INVARIANTS DE MATRIUS HADAMARD EN MAGMA

Memòria del projecte de final de carrera corresponent als estudis d'Enginyeria Superior en Informàtica presentat per Francesc Díez Aquilué i dirigit per Mercè Villanueva.

Bellaterra, 19 de Setembre de 2007

La firmant, Mercè Villanueva, professora del Departament d'Enginyeria de la Informació i de les Comunicacions de la Universitat Autònoma de Barcelona

CERTIFICA:

Que la present memòria ha estat realitzada sota la seva direcció per Francesc Díez Aquilué

Bellaterra, 19 de Setembre de 2007

Firmat: Mercè Villanueva

A la meva mare que ens va deixar enguany.

A la Núria, la Irene i l'Helena per la seva paciència.

Índex

Índex de figures

Capítol 1

Introducció

En el 1893, el "dépisté" Jacques Hadamard va trobar unes matrius quadrades d'ordres 12 i 20 amb valors ± 1 que tenien totes les seves files (i columnes) ortogonals. Aquestes matrius, $X = (x_{ij})$, satisfeien la igualtat de la següent desigualtat $|detX|^2 \leq \prod_{i=1}^{n} \sum_{j=1}^{n} |x_{ij}|^2$ i maximitzen el determinant. Malgrat que Hadamard es preguntava de fet quines matrius amb valors en el cercle unitari complex la satisfeien, el seu nom paradoxalment ha esdevingut definitivament associat als tipus de matrius reals que va trobar.

Hadamard no va ser pas el primer en estudiar aquestes matrius. J.J. Sylvester en el 1867 en el seu escrit titulat salvatgement (o pot ser hauria de dir "silvestrement") *"Thoughts on inverse orthogonal matrices, simultaneous sign-successions and tessellated pavements in two or more colours with application to Newton's rule, ornamental tile work and the theory of numbers"* troba aquestes matrius per tots els ordres potències de 2. Nogensmenys Hadamard conjectura que les matrius quadrades amb valors ± 1, ortogonals en files (i columnes), i màxim determinant podrien existir pels ordres 1, 2, i $4m$.

Així doncs la conjectura de Hadamard estableix que per tot n enter positiu i divisible per 4, existeix una matriu quadrada H d'ordre n, amb tots els seus valors ± 1, tal que $HH^T = nI$. Encara que pugui semblar que la conjectura

de Hadamard és un problema similar als més difícils problemes matemàtics per la simplicitat decebedora del seu plantejament, no obstant això és de naturalesa diferent. És diferent per les seves extenses conseqüències en molts camps de recerca, tal com la teoria del disseny, la teoria de la informació i la teoria de grafs. Mentre que el valor dels grans problemes matemàtics, com l'últim teorema de Fermat per exemple, rau en els subproductes matemàtics que han resultat de l'intent per resoldre'ls.

Quan el 14 de Novembre de 1971 el prototip espacial d'orbitador de la NASA Mariner 9, després de 5 mesos i mig de viatge, va arribar a Mart, va esdevenir la primera nau espacial en orbitar un altre planeta diferent de la Terra. Degut a una turmenta de pols, l'ordinador de la nau es va haver de reprogramar des de la Terra per tal de posposar un parell de mesos la retransmissió de les 7.329 imatges en blanc i negre d'una qualitat sorprenent del 80% de la superfície del planeta vermell des d'una alçada de $1.500Km$ durant 349 dies, una resolució de 1.000 a 100 metres per pixel, i amb 6 bits de dades per pixel que representaven 64 valors de l'escala de grisos.

El Mariner 9 és petit [1], mitja tona destinada quasi tota al potent sistema de propulsió per controlar la nau espacial en la orbita marciana. Un dels sis instruments que portava era l'aparell fotogràfic que, pels problemes de càrrega esmentats, havia d'incloure un transmissor petit, per tant la senyal transmesa havia de ser direccional. Però, en tant llargues distàncies, la senyal té problemes d'alineament. En la transmissió cada aproximadament 5 vegades la mida de les dades originals se havia de realinear el transmissor. A més a més, la senyal que arribava a la Terra era molt feble i havia d'amplificar-se, el soroll de l'espai afegit al soroll tèrmic de l'amplificador produïen errors en la transmissió.

La correcció d'errors en la transmissió d'imatges es va adequar als 30 bits de màxima longitud útil. Enlloc d'un fàcilment implementable codi de repetició de 5 bits que només en corregeix 2, es va utilitzar un codi Hadamard lineal $(32, 6, 16)$ que podia corregir fins a 7 bits per paraula codi. La proba-

[1] Esgotat i erm, encara dóna voltes fins al 2020 que entrarà en l'atmosfera marciana.

bilitat d'error en la imatge es reduïa només al 0.01% (el codi de repetició de 5 bits la tindria del 1%). Gràcies a la linealitat del codi Hadamard triat, resultava més econòmic en espai i pes, enlloc d'emmagatzemar les 64 paraules codi de 32 bits, dissenyar circuits que calculin les paraules codi mitjançant la matriu generadora del codi. A més a més, la matriu de Hadamard s'obtenia recursivament pel producte directe amb la matriu de Hadamard d'ordre 2 i els seus ordres potències de 2 eren anàlogues a les funcions de Walsh, per tant tot el tractament per ordinador és podia aconseguir utilitzant sumadors, que són més ràpids i fàcils d'implementar a nivell de *hardware*, enlloc de multiplicadors que són molt més lents.

A més de la utilització dels codis Hadamard lineals per a la detecció i correcció d'errors, el processament del senyal, el multiplexatge òptic, el disseny i anàlisis estadístics, ... [3], últimament s'estan usant amb tècniques de *fingerprinting* que permeten fer còpies autoritzades de música i video digital inserint un conjunt únic i diferent de marques que les fidel·litza.

De tots aquests antecedents esmentats en clau novel·lística es dedueix que un projecte que parli de les matrius Hadamard és prou interessant. I és per això que em vaig decidir a fer el PFC sobre matrius i codis de Hadamard dins del marc de la teoria de codis.

La implementació del projecte es realitza amb el paquet de software computacional algebraic MAGMA perquè ja té funcions per treballar amb matrius Hadamard, i perquè és un dels que empra el Grup de Combinatòria i Codificació, *CCG*, del Departament d'Enginyeria de la Informació i de les Comunicacions. Aquest projecte utilitzarà la infraestructura del grup per assegurar la fiabilitat tècnica, operativa i legal. I els resultats obtinguts podran ser utilitzats pel grup.

El projecte, els objectius del qual els descriuré a continuació, tracta de la construcció de noves matrius i codis Hadamard utilitzant dues invariants de tipus estructural o algebraic que són el *rang* i la *dimensió del nucli*.

L'any passat es van iniciar dos projectes en aquest sentit [10, 11]. L'un era per la construcció de matrius i codis de Hadamard de mida $n = 2^t$, i l'altre per la resta que són els de mida $n = 2^t \cdot s$, on $s \neq 1$ i senar. Aquest segón projecte també incluïa l'anàlisi del *rang* i la *dimensió del nucli* com invariants i veure el grau de classificació que aportaven en les matrius de Hadamard.

Aquest projecte és la continuació d'aquests en el sentit de generalitzar e integrar la funcionalitat dels dos projectes anteriors. Això ha comportat haver de crear noves funcions, modificar-ne d'altres, i també substituir funcions específiques per casos per una sola funció recursiva. També, de passada, analitzarem un nou invariant proposat pels xinesos Kai-Tai Fang i Gennian Ge.

1.1 Objectius

- Estudiar les propietats i característiques de les matrius i codis Hadamard, així com alguns dels seus invariants que ens ajuden a classificarles. Aprendre a utilitzar el MAGMA i conèixer el treball dels dos projectistes anteriors [10, 11] sobre la funcionalitat que ofereix el MAGMA per treballar amb matrius i codis Hadamard. Saber per quins ordres existeixen i quines matrius de Hadamard no equivalents conté la llibreria del MAGMA.

- Dissenyar e implementar funcions que construeixin totes les matrius de Hadamard no equivalents amb diferents rangs i dimensions del nucli per un ordre donat, generalitzant i racionalitzant algunes funcions desenvolupades en els projectes anteriors. Millorar la funció que calcula el nucli d'un codi binari: `KernelZ2(C)`, tot provant la seva fiabilitat i rapidesa.

- Analitzar la invariant 4-profile implementada en el MAGMA i les invariants *rang* i *dimensió del nucli* introduïts en els articles [8, 9]. Dissenyar, implementar i analitzar la invariant de Kai-Tai Fang i Gennian Ge en el seus articles [5, 4]. Comparar aquestes invariants esmentades sobre la base de dades de matrius Hadamard del MAGMA, i en algun cas, sobre un grup de matrius de Hadamard no equivalents d'altres procedències.

- Construir matrius de Hadamard inequivalents a les matrius Hadamard de la llibreria del MAGMA.

1.2 Contingut de la memòria

El contingut de la memòria està organitzat de la següent manera:

- Capítol 2. Fonaments teòrics: En aquest capítol rau la part teòrica en que es fonamenta aquest projecte. Matrius i codis de Hadamard, i diferents construccions de les matrius de Hadamard segons les dues invariants *rang* i *dimensió del nucli*. També farem una breu explicació de la invariant 4-profile i de la invariant de Kai-Tai Fang i Gennian Ge.

