end if;
    end for;

    return false, [], <[],[]>;

end intrinsic;
```

```
/*****************************************************************/
/*                                                               */
/* Function name: IsInformationSet                               */
/* Parameters: C, I                                              */
/* Function description: Return true if and only if the set      */
/* I C {1,...,n} of coordinate positions is an information set   */
/* for the q-ary code C of length n. An information set          */
/* for C is an ordered set of t coordinate positions             */
/* I C {1,...,n} such that |C|=|C_I|=q^t, where                  */
/* C_I={ v_I | v in C } and v_I denote the restriction of the    */
/* vector v to the coordinates in I.                             */
/* The set I can be given as a set or sequence of integers,      */
/* or as a tuple < I_k, I_r > with two sets or sequences         */
/* of integers. In the case that I is given as a tuple, the      */
/* function returns true if and only if I_k is an information    */
/* set for the kernel K(C) and I=I_k U I_r is an information     */
/* set for C.                                                    */
/*   - C: A q-ary code                                           */
/*   - I: A subset of coordinate possitions                      */
/* Output parameters description:                                */
/*   - true if I is an information set for C, false otherwise    */
/*                                                               */
/* Signature: (<CodeFld> C, <SetEnum> I) -> BoolElt              */
/*                                                               */
/*****************************************************************/
```

```
intrinsic IsInformationSetV1(C::CodeFld, I::SeqEnum) -> BoolElt
{Return true if and only if the set I C [1,...,n] of coordinate positions is
an information set for the q-ary code C of length n. An information set for C
is an ordered set of t coordinate positions I C [1,...,n] such that
|C|=|C_I|=q^t, where C_I=[ v_I | v in C ] and v_I denote the restriction of
the vector v to the coordinates in I. The set I can be given as a set or sequence
of integers, or as a tuple < I_k, I_r > with two sets or sequences of integers.
In the case that I is given as a tuple, the function returns true if and only if I_k
is an information set for the kernel K(C) and I=I_k U I_r is an information set for C.}

    n := Length(C);
    require (I subset [1..n]): "Argument 2 is not a subset of coordinates";
    t, e := ParametersInfoSet(#C, C`BaseField);
    k := Dimension(C`Kernel);
    I := Set(I);
    if(C`IsLinear) then
        if #I eq k then
            return Rank(Submatrix(GeneratorMatrix(C`Kernel), [1..k], Setseq(I))) eq k;
        end if;
        return false;
    end if;
    r := t-k;
    V := InformationSpace(C`Kernel);
    numCosetRep := #C`CosetRepresentatives;
    // #C is not a power of q
    if e ne 1 then
        return false;
    // Since #C=q^t, #I has to be equal to t
    elif t ne #I then
        return false;
    else
        if k ne 0 then
            for IK in AllInformationSets(C`Kernel) do
                if IK subset I then
                    GIK := DiagonalizeKernelFromInfoSetK(C`Kernel, IK);
                    newCosetRepresentatives := [v - V![v[i] : i in IK] * GIK :
                                                v in C`CosetRepresentatives];
                    for IR in Subsets(I diff Set(IK), r) do
                        setRepresentatives := { [v[i] : i in IR] :
                                                v in newCosetRepresentatives };
                        if #setRepresentatives eq numCosetRep then
                            return true;
                        end if;
                    end for;
                end if;
            end for;
        else
            CIcodewords := { [v[i] : i in I] : v in C`CosetRepresentatives };
            if #CIcodewords eq #C then
                return true;
            end if;
        end if;
    end if;
    return false;

end intrinsic;
```

```
intrinsic IsInformationSetV2(C::CodeFld, I::SeqEnum) -> BoolElt
{}

    n := Length(C);
    require (I subset [1..n]):"Argument 2 is not a subset of coordinates";
    I := Set(I);
    t, e := ParametersInfoSet(#C, C`BaseField);

    if e ne 1 then
        return false;
    elif t ne #I then
        return false;
    else
        CIcodewords := { [(k+v)[i] : i in I] : k in C`Kernel,
                                               v in C`CosetRepresentatives };

        if #CIcodewords eq #C then
            return true;
        else
            return false;
        end if;
    end if;

end intrinsic;




intrinsic IsInformationSetV3(C::CodeFld, I::SeqEnum) -> BoolElt
{}

    n := Length(C);
    require (I subset [1..n]):"Argument 2 is not a subset of coordinates";
    I := Set(I);
    t, e := ParametersInfoSet(#C, C`BaseField);