- Capítol 3. Planificació del projecte: En aquest capítol es detallen els objectius del projecte, els quals generen les tasques a realitzar. Sobre elles es fa una planificació temporal inicial segons els recursos disponibles.

- Capítol 4. Entorn de desenvolupament: En aquest capítol explicarem la màquina i la part del MAGMA que utilitzarem excepte el llenguatge: funcions i procediments, *packages* i les funcions del MAGMA per treballar amb les matrius de Hadamard.

- Capítol 5. Desenvolupament del projecte: En aquest capítol explicarem les funcions que hem implementat, els criteris que hem seguit per a la seva realització, com fer un *package* amb aquestes funcions i la realització dels tests per assegurar la correcció, la integritat, la fiabilitat i el rendiment de les funcions implementades.

- Capítol 6. Resultats: En aquest capítol es mostren els resultats obtinguts en l'assoliment dels objectius del projecte.

- Capítol 7. Conclusions: En aquest capítol resumim els resultats obtinguts, relacionem els objectius assolits del projecte, i es proposen ampliacions o noves línies de treball a seguir de cara al futur.

- Capítol 8. Handbook of MAGMA Functions: Hadamard matrices and codes. Aquest capítol en anglès conté el manual d'aquelles funcions que hem implementat, executables per l'usuari i d'acord amb la estructuració seguida pels manuals del MAGMA.

- Bibliografia.

- Apèndix A : CD amb codi font, tests, exemples i manual.

Capítol 2

Fonaments Teòrics

Aquesta secció està extreta pràcticament en la seva totalitat dels respectius fonaments teòrics dels dos projectes anteriors sobre matrius i codis Hadamard [10, 11]. Hem afegit la part teòrica corresponent a la invariant nova implementada, batejada com SHDE, proposada pels xinesos Kai-Tai Fang i Gennian Ge, i l'algorisme per a la construcció de matrius Hadamard d'ordre $n = 2^t \cdot s$ amb rang i dimensió de nucli.

2.1 Matrius de Hadamard

Una *matriu Hadamard* H d'ordre n és una matriu quadrada $n \times n$ amb valors ± 1 tal que $HH^T = nI$, on I és la matriu identitat del mateix ordre. En altres paraules, el producte escalar de qualsevol fila per ella mateixa és n i les files distintes són ortogonals. De $nH^{-1} = H^T$, tenim que $H^T H = nI$, per tant les columnes també són ortogonals i la transposada d'una matriu de Hadamard és també matriu Hadamard.

Els exemples de matrius Hadamard no binàries que mostrarem en aquest capítol, representem per simplicitat el valor -1 per $-$. Alguns exemples de matrius Hadamard són:

$$H_1 = \begin{pmatrix} - & 1 \\ - & - \end{pmatrix} \quad H_2 = \begin{pmatrix} - & 1 & 1 & 1 \\ 1 & 1 & - & 1 \\ - & - & - & 1 \\ - & 1 & - & - \end{pmatrix}$$

$$H_3 = \begin{pmatrix} 1 & 1 & - & 1 & 1 & 1 & - & 1 \\ 1 & - & - & - & - & 1 & 1 & 1 \\ 1 & 1 & - & 1 & - & - & 1 & - \\ - & - & - & 1 & - & - & - & 1 \\ 1 & 1 & 1 & - & - & - & - & 1 \\ 1 & - & - & - & 1 & - & - & - \\ - & 1 & - & - & - & 1 & - & - \\ 1 & - & 1 & 1 & - & 1 & - & - \end{pmatrix}$$

De l'ortogonalitat de les files (columnes) de les matrius de Hadamard es dedueix que qualsevol parella de files (columnes) coincideix i difereix en $n/2$ components, això fa necessari en principi que l'ordre d'una matriu de Hadamard, fora de l'ordre 1, sigui parell. Concriguraetant més, en [7] es demostra que si una matriu Hadamard H d'ordre n existeix, llavors n és 1, 2 o $4m$.

Dues matrius de Hadamard són equivalents si una pot obtenir-se de l'altre permutant i/o negant, files i/o columnes. Si utilitzem aquestes transformacions podem obtenir una matriu Hadamard equivalent, on la primera fila i columna són tot 1's, anomenada *normalitzada*. Les corresponents matrius normalitzades dels exemples anteriors són:

$$H_1' = \begin{pmatrix} 1 & 1 \\ 1 & - \end{pmatrix} \quad H_2' = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & - & - \\ 1 & - & 1 & - \\ 1 & - & - & 1 \end{pmatrix}$$

$$H_3' = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & - & - & - & - \\ 1 & 1 & - & - & 1 & 1 & - & - \\ 1 & 1 & - & - & - & - & 1 & 1 \\ 1 & - & 1 & - & 1 & - & 1 & - \\ 1 & - & 1 & - & - & 1 & - & 1 \\ 1 & - & - & 1 & 1 & - & - & 1 \\ 1 & - & - & 1 & - & 1 & 1 & - \end{pmatrix}$$

A partir d'ara, utilitzarem H' per denotar una matriu Hadamard normalitzada d'ordre n.

2.2 Codis Hadamard

Direm que $\mathbb{F}^n$ representa el conjunt de tots els vectors binaris de longitud n. La distància de Hamming entre dos vectors $x, y \in \mathbb{F}^n$, $d(x, y)$, és el número de coordenades on x i y difereixen. Un *codi binari* $C(n, M, d)$ és un subconjunt de $\mathbb{F}^n$ on $|C| = M$ i $d(c_l, c_2) \geq d$ per tota parella $c_l, c_2 \in C$. Els elements d'un codi s'anomenen *paraules codi* i d és la *distancia mínima*. Denotem $\langle C \rangle$ el subespai lineal i binari generat per C. Si $C = \langle C \rangle$, aleshores direm que el codi C és lineal.

Si substituïm els 1's per 0's i els -1's per 1's obtindrem la *matriu binària Hadamard*, que escriurem $c(H')$. A partir d'una matriu binària Hadamard es pot construir el codi Hadamard amb les files de la matriu binària i els seus complementaris. El *codi binari Hadamard* és $(n, 2n, n/2)$, és a dir, és un codi de longitud n, amb $2n$ paraules codi i distància mínima $n/2$.

Seguint amb els exemples proposats fins ara, obtenim les següents matrius binàries i els seus corresponents codis Hadamard:

$$c(H_1') = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \quad H_1 = \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{pmatrix}$$

$$c(H_2') = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \quad H_2 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

$$c(H_3') = \begin{pmatrix} 0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&1&1&1 \\ 0&0&1&1&0&0&1&1 \\ 0&0&1&1&1&1&0&0 \\ 0&1&0&1&0&1&0&1 \\ 0&1&0&1&1&0&1&0 \\ 0&1&1&0&0&1&1&0 \\ 0&1&1&0&1&0&0&1 \end{pmatrix} \quad H_3 = \begin{pmatrix} 0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&1&1&1 \\ 0&0&1&1&0&0&1&1 \\ 0&0&1&1&1&1&0&0 \\ 0&1&0&1&0&1&0&1 \\ 0&1&0&1&1&0&1&0 \\ 0&1&1&0&0&1&1&0 \\ 0&1&1&0&1&0&0&1 \\ 1&1&1&1&1&1&1&1 \\ 1&1&1&1&0&0&0&0 \\ 1&1&0&0&1&1&0&0 \\ 1&1&0&0&0&0&1&1 \\ 1&0&1&0&1&0&1&0 \\ 1&0&1&0&0&1&0&1 \\ 1&0&0&1&1&0&0&1 \\ 1&0&0&1&0&1&1&0 \end{pmatrix}$$

2.3 Construccions de matrius o codis Hadamard

2.3.1 A partir de codis Hamming

Un *codi* 1*-perfecte* C de longitud n és un subconjunt de $\mathbb{F}^n$, amb distància mínima $d = 3$, de manera que qualsevol vector de $\mathbb{F}^n$ es troba a una distància menor o igual que 1 d'una paraula codi. Per qualsevol $t > 1$ existeix exactament un codi 1-perfecte lineal de longitud $2^t - 1$ conegut com a *codi de Hamming*. El *codi estès* del codi C és un codi que resulta d'afegir un dígit de paritat per a cada paraula codi de C, per tant serà un codi de longitud $n = 2^t$. Donat un codi lineal C de longitud n, direm que el seu codi dual és $C^\perp = \{v \in \mathbb{F}^n | \forall x \in C, v \cdot x = 0\}$ i que també és lineal.

La construcció més simple d'una matriu Hadamard de mida $n = 2^t$ s'obté si considerem el codi dual d'un codi estès de Hamming. Per exemple, el dual del codi de Hamming estès de longitud 4, que és el codi lineal amb una matriu generadora,

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

és un codi Hadamard H. En aquest cas:

$$H = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \quad c(H') = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \quad H' = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & - & - \\ 1 & - & 1 & - \\ 1 & - & - & 1 \end{pmatrix}$$

2.3.2 Transposada

Com ja s'ha vist a l'apartat 2.1, en una matriu Hadamard les columnes tenen les mateixes propietats d'ortogonalitat que les files, $H^T H = nI$, per tant la transposada de qualsevol matriu Hadamard, H, és també una matriu Hadamard.

Per exemple, si utilitzem el MAGMA podem veure que com que la matriu `a` és de Hadamard, la seva transposada `at` també ho és.

```
> D:=HadamardDatabase();
> a:=Matrix(D,8,1);
> a;
[ 1  1  1  1  1  1  1  1]
[ 1  1  1  1 -1 -1 -1 -1]
[ 1  1 -1 -1  1  1 -1 -1]
[ 1  1 -1 -1 -1 -1  1  1]
[ 1 -1  1 -1  1 -1  1 -1]
[ 1 -1  1 -1 -1  1 -1  1]
[ 1 -1 -1  1  1 -1 -1  1]
[ 1 -1 -1  1 -1  1  1 -1]
> at:=Transpose(a);
> at;
```

```
[ 1  1  1  1  1  1  1  1]
[ 1  1  1  1 -1 -1 -1 -1]
[ 1  1 -1 -1  1  1 -1 -1]
[ 1  1 -1 -1 -1 -1  1  1]
[ 1 -1  1 -1  1 -1  1 -1]
[ 1 -1  1 -1 -1  1 -1  1]
[ 1 -1 -1  1  1 -1 -1  1]
[ 1 -1 -1  1 -1  1  1 -1]
> IsHadamard(at);
true
```

2.3.3 Producte de Kronecker

Les matrius Hadamard també es poden construir utilitzant el *producte de Kronecker*. Així és, si $H' = (hij)$ és una matriu Hadamard $n \times n$ i $B_1, B_2, \ldots, B_n$ són matrius Hadamard $k \times k$, la matriu que s'obté

$$H' \otimes [B_1, B_2, \ldots, B_n] = \begin{pmatrix} h_{11}B_1 & h_{12}B_1 & \cdots & h_{1n}B_1 \\ h_{21}B_2 & h_{22}B_2 & \cdots & h_{2n}B_2 \\ \vdots & \vdots & \vdots & \vdots \\ h_{n1}B_n & h_{n2}B_n & \cdots & h_{nn}B_n \end{pmatrix}$$

és d'ordre $nk \times nk$ i és, per construcció, també matriu Hadamard [8]. Si $B_1 = \cdots = B_n = B$, es pot escriure $H' \otimes [B_1, B_2, \ldots, B_n] = H' \otimes B$.

Tot seguit s'expliquen les formes concretes on utilitzarem el producte de Kronecker en el projecte.

Producte de Kronecker amb matrius Sylvester

Considerem que S és la matriu Hadamard $\begin{pmatrix} 1 & 1 \\ 1 & - \end{pmatrix}$.
Començant des d'una matriu Hadamard S_0 podem definir recursivament S_t per a $t \geq 1$, amb l'equació recurrent $S_t = S \otimes [S_{t-1}, S_{t-1}] = S \otimes S_{t-1}$. Agafem $S_0 = (1)$, la successió corresponent S_1, S_2, S_3, ..., S_t, ens dóna matrius Hadamard de totes les mides que són potències de dos, concretament

$$S_0 = (1) \quad S_1 = S \otimes S_0 = \begin{pmatrix} 1 & 1 \\ 1 & - \end{pmatrix} \quad S_2 = S \otimes S_1 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & - & 1 & - \\ 1 & 1 & - & - \\ 1 & - & - & 1 \end{pmatrix}$$

$$S_2 = H' \quad c(H') = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \quad H = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

la matriu Hadmard S_t és de mida $n = 2^t$. Aquestes matrius s'anomenen *matrius Sylvester*. Sabem que el codi binari d'aquestes matrius Hadamard, S_t, és el dual del codi de Hamming estès de longitud $n = 2^t$. Per tant són equivalents als codis Hadamard vistos a l'apartat 2.3.1 A l'exemple següent es pot veure com el codi obtingut és equivalent a l'obtingut a l'exemple de l'apartat 2.3.1.

Producte de Kronecker amb dues matrius Hadamard

El producte de Kronecker, com ja hem vist, es pot utilitzar de formes diferents, ja sigui amb una o més matrius de Hadamard. Així doncs, aquesta construcció, concretament, consisteix en fer el producte de Kronecker amb la matriu S i dues matrius (diferents) de la mateixa mida n, és a dir $S \otimes [B_1, B_2]$.

Sabem que si es permutem les columnes d'una matriu Hadamard, continua essent una matriu Hadamard, per tant un cas particular de la construcció anterior que també s'utilitzarà, consisteix en fer el producte de Kronecker amb la matriu S i dues matrius Hadamard B_1 i $\pi(B_2)$, on π és una permutació de n coordenades i $\pi(B_2)$ és la matriu B_2 amb les columnes permutades, és a dir $S \otimes [B_1, \pi(B_2)]$.

2.3.4 Tècnica del *Switching*

Podem construir matrius Hadamard fent servir la tècnica del *Switching*. Aquesta tècnica consisteix en agafar un subconjunt S_0 de vectors d'un codi i intercanviar-lo per un nou subconjunt S_1. Explicat de forma matemàtica seria, $(C \backslash S_0) \cup S_1$.

En el cas concret de les matrius Hadamard construïdes amb aquesta tècnica, es pot triar que el subconjunt del codi Hadamard a substituir sigui un traslladat d'un subespai de dimensió 3 del nucli , S_0 en aquest cas. A més, també es pot triar que el nou subconjunt, S_1 sigui la suma per cada vector pertanyent a S_0 amb $v_1 v_2$, on v_1 i v_2 són els dos vectors base del nucli diferents al vector tot uns, $(1, 1, \cdots, 1)$ i $v_1 v_2$ $(= v_1$ and $v_2)$ representa el vector amb 1's a les coordenades on els dos vectors tenen un 1 i 0 a la resta de coordenades. Per exemple, si C és un codi Hadamard amb nucli $K(C) = \langle 1, v_1, v_2 \rangle$ on

$$\begin{aligned} v_1 &= (\underbrace{1,1,1,1,1,...,1}_{n/2}, \underbrace{0,0,0,0,0,...,0}_{n/2}), \\ v_2 &= (\underbrace{1,1,...,1}_{n/4}, \underbrace{0,...,0}_{n/4}, \underbrace{1,...,1}_{n/4}, \underbrace{0,...,0}_{n/4}), \\ v_1 v_2 &= (\underbrace{1,1,...,1}_{n/4}, \underbrace{0,0,0,0,0,0,...,0,0,0}_{3n/4}), \end{aligned} \tag{2.1}$$

, aleshores $C \backslash (K(C) + x) \cup (K(C) + x + v_1 v_2)$ és també un codi Hadamard en aquest cas concret.

A la Figura 2.1 es mostra de forma gràfica la tècnica del Switching.

2.4 Invariants: 4-profile

El problema per identificar l'equivalència entre dues matrius Hadamard és un problema NP-*hard* ja que es podrien arribar a fer $(2^n n!)^2$ comparacions. La invariant com a condició necessària (però no suficient) en l'equivalència entre matrius Hadamard, serveix per identificar les matrius Hadamard ine-

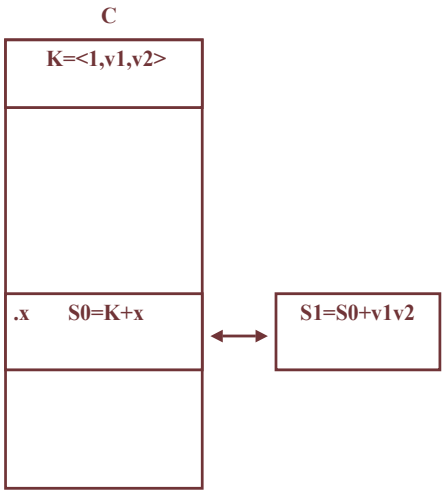

Figura 2.1: Switching

quivalents. A [5] es proposa una invariant més sensible que pot servir per identificar dues matrius Hadamard inequivalents, ja que si dues matrius Hadamard són equivalents tenen el mateix 4-profile, 1'invers, però, no és cert. Aquesta invariant és e1 4-*profile*.

Suposem que $H' = (h_{ij})$ és una matriu Hadamard de mida $n \geq 8$. Es defineix

$$p_{ijkl} = \left| \sum_{x=1}^{n} h_{ix} h_{jx} h_{kx} h_{lx} \right| \tag{2.2}$$

Diem que $\pi(m)$ és el numero de conjunts, $\{i, j, k, l\}$, de quatre files diferents on $m = p_{ijkl}$, $\pi(m)$ és el 4-profile d'H. De forma similar es pot definir el 6-profile, el 8-profile, etc.

La complexitat d'un algorisme per calcular aquesta invariant, 4-profile és: $kn\binom{n}{k}$. El MAGMA implementa aquest algorisme per calcular el 4-profile amb la funció `HadamardInvariant(H)`.

Tot seguit es mostra un exemple de com calcular el 4-profile i el resultat que se n'obté amb el MAGMA.