    if e ne 1 then
        return false;
    elif t ne #I then
        return false;
    else
        CIcodewords := { [c[i] : i in I] : c in Set(C) };

        if #CIcodewords eq #C then
            return true;
        else
            return false;
        end if;
    end if;

end intrinsic;
```

```
/*************************************************************/
/*                                                           */
/* Function name: IsInformationSet                           */
/* Parameters: C, I                                          */
/* Function description: Return true if and only if the set  */
/* I C {1,...,n} of coordinate positions is an information set */
/* for the q-ary code C of length n. An information set      */
/* for C is an ordered set of t coordinate positions         */
/* I C {1,...,n} such that |C|=|C_I|=q^t, where              */
/* C_I={ v_I | v in C } and v_I denote the restriction of the */
/* vector v to the coordinates in I.                         */
/* The set I can be given as a set or sequence of integers,  */
/* or as a tuple < I_k, I_r > with two sets or sequences     */
/* of integers. In the case that I is given as a tuple, the  */
/* function returns true if and only if I_k is an information */
/* set for the kernel K(C) and I=I_k U I_r is an information */
/* set for C.                                                */
/*   - C: A q-ary code                                       */
/*   - I: A subset of coordinate possitions                  */
/* Output parameters description:                            */
/*   - true if I is an information set for C, false otherwise */
/*                                                           */
/* Signature: (<CodeFld> C, <SetEnum> I) -> BoolElt          */
/*                                                           */
/*************************************************************/
```

```
//Mateixa funció que l'anterior però entra un SetEnum
intrinsic IsInformationSet(C::CodeFld, I::SetEnum) -> BoolElt
{Return true if and only if the set I C [1,...,n] of coordinate positions is
an information set for the q-ary code C of length n. An information set for C
is an ordered set of t coordinate positions I C [1,...,n] such that
|C|=|C_I|=q^t, where C_I=[ v_I | v in C ] and v_I denote the restriction of
the vector v to the coordinates in I. The set I can be given as a set or sequence
of integers, or as a tuple < I_k, I_r > with two sets or sequences of integers.
In the case that I is given as a tuple, the function returns true if and only if I_k
is an information set for the kernel K(C) and I=I_k U I_r is an information set for C.}

    n := Length(C);
    require (I subset [1..n]): "Argument 2 is not a subset of coordinates";
    t, e := ParametersInfoSet(#C, C`BaseField);
    k := Dimension(C`Kernel);
    if(C`IsLinear) then
        if #I eq k then
            return Rank(Submatrix(GeneratorMatrix(C`Kernel), [1..k], Setseq(I))) eq k;
        end if;
        return false;
    end if;
    r := t-k;
    V := InformationSpace(C`Kernel);
    numCosetRep := #C`CosetRepresentatives;
    // #C is not a power of q
    if e ne 1 then
        return false;
    // Since #C=q^t, #I has to be equal to t
    elif t ne #I then
        return false;
    else
        if k ne 0 then
            for IK in AllInformationSets(C`Kernel) do
                if IK subset I then
                    GIK := DiagonalizeKernelFromInfoSetK(C`Kernel, IK);
                    newCosetRepresentatives := [v - V![v[i] : i in IK] * GIK :
                                                v in C`CosetRepresentatives];
                    for IR in Subsets(I diff Set(IK), r) do
                        setRepresentatives := { [v[i] : i in IR] :
                                                v in newCosetRepresentatives };
                        if #setRepresentatives eq numCosetRep then
                            return true;
                        end if;
                    end for;
                end if;
            end for;
        else
            CIcodewords := { [v[i] : i in I] : v in C`CosetRepresentatives };
            if #CIcodewords eq #C then
                return true;
            end if;
        end if;
    end if;
    return false;

end intrinsic;
```

```
/*************************************************************/
/*                                                           */
/* Function name: IsInformationSet                           */
/* Parameters: C, I                                          */
/* Function description: Return true if and only if the set  */
/* I C {1,...,n} of coordinate positions is an information set */
/* for the q-ary code C of length n. An information set      */
/* for C is an ordered set of t coordinate positions         */
/* I C {1,...,n} such that |C|=|C_I|=q^t, where              */
/* C_I={ v_I | v in C } and v_I denote the restriction of the */
/* vector v to the coordinates in I.                         */
/* Input parameters description:                             */
/*   - C: A linear code                                      */
/*   - I: A subset of coordinate possitions                  */
/* Output parameters description:                            */
/*   - true if I is an information set for C, false otherwise */
/*                                                           */
/* Signature: (<CodeLinFld> C, <SeqEnum> I) -> BoolElt       */
/*                                                           */
/*************************************************************/