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

$$p_{1234} = |\sum_{x=1}^{8} h_{1x}h_{2x}h_{3x}h_{4x}| = |(h_{11}h_{21}h_{31}h_{41}) + (h_{12}h_{22}h_{32}h_{42}) + (h_{13}h_{23}h_{33}h_{43}) + \cdots + (h_{18}h_{28}h_{38}h_{48})| = |81| = 8 \tag{2.3}$$

$$p_{1235} = |\sum_{x=1}^{8} h_{1x}h_{2x}h_{3x}h_{5x}| = |(h_{11}h_{21}h_{31}h_{51}) + (h_{12}h_{22}h_{32}h_{52}) + (h_{13}h_{23}h_{33}h_{53}) + \cdots + (h_{18}h_{28}h_{38}h_{58})| = |0| = 0 \tag{2.4}$$

$$p_{1236} = |\sum_{x=1}^{8} h_{1x}h_{2x}h_{3x}h_{6x}| = |(h_{11}h_{21}h_{31}h_{61}) + (h_{12}h_{22}h_{32}h_{62}) + (h_{13}h_{23}h_{33}h_{63}) + \cdots + (h_{18}h_{28}h_{38}h_{68})| = |0| = 0 \tag{2.5}$$

etc.

Es segueix així amb totes les combinacions possibles de 4 files i al final s'han de sumar el nombre de 8's i 0's que han sortit, el resultat coincideix amb l'obtingut amb el MAGMA.

```
> D:=HadamardDatabase();
> a:=Matrix(D,8,1);
> a;
[ 1  1  1  1  1  1  1  1]
[ 1  1  1  1 -1 -1 -1 -1]
[ 1  1 -1 -1  1  1 -1 -1]
[ 1  1 -1 -1 -1 -1  1  1]
[ 1 -1  1 -1  1 -1  1 -1]
[ 1 -1  1 -1 -1  1 -1  1]
[ 1 -1 -1  1  1 -1 -1  1]
[ 1 -1 -1  1 -1  1  1 -1]
```

```
> HadamardInvariant(a);
[ 56, 14 ]
```

Comprovem que $56 + 14 = 70$ que són les combinacions de 8 elements agafats de 4 en 4, $\binom{8}{4}$.

2.5 Invariants: SHDE

Això és un resum dels fonaments teòrics necessaris per la presentació d'aquesta invariant extret de l'article dels xinesos Kai-Tai Fang i Gennian Ge [5]. Es defineix la *distància simètrica de Hamming* entre dos files qualsevol com el menor entre el número de posicions amb el mateix valor i diferent valor. Per exemple la distància simètrica de Hamming entre les files $(1, 1, -, -, 1, 1, -, -)$ i $(1, -, 1, -, -, -, 1, 1)$ és 2, mentre que la *distància de Hamming* és 6. De la definició es dedueix que el valor màxim de la distància simètrica de Hamming és la meitat de la longitud de les files.

Per la detecció de la inequivalència entre matrius de Hadamard, definim ara un conjunt de magnituds que són funcions de les distàncies simètriques de Hamming i les distàncies simètriques de Hamming projectades d'una matriu Hadamard. Donada una matriu Hadamard H d'ordre n, sigui $S_i(H)$ el número de parelles de files distintes que la distància simètrica de Hamming és i. Ens referirem a $(S_0(H), ..., S_{n/2}(H))$ com la *distribució de distància* de H. Definim

$$B_a(H) = \sum_{i=0}^{n/2} S_i(H)(a^i + a^{n-i}) \tag{2.6}$$

com l'enumerador de distància de H, on a és un número positiu.
Donada una $k(1 \leq k \leq n)$, definim

$$B_a(H^K) = \sum_{i=0}^{k/2} S_i(H^K)(a^i + a^{k-i}) \tag{2.7}$$

com el valor B_a sobre una projecció de k columnes, on $S_i(H^K)$ és la distribució de distància de H^K (només considerem la distància simètrica de Hamming

de k columnes de les files de H).

La distància simètrica està íntimament lligada al producte escalar. Sigui K un subconjunt de k elements de $X = \{1, 2, \ldots, n\}$. Sigui u_K, v_K la i-èssima i la j-èssima fila de H^K respectivament, aleshores:

$$\langle u_K, v_K \rangle = \sum_{c \in K} h_{ic} h_{jc}$$

.

Sigui d la distància simètrica de u_K i v_K. Quan d és el número de posicions on u_K i v_K són idèntics, $k - d$ és el número de posicions on u_K i v_K són diferents. I també a l'inrevés, quan d és el número de posicions on u_K i v_K són diferents, $k - d$ és el número de posicions on u_K i v_K són idèntics. En qualsevol cas, $k - 2d = |\langle u_K, v_K \rangle|$. Per tant, $d = (k - |\langle u_K, v_K \rangle|)/2$. En $B_a(H^K)$, podem substituir a^i i a^{k-i} per $a^{(k-|\langle u_K, v_K \rangle|)/2}$ i $a^{(k+|\langle u_K, v_K \rangle|)/2}$ respectivament. Això redueix el càlcul de $B_a(H^K)$, doncs és més fàcil de calcular el producte escalar que la distància simètrica i l'equació 2.7 ara queda

$$B_a(H^K) = \sum_{i=0}^{k/2} S_i(H^K)(a^d + a^{k-d}). \tag{2.8}$$

Donada una k ($1 \leq k \leq n$), definim $F_{B_{a,k}}(H)$ com la distribució de valors Ba sobre totes les projeccions de k columnes. Resulta que $F_{B_{a,k}}(H)$ és un invariant a les permutacions i negacions de files i columnes. Per tant, a continuació aquest teorema estableix la condició necessària per la equivalència de matrius Hadamard.

Teorema 2.5.1 *Si les matrius de Hadamard H_1 i H_2 són equivalents, aleshores $F_{B_{a,k}}(H_1) = F_{B_{a,k}}(H_2)$ per $k = 1, 2, \ldots, n/2$.*

Del Teorema 2.5.1 podem concloure que H_1 i H_2 són inequivalents si per alguna k tenim que $F_{B_{a,k}}(H_1) \neq F_{B_{a,k}}(H_2)$.

Els següents dos lemes que veurem, ens poden estalviar molt cost computacional en la classificació de les matrius de Hadamard.

Lema 2.5.2 *Les relacions* $F_{B_{a,k}}(H_1) = F_{B_{a,k}}(H_2)$, $k = 1, 2$. *es compleixen per qualsevol matrius de Hadamard* H_1 *i* H_2 *del mateix ordre.*

Donat que dos files qualsevol d'una matriu Hadamard d'ordre n són ortogonals, tenim que la distància de Hamming entre elles és $n/2$. Notem que el residu d'una projecció de k columnes de H correspon a una projecció $n-k$ columnes de H i a l'inrevés. Per tant, tenim:

Lema 2.5.3 *Siguin* H_1 *i* H_2 *dos matrius de Hadamard d'ordre* n. *Per qualsevol* k $(1 \leq k \leq n)$ $F_{B_{a,k}}(H_1) = F_{B_{a,k}}(H_2)$ *si i només si* $F_{B_{a,n-k}}(H_1) = F_{B_{a,n-k}}(H_2)$.

A partir del Teorema 2.5.1 i els Lemes 2.5.2 i 2.5.3, es proposa el següent algorisme per detectar matrius Hadamard inequivalents, on s'assigna un número irracional pel paràmetre a.

Un algorisme per detectar matrius de Hadamard inequivalents
Pas 1. Posem $k = 4$.
Pas 2. Comparem $F_{B_{a,k}}(H_1)$ amb $F_{B_{a,k}}(H_2)$. Si no són iguals anem al pas 4.
Pas 3. Si $k = n/2$ hem fracasat intentant trobar la inequivalència entre H_1 i H_2 i sortim. Sinó posem $k = k + 2$ i tornem al pas 2.
Pas 4. Conclourem que H_1 i H_2 no són equivalents i hem acabat.

Comentaris:

- Aquest algorisme quan fracassa vol dir que no sap si H_1 i H_2 són inequivalents o equivalents.

- Comencem des de $k = 4$ enlloc de començar des de $k = 3$ e incrementem $k = k+2$ enlloc de $k = k+1$, perquè des de l'experiència computacional es descobreix que el càlcul de $F_{B_{a,2i+1}}(H)$ quasi té el mateix efecte que el de $F_{B_{a,2i}}(H)$ a l'hora de distingir les matrius Hadamard inequivalents.

2.6 Invariants: Rang i Nucli

Dues propietats estructurals més dels codis no-lineals són el rang i el nucli (en anglès, rank i kernel).

El rang d'un codi binari C de longitud n, rank(C), és simplement la dimensió del subespai lineal generat per C. El nucli d'un codi binari C és

$$K(C) = \{x \in \mathbb{F}^n \mid x + C = C\}$$

Si el vector tot zero pertany al codi, és a dir, si $(0, 0, \cdots, 0) \in C$ llavors $K(C)$ és un subespai lineal de C. Direm que $ker(C)$ és la dimensió de $K(C)$. En general, C es pot escriure com una unió de cosets de $K(C)$, i $K(C)$ és el codi lineal més gran pel qual això és cert.

En el pitjor dels casos, la complexitat dels algorismes utilitzats per calcular el rang i la dimensió del nucli és equivalent a resoldre un sistema d'equacions, per tant és d'ordre n^3. Tot i això, és possible que la funció `Dimension()` implementada pel MAGMA sigui més eficient.

Aquestes dues invariants també poden servir per identificar dues matrius Hadamard inequivalents, ja que si dues matrius Hadamard són equivalents els seus respectius codis Hadamard tenen el mateix rang i dimensió del nucli.

2.6.1 Rang dels codis Hadamard

Com es demostra a [8] podem dir que existeix un codi Hadamard de longitud $n = 2^t$, amb rang r, $\forall r \in \{t+1, \cdots, n/2\}$. Per a codis Hadamard de longitud $4s$ i $8s$ on $s > 1$ i senar, se sap que el seu rang és $4s - 1$ i $4s$, respectivament [9]. Per la resta de longituds, si existeix un codi Hadamard de longitud $4s$, amb $s > 1$ i senar, per tot $t \geq 3$, llavors existeix un codi Hadamard de longitud $n = 2^t.s$ amb rang r, $\forall r \in \{4s + t - 3, \cdots, n/2\}$ [9].

Per exemple, els codis Hadamard de longitud $n = 12$ tenen rang 11, els de longitud $n = 24$ tenen rang 12 i els de longitud $n = 2^4 3 = 48$ se sap que tenen rang $r \in \{13, 14, \cdots, 24\}$.

2.6.2 Dimensió del nucli dels codis Hadamard

A [8] queda demostrat que la dimensió del nucli d'un codi Hadamard de longitud $n = 2^t$ és $k \in \{1, 2, \cdots, t-1, t+1\}$. A més, existeix un codi Hadamard de longitud $n = 2^t$ per cada possible dimensió del nucli.

Per aquests codis Hadamard de longitud 2^t, sempre n'existeix un de lineal amb dimensió $t+1$, S_t. Es pot assumir que S_t està generat pels vectors binaris $\mathbf{1}, v_l, v_2, \cdots, v_t$ de longitud 2^t, on els vectors v_i, $\forall i \in \{1, \cdots, t\}$, són:

$$\begin{aligned} v_1 &= (\underbrace{1,1,1,1,1,...,1}_{n/2}, \underbrace{0,0,0,0,0,...,0}_{n/2}), \\ v_2 &= (\underbrace{1,1,...,1}_{n/4}, \underbrace{0,...,0}_{n/4}, \underbrace{1,...,1}_{n/4}, \underbrace{0,...,0}_{n/4}), \\ &\vdots \\ v_t &= (\underbrace{1,...,1}_{n/2^t}, \underbrace{0,...,0}_{n/2^t}, \underbrace{1,...,1}_{n/2^t}, ..., \underbrace{0,...,0}_{n/2^t}). \end{aligned} \tag{2.9}$$

En general, si $n = 2^t \cdot s$, es poden considerar els vectors $\mathbf{1}, v_1, v_2, \ldots, v_t$ de longitud $2^t \cdot s$ construïts de la mateixa manera. Se sap que no sempre $\langle \mathbf{1}, v_1, v_2, \ldots, v_t \rangle \subseteq \langle H \rangle$ [9], però si un codi Hadamard H té $ker(H) = k$ es pot veure que el nucli esta generat per k vectors (independents) de $\mathbf{1}, v_1, v_2, \ldots, v_t$, per tant podem assumir que $K(C) = \langle \mathbf{1}, v_1, v_2, \ldots, v_{k-1} \rangle$.

Un codi Hadamard de longitud $n = 2^t \cdot s$ ($t \geq 2$), on $s > 1$ i senar, té dimensió del nucli k, on $k \in \{1, 2, \ldots, t-1\}$. A més si existeix un codi Hadamard de longitud $4s$, amb $s > 1$ i senar, per tot $t \geq 3$ existeix un codi Hadamard de longitud $n = 2^t \cdot s$ amb nucli de dimensió k, $\forall k \in \{1, \ldots, t-1\}$ [9].

Per exemple, se sap que els codis Hadamard de longitud $n = 2^4.3 = 48$ tenen dimensió del nucli $k \in \{1, 2, 3\}$.

2.6.3 Rang i nucli dels codis Hadamard

Un cop definits i donats els límits del rang i el nucli per codis Hadamard, tant si són de longitud $n = 2^t$ com de longitud $n = 2^t \cdot s$, es donaran els límits superiors del rang, en funció de la dimensió del nucli.

Per un codi Hadamard de longitud $n = 2^t$, apart del codi lineal que té rang $r = t + 1$ i dimensió del nucli $k = t + 1$ per qualsevol longitud n (per $t < 4$ només existeix un codi de Hadamard), per $t = 4$ les combinacions de rang i dimensió del nucli són aquestes: $r = 6$ i $k = 3$, $r = 7$ i $k = 2$, $r = 8$ i $k = 2$ i $r = 8$ i $k = 1$. Per $t > 4$ les combinacions de rang i dimensió del nucli són aquestes [8]:

$$\begin{cases} t + 2 \leq r \leq 2^{t+1-k} + k - 1 & \text{si} \quad 3 \leq k \leq t - 1 \\ t + 3 \leq r \leq 2^{t-1} & \text{si} \quad 1 \leq k \leq 2 \end{cases} \tag{2.10}$$

I per un codi Hadamard (no lineal) de longitud $n = 2^t \cdot s$ ($t \geq 3$), on s és senar, amb rang r i dimensió del nucli k es compleix:

$$r \leq \begin{cases} 2^{t+1-k} \cdot s + k - 1 & \text{si} \quad 3 \leq k \leq t - 1 \\ 2^{t-1} \cdot s & \text{si} \quad 1 \leq k \leq 2 \end{cases}$$

Els límits inferiors exactes del rang, en funció de la dimensió del nucli, només es coneixen per codis Hadamard de longitud $n = 2^t$ (desigualtats 2.10). Per a codis Hadamard de longitud $n = 2^t \cdot s$, on $s > 1$ i senar no es coneixen, però a [9] també es demostra que existeix un codi Hadamard amb rang r i dimensió del nucli k sempre que es compleixi:

$$4s + t - 3 \leq r \leq \begin{cases} 2^{t+1-k} \cdot s + k - 1 & \text{si} \quad 3 \leq k \leq t - 1 \\ 2^{t-1} \cdot s & \text{si} \quad 1 \leq k \leq 2 \end{cases} \tag{2.11}$$

A la Taula 2.1 es mostren totes les combinacions possibles de rang i dimensió del nucli per a codis Hadamard de longitud $n = 2^4 \cdot 3 = 48$ ($t =$

$4, s = 3$). Els símbols que apareixen representen les diferents construccions de les matrius Hadamard que generen aquest codis.

Taula 2.1: Combinacions de rang i dimensió del nucli per a $n = 48$.

	$rank(C)$											
$ker(C)$	13	14	15	16	17	18	19	20	21	22	23	24
3	⋆	•										
2	⋆	◇	◇	◇	◇	◇	◇	◇	◇	◇	◇	◇
1	○	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗

2.6.4 Construcció de matrius Hadamard d'ordre $n = 2^t \cdot s$ amb rang i dimensió de nucli

L'algorisme, en funció de t, construeix les matrius de Hadamard d'ordre $n = 2^t \cdot s$ ($t \geq 2$ i $s \neq 1$ senar) amb un rang r i dimensió de nucli k donats. Les construccions que s'utilitzen es troben en la referència [9]. En aquest algorisme denotem amb $H^n_{r,k}$ una matriu Hadamard d'ordre n, rang r i dimensió del nucli k.

- Si $t = 2$, existeix una única matriu de Hadamard d'ordre $n = 4s$ tal que el seu codi té rang $r = 4s - 1$ i dimensió de nucli $k = 1$. Aquesta matriu Hadamard, $H^{4s}_{4s-1,1}$, caldrà construir-la o trobar-la en alguna base de dades, sinó no podrem construir cap matriu de la seva seqüència $8s, 16s, 32s, ...$

- Si $t = 3$, existeixen dues matrius Hadamard d'ordre $n = 8s$ amb el mateix rang $r = 4s - 1$ però diferent dimensió de nucli $k = 1, 2$, que es poden construir de la manera següent:

  1. $H^{8s}_{4s-1,1} = S \otimes [H^{4s}_{4s-1,1}, \pi(H^{4s}_{4s-1,1})]$ on π és una permutació de 2 columnes linealment independents.
  2. $H^{8s}_{4s-1,2} = S \otimes [H^{4s}_{4s-1,1}, H^{4s}_{4s-1,1}]$.