intrinsic IsInformationSet(C::CodeLinFld, I::SeqEnum) -> BoolElt
{Return true if and only if the set I C [1,...,n] of coordinate positions
is an information set for the q-ary code C of length n. An information set
for C is an ordered set of t coordinate positions I C [1,...,n] such that
|C|=|C_I|=q^t, where C_I=[ v_I | v in C ] and v_I denote the restriction of
the vector v to the coordinates in I.}

    require (I subset [1..Length(C)]): "Argument 2 is not a subset of coordinates";
    k := Dimension(C);
    I := Set(I);

    if #I eq k then
        return Rank(Submatrix(GeneratorMatrix(C), [1..k], Setseq(I))) eq k;
    end if;

    return false;

end intrinsic;
```

```
/*****************************************************************/
/*                                                               */
/* Function name: IsInformationSet                               */
/* Parameters: C, I                                              */
/* Function description: Return true if and only if the set      */
/* I C {1,...,n} of coordinate positions is an information set   */
/* for the q-ary code C of length n. An information set          */
/* for C is an ordered set of t coordinate positions             */
/* I C {1,...,n} such that |C|=|C_I|=q^t, where                  */
/* C_I={ v_I | v in C } and v_I denote the restriction of the    */
/* vector v to the coordinates in I.                             */
/* The set I can be given as a set or sequence of integers,      */
/* or as a tuple < I_k, I_r > with two sets or sequences         */
/* of integers. In the case that I is given as a tuple, the      */
/* function returns true if and only if I_k is an information    */
/* set for the kernel K(C) and I=I_k U I_r is an information     */
/* set for C.                                                    */
/* Input parameters description:                                 */
/*   - C: A q-ary code                                           */
/*   - I: A tuple < I_k, I_r > with two sets or sequences        */
/*        of integers                                            */
/* Output parameters description:                                */
/*   - true if if I_k is an information set for the kernel K(C)  */
/*     and I=I_k U I_r is an information set for C.              */
/*                                                               */
/* Signature: (<CodeFld> C, <Tuple> I) -> BoolElt                */
/*                                                               */
/*****************************************************************/
```

```
intrinsic IsInformationSetV1(C::CodeFld, I::Tup) -> BoolElt
{Return true if and only if the set I C [1,...,n] of coordinate positions is
an information set for the q-ary code C of length n. An information set for C
is an ordered set of t coordinate positions I C [1,...,n] such that |C|=|C_I|=q^t,
where C_I=[ v_I | v in C ] and v_I denote the restriction of the vector v to the
coordinates in I. The set I can be given as a set or sequence of integers, or as
a tuple < I_k, I_r > with two sets or sequences of integers. In the case that I is
given as a tuple, the function returns true if and only if I_k is an information
set for the kernel K(C) and I=I_k U I_r is an information set for C.}

    require #I eq 2: "Argument 2 must contain two components";
    IK := I[1];
    IR := I[2];
    require (Set(IK) meet Set(IR) eq {}): "Both components in argument 2 must be disjoint";

    n := Length(C);

    require (IK subset [1..n]): "Argument 2 is not a subset of coordinates";
    require (IR subset [1..n]): "Argument 3 is not a subset of coordinates";

    k := Dimension(C`Kernel);
    r := t-k;

    require (#IK eq k): "Argument 2 must contain ",k," coordinates in first component";
    require (#IR eq r): "Argument 2 must contain ",r," coordinates in second component";

    if(C`IsLinear) then
        return Rank(Submatrix(GeneratorMatrix(C`Kernel), [1..k], IK)) eq k;
    end if;