- Si $t > 3$ denotem amb $rmin = 4 \cdot s + t - 3$ el rang mínim, i les construccions de les matrius de Hadamard es poden dividir en 5 casos:

  1. Si $r = rmin$ i $k = 1$ aleshores

  $$H^n_{rmin,1} = S \otimes [H^{n/2}_{rmin-1,1}, H^{n/2}_{rmin-1,2}].$$

  2. Si Existeix $H^{n/2}_{r-1,k-1}$ aleshores

  $$H^n_{r,k} = S \otimes H^{n/2}_{r-1,k-1}.$$

  3. Si $r \geq (n/4) + 2$ i $k = 2$ aleshores

  $$H^n_{r,2} = (HadamardNormalize(H^n_{r,1}))^T.$$

  4. Si $r = (n/4)+2$ i $k = 3$ llavors utilitzem el *switching*: Es comença amb el codi C de la matriu de Hadamard $H^n_{r-1,3}$, que té, evidentment, longitud n, rang $r - 1$ i dimensió de nucli $dim(K) = 3$. Transformen la base del nucli per tal que un dels vectors base sigui el tot uns, és a dir, $K = \langle \mathbf{1}, v1, v2 \rangle$. Sigui $x \in C, x \notin K$ i considerem el traslladat de K, $K + x$ inclòs en C. Denotem per L el codi C on s'ha substituït el conjunt de paraules codi $K + x$ per $K + x + v1v2$ amb rang $r = (n/4) + 2$ i dimensió de nucli $k = 3$. llavors $H^n_{r,3}$ és la matriu de Hadamard del codi L.

  5. Si $k = 1$ llavors utilitzem les permutacions

     - Si $r \geq rmin + 1$ i $r < (n/4) + 2$ aleshores

     $$H^n_{r,1} = S \otimes [H^{n/2}_{r-2,1}, \pi(H^{n/2}_{r-2,1})]$$

     on π és una permutació de 2 columnes linealment independents.

– Si $r \geq (n/4) + 2$ aleshores

$$H^n_{r,1} = S \otimes [H^{n/2}_{n/4,1}, \pi(H^{n/2}_{n/4,1})]$$

on π és una permutació de $n/4$ columnes linealment independents.

Capítol 3

Planificació del projecte

En aquest capítol es detallaran els objectius que s'han proposat en la introducció, i a partir d'aquesta relació detallada generem les tasques a realitzar. Sobre les tasques i segons els recursos disponibles farem una planificació temporal inicial que es veurà corregida finalment per la planificació temporal final.

3.1 Objectius

Tal com hem dit, anem a marcar primer els diferents objectius que es volen assolir. Els objectius d'aquest projecte són els següents:

1. Estudiar les matrius i codis Hadamard.
2. Estudiar les invariants 4-profile, rang i dimensió del nucli.
3. Estudiar la invariant SHDE dels xinesos Kai-Tai Fang i Gennian Ge.
4. Estudiar el MAGMA.
5. Estudiar els treballs dels dos projectistes anteriors.

6. Implementar la llibreria amb les noves funcions actualitzant la versió anterior, això també inclou, en especial, millorar i realitzar les proves de rendiment de la funció KernelZ2(C), i implementar la invariant SHDE.

7. Implementar el *package* de la llibreria anteriorment esmentada.

8. Actualitzar el manual incorporant les funcions creades o modificades i eliminant les obsoletes.

9. Actualitzar els test de proves i els exemples de les noves funcions.

10. Analitzar les invariants 4-profile, rang i dimensió del nucli, i SHDE sobre les matrius de Hadamard de la base de dades del MAGMA.

11. Analitzar les invariants 4-profile, rang i dimensió del nucli, i SHDE sobre algunes matrius de Hadamard inequivalents proporcionades per Ilias Kotsireas i Cristos Koukouvinos.

12. Ampliar la base de dades del MAGMA amb la construcció de noves matrius de Hadamard utilitzant les noves funcions genèriques implementades i tenint en compte el rang i la dimensió del nucli.

13. Redactar la memòria.

3.2 Tasques a realitzar

- Tasca 1: Estudi de les matrius i codis Hadamard i els diferents constructors que utilitzarem per crear-ne de noves, així com les invariants 4-profile, rang i dimensió del nucli i la invariant SHDE de Kai-Tai Fang i Gennian Ge.

- Tasca 2: Estudi del MAGMA. És un requeriment del departament, el llenguatge d'aquest sistema algebraic simbólic és en el que codifiquem les funcions. Després farem el muntatge de la llibreria amb aquestes funcions, empaquetant-les en els *packages* del MAGMA.

Figura 3.1: Llegenda de colors de les tasques

- Tasca 3: Estudi del treball dels projectistes anteriors. Doncs aquest treball és la seva continuació.
- Tasca 4: Implementació de les funcions (llibreria i *package*). Establerts els criteris de disseny, escriure el codi de les funcions i empaquetar-les.
- Tasca 5: Proves de rendiment de la funció revisada `KernelZ2(C)`.
- Tasca 6: Test de proves. Per tal de comprovar la fiabilitat de les funcions implementades i la seva integració en la llibreria.
- Tasca 7: Analitzar les invariants 4-profile, rang i dimensió del nucli, i la invariant SHDE; tant sobre la base de dades del MAGMA com sobre algunes matrius de Hadamard inequivalents de Ilias Kotsireas i Cristos Koukouvinos.
- Tasca 8: Obtenir noves matrius Hadamard per tal d'ampliar la base de dades del MAGMA utilitzant les funcions implementades.
- Tasca 9: Redacció del manual i realització dels exemples.
- Tasca 10: Redacció de la memòria.

En la Figura 3.1 mostrem la taula del codi de colors de les diferents tasques.

3.3 Planificació inicial i final

3.3.1 Planificació temporal inicial

En la següent Figura 3.2 es mostra la taula de la planificació temporal que es va fer en un principi. Enguany, degut al calendari acadèmic, vam començar

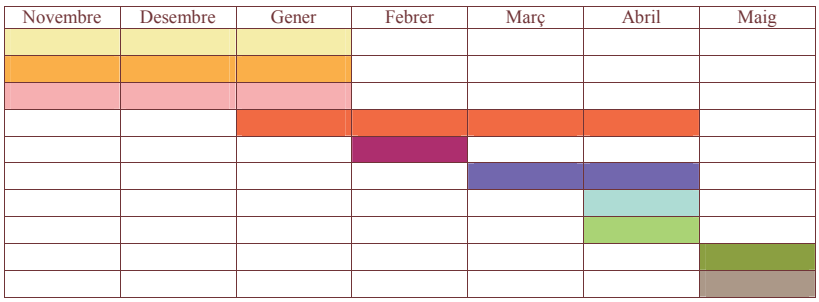

Figura 3.2: Planificació temporal inicial.

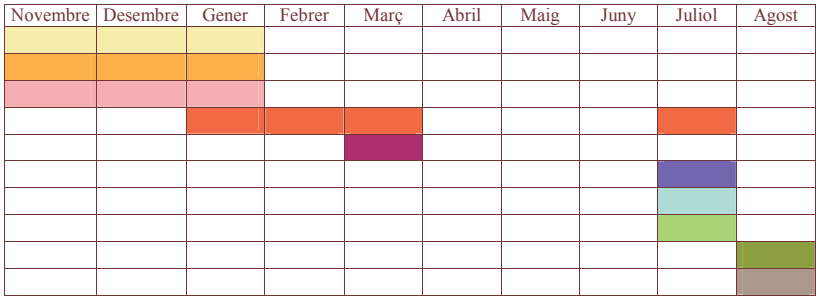

Figura 3.3: Planificació temporal final.

a principis de Novembre.

3.3.2 Planificació temporal final

La Figura 3.3 mostra la taula de la temporització que al final s'ha dut a terme.

S'han produït bàsicament tres canvis respecte a la planificació temporal inicial:

- Un fet greu familiar va provocar que els mesos d'Abril a Maig no poqués dedicar-me gaire bé gens al projecte o, al menys, amb la intensitat que calia.

- Les proves de rendiment de la funció `KernelZ2(C)` es desplaça a un mes més tard quan la implementació de les funcions està més avançada, i

ja hem passat de la primera versió millorada de la funció `KernelZ2(C)` amb retorn del nucli en tipus llista, a la versió definitiva amb retorn tipus espai vectorial.

- El test de proves també es desplaça a un mes més tard, i a més, s'escurça a un sol mes. Això és degut a la reducció del nombre de funcions en la versió millorada de la llibreria, el test és menys extensiu, però més intensiu en les poques funcions que hem de provar unitàriament.

Capítol 4

Entorn de desenvolupament

4.1 Sistema de computació simbòlica MAGMA

El MAGMA és un sistema d'àlgebra dissenyat per proporcionar un entorn *software* a la computació amb estructures, que poden aparèixer en àrees com l'àlgebra, la teoria dels números, la geometria algebraica i la combinatòria. També el MAGMA permet als usuaris definir altres estructures com grups, anells, ..., **codis** en el nostre cas. Algunes característiques importants del MAGMA són:

- Intenta aproximar-se tant com pot al tipus de pensament i notació matemàtica habituals, per tal de proporcionar un entorn matemàtic rigurós.
- Té tipus explícits, és a dir, l'usuari ha de definir explícitament la major part d'estructures algebraiques on ha de tenir lloc un càlcul.
- Té un gran *kernel* escrit en *C* per aconseguir eficiència, i una cada vegada més extensa llibreria de funcions empaquetades, programades en el llenguatge MAGMA, per aconseguir més funcionalitat.
- Té un llenguatge que és imperatiu, la crida a funcions per valor i dinàmicament tipat.

- Té un interpret que permet els càlculs interactius.
- Disposa d'un gran nombre de bases de dades matemàtiques que contenen informació que pot ser útil.

4.2 Estructura del MAGMA

La versió del *software* que utilitzarem nosaltres és la MAGMA 2.13-12. El MAGMA està instal·lat en la màquina anomenada *MacWilliams*, l'accés a la qual és remot. Per exemple en el meu cas, l'accés es feia des de casa o des d'alguna de les estacions de treball de la sala de projectistes. La màquina *MacWilliams* té les següents característiques:

- 2 processadors de doble nucli "Intel(R) Core(TM)2 CPU" de 1.86GHz de freqüència i 2028 Kb de memòria cache.
- 2 Gb de memòria RAM.

El MAGMA està organitzat en diferents directoris. Aquests els podem trobar a la ruta `/usr/local/magma2.13-12`. Els principals directoris són:

- `/libs`:
  - `/data`: en aquest directori s'hi troben totes les bases de dades de què disposa el MAGMA. Concretament aquí és on hi ha la base de dades de matrius Hadamard.
  - `/examples`: aquí s'hi troben tots els exemples que es proposen en el manual per tal de poder-los executar.
  - `/test`: en aquest directori s'hi troben els tests que ha passat el `magma` per comprovar el correcte funcionament de les seves funcions.
- `/package`: directori on es troben tots els *package* de què està format el MAGMA. Aquí trobarem entre d'altres tant el `hadamard.sig` com el `hadamard.m`.

- `/pdf`, `/ps` i `/dvi`: en aquest directoris podem trobar els manuals en format *pdf*, *ps* i *dvi*.

Per tal de poder treballar amb el MAGMA és necessari executar la comanda MAGMA. Un cop s'ha fet això, davant del prompt, ja podem començar a treballar de forma interactiva realitzant alguns càlculs manuals, o bé carregant programes per executar-los.

Si ens cal, podem interrompre l'execució de programes pitjant `<Crtl>-C`, la interrupció pot trigar una estoneta.

Per finalitzar la sessió en el MAGMA es pot fer de tres maneres igualment vàlides: Escrivint al prompt `quit;`, o bé fent `<Ctrl>-D`, o bé dos cops seguits `<Crtl>-C`.

4.2.1 Creació de funcions i procediments en MAGMA

Una forma més còmoda de treballar és escrivint el codi de les funcions i procediments en un fitxer a part amb extensió `.m`, així cada cop que comencem una nova sessió i necessitem les funcions implementades les podrem carregar amb la comanda `load "nom_fitxer.m";`.

Les funcions són un dels elements més importants del llenguatge MAGMA. La forma de crear-ne una és la següent:

```
f:= function(x1, ..., xn : y1:=expr1, ... )
    local w1, ..., ws;
    statements;
    return z1, ..., zr, _, ..., _;
end function;
```

Els paràmetres `x` són obligatoris, i els `y` són opcionals que prenen el valor per defecte de `expr`. Les `w` són les variables locals de la funció. Les `z` són els paràmetres de retorn, i els `_` són valors de retorn no definits que s'utilitzen per proporcionar un conjunt consistent de valors de retorn.

La crida a funcions és:

```
z1, ..., zm := f( x1, ..., xn : y1:=expr1, ... );
```

Qualsevol de les `y` es pot ometre en la crida, aquells que ho estiguin assumiran el valor per defecte dins la funció. Qualsevol de les `z` podem ser `_` , en aquest cas el resultat d'aquesta `z` no es retorna. Per exemple en la funció `Valuation(x,p)` podem fer varies crides segons ens interessi l'exponent `t` i/o el factor senar `s` tal que $n = 2^t \cdot s$:

```
t,s:= Valuation(n,2);
t,_:= Valuation(n,2);
_,s:= Valuation(n,2);
```

Els procediments són com les funcions però sense valors de retorn. A més els paràmetres poden ser referències `x̃`, les quals permeten retornar resultats mitjançant els arguments.

La definició del procediment és la següent:

```
f:= procedure( [~]x1, ..., [~]xn : y1:=expr1, ... )
    local w1, ..., ws;
    statements;
end procedure;
```

La crida a procediment és:

```
f( [~]x1, ..., [~]xn : y1:=expr1, ... );
```

Per exemple el procediment `UpdateHadamardDatabase( R,S : parameters)` modifica el registre R que conté la informació de la base de dades amb les noves matrius de `S`. Aquest register R està en un format no descrit i només és manipulable per referència en les poques funcions que actualitzant la base de dades.

Les funcions i els procediments també es poden declarar d'aquesta altre manera:

```
function f(x1, ..., xn : y1:=expr1, ... )
    ...
end function;

procedure f( [~]x1, ..., [~]xn : y1:=expr1, ... )
    ...
end function;
```

La única diferència respecte la primera forma rau en la recursivitat; doncs en aquesta segona forma de declarar, el nom de la funció o procediment forma part de la sintaxi, i això li permet cridar-se a si mateix pel seu propi nom.

Les funcions i procediments recursius es defineixen de tal manera que:

- Si la funció o procediment està declarat de la segona forma, les referències a la funció o procediment dintre del seu propi cos es poden fer amb el propi nom.
- Sinó, les referències a la funció o procediment dintre del seu propi cos es fan amb només `$$`.
- Si, per contra, es vol per claredat que la referència a la funció o procediment hi sigui, aleshores caldrà posar la comanda `forward f` abans de la definició de la funció o procediment.

4.2.2 Creació d'un *package* en MAGMA

Un paquet o *package* consisteix en varis fitxers `.m` que defineixen funcions i procediments, no intrínseques e intrínseques.

A partir d'ara, en aquesta subsecció, ens referim a les funcions i procediments intrínsics com tan sols intrínsics. I a les funcions i procediments no intrínsics com només no intrínsics.

Un paquet doncs disposa de intrínsics visibles per a l'usuari, i no intrínsics que són necessàris per executar-los però als quals l'usuari no pot accedir.

Ara bé, els intrínsics requereixen a més:

- La especificació de tipus dels paràmetres d'entrada i sortida (la signatura), per prevenir la sobrecàrrega de identificadors en els paquets.
- Uns breus comentaris explicatius del intrínsic.

Hi han dos tipus de intrínsics: els ja contruïts o de sistema que contenen les funcions del *kernel C*, i les definits per l'usuari.

La definició dels intrínsics requereixen doncs d'una sintaxi especial. Un cop s'ha decidit quines seran les funcions o procediments intrínsics, s'han de modificar de la següent manera:

```
intrinsic nom(llista-arguments) [-> llista retorn]
         {Comentaris de la funci\'{o}}
          sentencies;
end intrinsic;
```

on la llista d'arguments ha de ser:

```
arg1::tipus,arg2::tipus,...,argn::tipus
```

Així doncs, ja no és necessari realitzar el control del tipus dels paràmetres al codi. Si els paràmetres, però, han de complir alguna altra restricció, aquesta s'ha d'especificar utilitzant les següents comandes:

- `require condició: print_args;`
  On a `condició` s'especifica el tipus de restricció que s'ha d'aplicar i a `print_args` el text que s'imprimirà en cas de no complir-se la restricció, un exemple seria:
  `require IsHadamard(H): la matriu H no és Hadamard;`
- `requirerange v, L, U;`
  On `v` és la variable de tipus sencer i `[ L, U ]` és el rang al qual han de pertànyer els valors d'aquesta variable.

- `requirege v, L;`
  On `v` és també de tipus sencer i ha de ser més gran o igual que `L`.

Tenim les següents comandes per tal de treballar amb els fitxers un cop modificats i escrits en format *package*:

- `Attach ("fitxer.m")`
  Aquesta comanda carrega el fitxer (és la comanda equivalent al `load`), el precompila i en genera dos de nous d'extensió `.dat` i `.sig`:

  - `fitxer.dat` que conté el codi compilat.
  - `fitxer.sig` que conté la signatura dels intrínsics, la informació d'aquest fitxer es guarda en una taula de signatures.

  Per tal que, finalment, aquest *packet* es converteix-hi en una llibreria interna hem d'agafar el `fitxer.sig` i el `fitxer.m` i afegir-los al directori que abans hem explicat, amb la resta de signatures de què disposa el MAGMA. Els canvis en el fitxer `.m` es detecten automàticament i els seus corresponents fitxers `.sig` i `.dat` són actualitzats.

- `Detach ("fitxer.m")`
  Aquesta comanda serveix per deixar d'utilitzar un *package*.

- `freeze`
  Aquesta comanda afegida al principi d'un fitxer evita que aquest es compil·li quan es fa un `Attach`, per tant els canvis que faci al fitxer `.m` no afectaran als seus homólegs precompilats.

L'avantatge que ens proporciona el `package` és que podem treballar amb les funcions i procediments de la llibreria com si fossin pròpies del MAGMA.

4.3 Funcions del MAGMA

En aquesta secció es veuran les funcions per treballar amb matrius Hadamard que ja estan definides en MAGMA i que hem utilitzat en aquest projecte. [1]

`IsHadamard(H)`

Aquesta funció ha estat molt utilitzada per tal de comprovar si l'entrada era realment una matriu Hadamard, en aquest cas retornarà true.