    t, e := ParametersInfoSet(#C, C`BaseField);
    q := #C`BaseField;
    V := InformationSpace(C`Kernel);
    // #C is not a power of q
    if e ne 1 then
        return false;
    end if;
    // #C is 1, so C is the zero code
    if t eq 0 then
        return true;
    end if;
    if k ne 0 then
        if Rank(Submatrix(GeneratorMatrix(C`Kernel), [1..k], IK)) eq k then
            GIK := DiagonalizeKernelFromInfoSetK(C`Kernel, IK);
            newCosetRepresentatives := [v - V![v[i] : i in IK] * GIK :
                                                    v in C`CosetRepresentatives];
            setRepresentatives := { [v[i] : i in IR] :
                                                    v in newCosetRepresentatives};
            if #setRepresentatives eq q^r then
                return true;
            end if;
        end if;
    else
        CIcodewords := { [v[i] : i in IR] : v in C`CosetRepresentatives };
        if #CIcodewords eq #C then
            return true;
        end if;
    end if;
    return false;

end intrinsic;
```

```
intrinsic IsInformationSetV2(C::CodeFld, I::Tup) -> BoolElt
{}

    n := Length(C);
    t, _ := ParametersInfoSet(#C, C`BaseField);
    k := Dimension(C`Kernel);
    r := t-k;
    IK := I[1];
    IR := I[2];

    require (#IK eq k):"Argument 2 must contain ",k," coordinates";
    require (#IR eq r):"Argument 3 must contain ",r," coordinates";
    require (IK subset [1..n]):"Argument 2 is not a subset of coordinates";
    require (IR subset [1..n]):"Argument 3 is not a subset of coordinates";

    if(IsInformationSet(C`Kernel, IK)) then
        if(IsInformationSet(C, IK cat IR)) then
            return true;
        end if;
    end if;
    return false;

end intrinsic;
```

```
/*****************************************************************/
/*                                                               */
/* Function name: InformationSpace                               */
/* Parameters: C                                                 */
/* Function description: Given a systematic q-ary code C of      */
/* cardinality q^t, return the vector space U=F_q^t, which is    */
/* the space of information vectors for the code C. It is not    */
/* checked whether the code is systematic or not. The function   */
/* returns this vector space even if C is not systematic.        */
/* Input parameters description:                                 */
/*   - C: A q-ary code                                           */
/* Output parameters description:                                */
/*   - vector space U=Zq^t                                       */
/*                                                               */
/* Signature: (<CodeFld> C) -> ModTupFld                         */
/*                                                               */
/*****************************************************************/
```

```
intrinsic InformationSpace(C::CodeFld) -> ModTupFld
{Given a systematic q-ary code C of cardinality q^t, return the vector
space U=F_q^t, which is the space of information vectors for the code C.
It is not checked whether the code is systematic or not. The function
returns this vector space even if C is not systematic.}

    t, e := ParametersInfoSet(#C, C`BaseField);
    require (e eq 1):"Argument 1 has not a power of q cardinality";
    q := #C`BaseField;

    return VectorSpace(GF(q), t);

end intrinsic;
```

```
/*****************************************************************/
/*                                                               */
/* Function name: InformationSet                                 */
/* Parameters: C                                                 */
/* Function description: Given a systematic q-ary code C of      */
/* length n, return an information set for C. An information     */
/* set for C is an ordered set of t coordinate positions         */
/* I C {1,...,n} such that |C|=|C_I|=q^t, where                  */
/* C_I={ v_I | v \in C } and v_I denote the restriction of       */
/* the vector v to the coordinates in I. The information set I   */
/* is returned as a sequence of t integers.                      */
/* The function also returns a tuple < I_k, I_r > with two       */
/* sequences of integers such that I_k is an information set     */
/* for the kernel K(C) and I=I_k U I_r.                          */
/* It is not checked whether the code is systematic or not.      */
/* If the function does not succeed in finding an information    */
/* set, it returns an empty sequence and a tuple with two        */
/* empty sequences.                                              */
/* Input parameters description:                                 */
/*   - C: A q-ary code                                           */
/* Output parameters description:                                */
/*   - An information set I for C                                */
/*   - A tuple < I_k, I_r > where I_k is an information          */
/*     set for the kernel K(C) and I=I_k U I_r is an             */
/*     information set for C.                                    */
/*                                                               */
/* Signature: (<CodeFld> C) -> SeqEnum, Tup                      */
/*                                                               */
/*****************************************************************/



intrinsic InformationSet(C::CodeFld) -> SeqEnum, Tup
{Given a systematic q-ary code C of length n, return an information set
for C. An information set for C is an ordered set of t coordinate positions
I C [1,...,n] such that |C|=|C_I|=q^t, where C_I=[ v_I | v \in C ] and v_I
denote the restriction of the vector v to the coordinates in I. The information
set I is returned as a sequence of t integers. The function also returns a tuple
< I_k, I_r > with two sequences of integers such that I_k is an information set
for the kernel K(C) and I=I_k U I_r.}

    isSystematic, I, Tup := IsSystematic(C);

    return I, Tup;

end intrinsic;
```

```
/*************************************************************/
/*                                                           */
/* Function name: AllInformationSets                         */
/* Parameters: C                                             */
/* Function description: Given a systematic q-ary code C of  */
/* length n, return all the possible information sets of C as a */
/* (sorted) sequence of sequences. Each inner sequence contains */
/* a set of t coordinate positions I C {1,...,n} such that   */
/* |C|=|C_I|=q^t, where C_I={ v_I : v in C } and v_I denote   */
/* the restriction of the vector v to the coordinates in I.  */
/* The function also returns a sequence of tuples. Each      */
/* tuple < I_k, I_r > contains two sequences of integers such */
/* that I_k is an information set for the kernel K(C)        */
/*  and I=I_k U I_r. It is not checked whether the code is   */
/* systematic or not. If the function does not succeed in    */
/* finding an information set, it returns two empty sequences. */
/* Input parameters description:                             */
/*   - C: A q-ary code                                       */
/* Output parameters description:                            */
/*   - A sequence of all information sets for C              */
/*   - A sequence of tuples < I_k, I_r > where I_k is an     */
/*     information set for the kernel K(C) and I=I_k U I_r is */
/*     an information set for C.                             */
/*                                                           */
/* Signature: (<CodeFld> C) -> SeqEnum, Tup                  */
/*                                                           */
/*************************************************************/
```

```
intrinsic AllInformationSetsV1(C::CodeFld) -> SeqEnum, Tup
{(...)}

    allInfoSets := [];
    allInfoTup := [];
    if(C`IsLinear) then
        for I in AllInformationSets(C`Kernel) do
            Append(~allInfoSets, I);
            Append(~allInfoTup, <I, []>);
        end for;
        return allInfoSets, allInfoTup;
    end if;
    n := Length(C);
    t, e := ParametersInfoSet(#C, C`BaseField);
    k := Dimension(C`Kernel);
    r := t-k;
    V := InformationSpace(C`Kernel);
    numCosetRep := #C`CosetRepresentatives;
    // #C is not a power of q
    if e eq 1 then
        return allInfoSets, allInfoTup;
    end if;
    // #C is 1, so C is the zero code
    if t eq 0 then
        return [], <[],[]>;
    end if;
    if k ne 0 then
        for IK in AllInformationSets(C`Kernel) do
            GIK := DiagonalizeKernelFromInfoSetK(C`Kernel, IK);
            newCosetRepresentatives := [v - V![v[i] : i in IK] * GIK :
                                                v in C`CosetRepresentatives];
            for IR in Subsets({1..n} diff Set(IK), r) do
                setRepresentatives := { [v[i] : i in IR] :
                                                  v in newCosetRepresentatives };
                if #setRepresentatives eq numCosetRep then
                    newI := Sort(IK cat Setseq(IR));
                    if newI notin allInfoSets then
                        Append(~allInfoSets, newI);
                    end if;
                    Append(~allInfoTup, <IK, Setseq(IR)>);
                end if;
            end for;
        end for;
    else
        for IR in Subsets({1..n}, t) do
            CIcodewords := { [v[i] : i in IR] : v in C`CosetRepresentatives };
            if #CIcodewords eq #C then
                newI := Sort(Setseq(IR));
                Append(~allInfoSets, newI);
                Append(~allInfoTup, <[], newI>);
            end if;
        end for;
    end if;
    return allInfoSets, allInfoTup;
end intrinsic;
```

```
intrinsic AllInformationSetsV2(C::CodeFld) -> SeqEnum
{}

    n := Length(C);
    t, e := ParametersInfoSet(#C, C`BaseField);
    allInfoSets := [];

    require e eq 1:"Argument 1 is not a systematic code";

    if t eq 0 then
        return [];
    end if;

    for I in Subsets({1..n},t) do
        newI := Setseq(I);
        if(IsInformationSet(C,newI)) then
            Append(~allInfoSets, newI);
        end if;
    end for;

    require allInfoSets ne []:"Argument 1 is not a systematic code";

    return allInfoSets;

end intrinsic;



intrinsic AllInformationSetsV3(C::CodeFld) -> SeqEnum
{}

    n := Length(C);
    t, e := ParametersInfoSet(#C, C`BaseField);
    allInfoSets := [];

    require e eq 1:"Argument 1 is not a systematic code";

    if t eq 0 then
        return [];
    end if;

    for I in Subsets({1..n},t) do
        CIcodewords := { [(k+v)[i] : i in I] : k in C`Kernel,
                                               v in C`CosetRepresentatives };
        if #CIcodewords eq #C then
            Append(~allInfoSets, Sort(Setseq(I)));
        end if;
    end for;

    require allInfoSets ne []:"Argument 1 is not a systematic code";

    return allInfoSets;

end intrinsic;
```

```
intrinsic AllInformationSetsV4(C::CodeFld) -> SeqEnum
{}

    n := Length(C);
    t, e := ParametersInfoSet(#C, C`BaseField);
    allInfoSets := [];

    require e eq 1:"Argument 1 is not a systematic code";

    if t eq 0 then
        return [];
    end if;

    for I in Subsets({1..n},t) do
        CIcodewords := { [c[i] : i in I] : c in Set(C) };
        if #CIcodewords eq #C then
            Append(~allInfoSets, Sort(Setseq(I)));
        end if;
    end for;

    require allInfoSets ne []:"Argument 1 is not a systematic code";

    return allInfoSets;

end intrinsic;
```

	ALGORISMES	10	9	8	7	6	5	4	3	2	1	0
UniverseCode(GF(3), 10)	v1	0.000	0.000	0.001	0.001	0.001	0.002	0.005	0.013	0.039	0.122	0.270
	v2	0.399	0.397	0.397	0.399	0.400	0.401	0.402	0.404	0.407	0.417	0.436
	v3	0.290	0.315	0.314	0.315	0.315	0.315	0.317	0.318	0.321	0.329	0.334
RepetitionCode(GF(4), 12)	v1										0.000	0.000
	v2										0.000	0.000
	v3										0.000	0.000
ZeroCode(GF(8), 13)	v1											0.000
	v2											0.000
	v3											0.000
Cq3n13ker8c_seq	v1			0.001	0.001	0.001	0.002	0.004	0.013	0.040	0.127	0.278
	v2			0.403	0.403	0.405	0.406	0.406	0.406	0.411	0.422	0.445
	v3			0.326	0.326	0.325	0.325	0.326	0.327	0.331	0.339	0.346
Cq3n13ker9a_seq	v1		0.002	0.001	0.001	0.001	0.002	0.005	0.014	0.041	0.129	0.288
	v2		0.413	0.412	0.414	0.413	0.414	0.415	0.416	0.422	0.431	0.455
	v3		0.336	0.335	0.336	0.336	0.335	0.337	0.339	0.342	0.350	0.357
CHadq4n16ker1b_seq	v1										0.000	0.001
	v2										0.001	0.000
	v3										0.000	0.000
Cq4n16ker0_seq	v1											
	v2											
	v3											
Cq8n20ker3_seq	v1											
	v2											
	v3											
CHadq4n16ker3_seq	v1								0.000	0.000	0.000	0.001
	v2								0.000	0.001	0.000	0.001
	v3								0.001	0.000	0.000	0.000
Cq2n20ker0_seq	v1											
	v2											
	v3											
CHamq2n16ker4t_seq	v1							0.001	0.002	0.003	0.007	0.008
	v2							0.014	0.014	0.013	0.014	0.014
	v3							0.010	0.010	0.010	0.010	0.010
QaryCodeCker0q2	v1											
	v2											
	v3											
CyclicCodeCker10q2	v1	0.000	0.001	0.000	0.001	0.000	0.001	0.000	0.001	0.002	0.003	0.004
	v2	0.007	0.006	0.006	0.007	0.006	0.007	0.007	0.006	0.007	0.007	0.007
	v3	0.005	0.005	0.005	0.005	0.005	0.005	0.005	0.005	0.005	0.005	0.005

Figura 11: Temps de càlcul dels algorismes de IsInformationSet per conjunts d'informació *true* per diferents dimensions del kernel

	ALGORISMES	10	9	8	7	6	5	4	3	2	1	0
UniverseCode(GF(3), 10)	v1	0.000	0.000	0.001	0.001	0.001	0.002	0.005	0.013	0.039	0.122	0.270
	v2	0.399	0.397	0.397	0.399	0.400	0.401	0.402	0.404	0.407	0.417	0.436
	v3	0.290	0.315	0.314	0.315	0.315	0.315	0.317	0.318	0.321	0.329	0.334
RepetitionCode(GF(4), 12)	v1										0.000	0.000
	v2										0.000	0.000
	v3										0.000	0.000
ZeroCode(GF(8), 13)	v1											0.000
	v2											0.000
	v3											0.000
Cq3n13ker8c_seq	v1			0.001	0.001	0.001	0.000	0.000	0.000	0.000	0.001	0.242
	v2			0.372	0.371	0.371	0.371	0.371	0.374	0.376	0.388	0.410
	v3			0.299	0.300	0.299	0.299	0.300	0.302	0.305	0.312	0.321
Cq3n13ker9a_seq	v1		0.001	0.003	0.004	0.007	0.012	0.018	0.040	0.081	0.122	0.245
	v2		0.378	0.375	0.377	0.374	0.375	0.378	0.378	0.379	0.395	0.411
	v3		0.300	0.300	0.300	0.300	0.300	0.300	0.302	0.302	0.314	0.320
CHadq4n16ker1b_seq	v1										0.001	0.000
	v2										0.000	0.000
	v3										0.000	0.000
Cq4n16ker0_seq	v1											0.000
	v2											0.000
	v3											0.000
Cq8n20ker3_seq	v1								0.000	0.000	0.000	0.000
	v2								0.000	0.000	0.000	0.000
	v3								0.000	0.000	0.000	0.000
CHadq4n16ker3_seq	v1								0.000	0.001	0.000	0.000
	v2								0.000	0.001	0.001	0.000
	v3								0.000	0.000	0.001	0.000
Cq2n20ker0_seq	v1											0.000
	v2											0.000
	v3											0.000
CHamq2n16ker4t_seq	v1							0.041	0.039	0.039	0.025	0.008
	v2							0.014	0.013	0.014	0.014	0.014
	v3							0.010	0.010	0.010	0.010	0.010
QaryCodeCker0q2	v1											0.000
	v2											0.000
	v3											0.000
CyclicCodeCker10q2	v1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	v2	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	v3	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Figura 12: Temps de càlcul dels algorismes de IsInformationSet per conjunts d'informació *false* per diferents dimensions del kernel

	ALGORISMES	10	9	8	7	6	5	4	3	2	1	0
UniverseCode(GF(3), 10)	v1	0.001	0.000	0.001	0.001	0.001	0.002	0.005	0.011	0.034	0.106	0.238
	v2	0.363	0.363	0.363	0.363	0.363	0.363	0.363	0.368	0.369	0.379	0.395
	v3	0.256	0.276	0.276	0.278	0.276	0.276	0.276	0.278	0.281	0.286	0.291
	v4	0.001	0.001	0.001	0.002	0.002	0.003	0.005	0.013	0.035	0.102	0.229
RepetitionCode(GF(4), 12)	v1										0.000	0.000
	v2										0.000	0.000
	v3										0.000	0.000
	v4										0.000	0.000
ZeroCode(GF(8), 13)	v1											0.000
	v2											0.000
	v3											0.000
	v4											0.000
Cq3n13ker8c_seq	v1			0.001	0.001	0.001	0.002	0.010	0.030	0.115	0.118	0.824
	v2			1.436	1.422	1.417	1.425	1.385	1.228	1.242	1.276	1.329
	v3			0.978	0.974	0.977	0.974	0.974	0.979	0.991	1.016	1.039
	v4			0.002	0.007	0.065	0.120	0.005	0.283	0.331	0.119	0.762
Cq3n13ker9a_seq	v1		0.001	0.000	0.001	0.001	0.002	0.005	0.033	0.131	0.134	0.834
	v2		1.221	1.221	1.228	1.229	1.227	1.230	1.242	1.256	1.289	1.340
	v3		0.980	0.980	0.982	0.984	0.984	0.982	0.984	0.993	1.023	1.057
	v4		0.003	0.002	0.002	0.002	0.037	0.119	0.015	0.247	0.130	0.787
CHadq4n16ker1b_seq	v1										0.001	0.001
	v2										0.001	0.002
	v3										0.002	0.002
	v4										0.001	0.001
Cq4n16ker0_seq	v1											0.004
	v2											0.005
	v3											0.006
	v4											0.006
Cq8n20ker3_seq	v1								0.000	0.000	0.000	0.000
	v2								0.000	0.000	0.000	0.000
	v3								0.000	0.000	0.000	0.001
	v4								0.000	0.000	0.000	0.000
CHadq4n16ker3_seq	v1								0.000	0.000	0.000	0.001
	v2								0.001	0.001	0.001	0.000
	v3								0.001	0.001	0.001	0.001
	v4								0.000	0.001	0.001	0.001
Cq2n20ker0_seq	v1											0.000
	v2											0.000
	v3											0.000
	v4											0.000
CHamq2n16ker4t_seq	v1							0.005	0.005	0.006	0.013	0.048
	v2							0.074	0.075	0.075	0.078	0.079
	v3							0.057	0.057	0.560	0.055	0.055
	v4							0.038	0.041	0.043	0.133	0.035
QaryCodeCker0q2	v1											0.000
	v2											0.000
	v3											0.000
	v4											0.000
CyclicCodeCker10q2	v1	0.000	0.000	0.000	0.001	0.000	0.001	0.001	0.001	0.002	0.003	0.004
	v2	0.006	0.006	0.007	0.006	0.006	0.007	0.007	0.006	0.007	0.007	0.007
	v3	0.005	0.005	0.005	0.005	0.005	0.005	0.005	0.005	0.005	0.005	0.005
	v4	0.000	0.001	0.000	0.002	0.001	0.002	0.001	0.002	0.002	0.003	0.003

Figura 13: Temps de càlcul dels algorismes de IsSystematic per diferents dimensions del kernel

	ALGORISMES	10	9	8	7	6	5	4	3	2	1	0
UniverseCode(GF(3), 10)	v1	0.000	0.000	0.000	0.000	0.000	0.000	0.005	0.015	0.045	0.100	0.235
	v2	0.000	0.005	0.000	0.000	0.000	0.000	0.005	0.010	0.035	0.105	0.235
	v3	0.365	0.365	0.365	0.370	0.370	0.370	0.365	0.365	0.375	0.385	0.400
RepetitionCode(GF(4), 12)	v1										0.005	0.000
	v2										0.000	0.000
	v3										0.000	0.005
ZeroCode(GF(8), 13)	v1											0.005
	v2											0.000
	v3											0.000
Cq3n13ker8c_seq	v1			0.120	0.280	0.820	1.790	5.025	9.860	24.855	48.455	84.050
	v2			0.570	0.720	1.450	2.555	6.505	12.080	29.285	56.950	74.435
	v3			122.085	122.240	122.285	122.145	122.420	122.845	124.245	127.260	132.395
Cq3n13ker9a_seq	v1		0.055	0.085	0.230	0.710	2.005	4.125	11.870	27.110	53.380	86.935
	v2		0.335	0.330	0.480	0.990	2.500	4.815	12.875	27.625	55.305	77.005
	v3		125.590	126.265	126.370	125.565	125.465	127.410	127.190	127.630	131.155	136.365
CHadq4n16ker1b_seq	v1										0.065	0.060
	v2										0.120	0.065
	v3										0.090	0.095
Cq4n16ker0_seq	v1											
	v2											
	v3											
Cq8n20ker3_seq	v1											
	v2											
	v3											
CHadq4n16ker3_seq	v1								0.005	0.045	0.055	0.060
	v2								0.015	0.080	0.065	0.065
	v3								0.090	0.085	0.090	0.100
Cq2n20ker0_seq	v1											
	v2											
	v3											
CHamq2n16ker4t_seq	v1							70.485	59.015	48.115	38.845	30.575
	v2							134.435	103.595	79.750	62.285	31.230
	v3							51.410	51.905	52.640	54.000	55.050
QaryCodeCker0q2	v1											
	v2											
	v3											
CyclicCodeCker10q2	v1	0.005	0.000	0.000	0.000	0.005	0.000	0.000	0.000	0.000	0.005	0.000
	v2	0.000	0.005	0.000	0.000	0.000	0.000	0.005	0.000	0.005	0.000	0.000
	v3	0.005	0.010	0.005	0.005	0.005	0.005	0.005	0.005	0.010	0.005	0.005

Figura 14: Temps de càlcul dels algorismes de AllInformationSets per diferents dimensions del kernel