`HadamardNormalize(H)`

Aquesta funció ens normalitza la matriu Hadamard que li passem. És a dir, ens retorna una matriu normalitzada equivalent a H. Això és molt útil ja que si treballem amb només matrius normalitzades són moltes les matrius equivalents que es descarten.

`HadamardCanonicalForm(H)`

Aquesta funció, donada una matriu de Hadamard H, retorna una d'equivalent H' juntament amb les matrius de transformació X i Y tal que $H' = XHY$. H' és canònica en el sentit que totes les matrius de Hadamard equivalents a H generaran la mateixa matriu H'.

`HadamardInvariant(H)`

Aquesta funció és la que ens retorna el 4-profile d'una matriu Hadamard, el 4-profile és una seqüència de sencers calculats a partir de les files de la matriu [2]. Donat que si dos matrius no tenen

[1] Per a més informació [2].
[2] Per més detalls 2.4.

el mateix 4-profile, no són equivalents; ha estat molt utilitzada per tal de trobar noves matrius no equivalents que no estiguessin a la base de dades.

`IsHadamardEquivalent(H, J : parameters)`

Aquesta funció ens retorna si dues matrius són equivalents, en aquest cas retornarà true. En cas de cridar-la amb el paràmetre "nauty", que és el valor per defecte; llavors, si les matrius són equivalents, retorna també les matrius de transformació X i Y tal que $J = XHY$.

`HadamardMatrixToInteger(H)`

Aquesta funció ens retorna un sencer que codifica els valors de la matriu Hadamard d'una forma més compacte, per tal d'estalviar temps en el tractament d'aquestes matrius.

`HadamardMatrixFromInteger(X,n)`

Aquesta funció, que la inversa de `HadamardMatrixToInteger()`, ens retorna la matriu Hadamard d'ordre n obtinguda del sencer X.

`HadamardDatabase()`

Aquesta funció ens permet recuperar totes les matrius Hadamard de la primera base de dades que té actualment el MAGMA, doncs retorna un apuntador en aquesta que s'usa com a primer paràmetre en les funcions d'obtenció dels nombres, ordres i matrius d'aquesta base de dades. Aquesta primera base de dades inclou totes les matrius inequivalents fins l'ordre 28, i alguns exemples

de tots els altres ordres fins el 256; els representants són canònics i ordenats lexicogràficament.
La segona base de dades de matrius Hadamard que té el MAGMA és la `skew-symmetric`, l'apuntador de la qual obtenim amb la funció `SkewHadamardDatabase()`, però que no són del nostre interés en aquest treball.

`Matrix(D, n, k)`

Aquesta funció retorna de la base de dades D, la matriu k-èssima d'ordre n. Permet recuperar qualsevol matriu Hadamard de la base de dades individualment.

`Matrices(D, n)`

Aquesta funció ens retorna una llista amb totes les matrius de mida n que estiguin a la base de dades D.

`DegreeRange(D)`

Aquesta funció retorna el interval dels ordres de les matrius Hadamard de la base de dades D. És a dir, el més petit i el gran més ordre existent a la base de dades.

`Degrees(D)`

Aquesta funció ens retorna una llista amb els ordres de les matrius Hadamard de la base de dades D, com a mínim ha d'existir una per cada ordre a la base de dades.

`NumberOfMatrices(D, n)`

Aquesta funció retorna el nombre de matrius d'ordre n disponibles a la base de dades D.

Donat que la base de dades de les matrius de Hadamard en el MAGMA és notablement incompleta com ja hem comentat anteriorment, i per tal de crear noves versions de la base de dades incloent matrius que actualment no hi són sense haver d'esperar les noves versions oficials de la base de dades; utilitzarem aquest conjunt de funcions per ampliar la base de dades de matrius Hadamard.
Aquestes funcions usen un registre de format no descrit, així només serà manipulable per aquestes.

`HadamardDatabaseInformation(D : parameters)`

Aquesta funció extrau la informació de la base de dades apuntada per D, a una format intern per tal de ser utilitzat pels altres intrínsics. El paràmetre `canonical` quan és `true`, que és el valor per defecte, indica que les matrius guardades a la base de dades estàn en forma canònica.
Aquest paràmetre també controla si la base de dades creada les guardarà en la forma canònica o en la forma original. Si vols recuperar les matrius des d'una base de dades canònica però emmagatzemar-la en una forma no canònica, s'hauria de crear primer la base de dades no canònica - ja sia usant aquest intrínsic o `HadamardDatabaseInformationEmpty` amb el paràmetre `canonical` posat a `false`, i després afegir les matrius de la base de dades amb `UpdateHadamardDatabase` i `canonical` posat a `true`.

`HadamardDatabaseInformationEmpty(: parameters)`

Aquesta funció retorna la informació en format intern corresponent a una base de dades buida. El paràmetre `canonical` serveix per indicar si les matrius en la base de dades seran guardades en la forma canónica o no.
Això permet la creació d'una nova base de dades , o un troç d'una d'existent sense incloure-la tota sencera.

`UpdateHadamardDatabase(R̃, S : parameters)`

Aquesta funció actualitza el registre R que conté la informació de la base de dades amb la de les matrius de S. Aquestes matrius seran afegides només si són inequivalents amb les que ja hi són. Això requereix trobar les formes canòniques, la qual cosa pot ser costós. Per això, si sabem que les matrius de S ja estan en forma canònica aleshores el paràmetre `canonical` hauria de ser posat a `true`.
Si de les matrius del mateix ordre que les que ja hi són en R, no sabem que siguin canòniques, llavors caldrà també calcular les seves formes canòniques, que un altre vegada pot arribar a durar un temps significatiu. Per tractar amb això hem de veure la funció `WriteRawHadamardData`.

`WriteHadamardDatabase(S, R̃)`

Aquesta funció crea els fitxers de la base de dades `name.dat` i `name.ind` del registre de dades d'informació de la base de dades R, on `name` es pren del *string* S. El registre de dades es passa per referència perquè pot ser necessari calcular les formes canòniques, així aquest càlcul no es perd.

`WriteRawHadamardData(S, R)`

Aquesta guarda les dades de R al fitxer el nom del qual ve donat pel *string* S. Un cop carregat, aquest fitxer definirà una única variable de dades que esdevindrà idèntica a R. Això és desitjable per les bases de dades no canòniques, ja que les formes canòniques no han de ser recalculades.
Aquesta rutina destrueix els continguts originals, si existien, del fitxer.

`SetVerbose("HadamardDB", v)`

Aquest procediment canvia el nivell de impressió *"verbose"* per les rutines d'actualització de la base de dades Hadamard. El valor *"verbose"* v hauria de ser un sencer en el interval de 0 a 3. Això pot suministrà indicacions segures de progrés quan un llarg procés d'actualització s'està efectuant.

4.4 Regles per executar grans càlculs.

Dintre d'un entorn operatiu *Linux*, que és en el que hem treballat, les regles són les següents:

1. Reduir la prioritat, realitzant els càlculs amb la prioritat de programació més baixa no hauria de tenir un impacte significatiu en el temps d'execució, sempre que no s'estigui executant un altre procés amb prioritat més alta a la mateixa estació de treball. La prioritat en la planificació de processos va de -20 a 19. La més baixa o lenta és 19. Haurem d'usar la comanda `nice -n 19 COMMAND`.

2. Incloure punts de control en el codi *(checkpoints)* : D'aquesta manera podrem fer un seguiment del procés, detectarem "bucles", i si es reinicia la màquina podrem reprendre el càlcul potser des del punt de tall i amb la mínima pèrdua de dades.

3. Apuntar-se, si existeix, a la llista de usuaris els processos dels quals no es poden matar *(DONTKILL LIST)* en cada una de les estacions de treball on penses realitzar els teus càlculs.

4. Minimitzar el tràfic de xarxa fent directament les operacions I/O al disc local, doncs l'accés local és 10 vegades més ràpid que el emmagatzemament en xarxa.

5. Executar la tasca no interactivament, ja que en alguns entorns interactius, al gestionar-se la memòria com una pila, pot fragmentar-se en el temps.

6. Planificar la tasca per que s'executi tard quan la càrrega del sistema és baixa.

7. Utilitzar la comanda "NOHUP" per executar la tasca, doncs el procés continuarà encara que es talli la connexió.

8. Executar la tasca en *background*, adreçant els canals d'errors i d'*output* a fitxers del disc dur local

9. Utilitzar una màquina ràpida.

Amb aquestes regles la tasca es podria llançar d'aquesta manera:

```
nohup nice -n 19 magma <infile >& outfile &
```

- En el `infile` tindrem la càrrega dels fitxers `.m` necessaris per la crida a la funció o procediment amb els valors escollits dels paràmetres.
- En el `outfile` es guarden els *checkpoints* i els possibles errors de la funció o procediment i de la tasca.

Capítol 5

Desenvolupament del projecte

Mostrem el codi implementat, els criteris que hem adoptat durant la implementació i les dificultats que hem trobat. Les funcions que no tenen el codi, tan sols la descripció en anglés tal com està en el manual, són les que han estat implementades per les dos projectistes anteriors [10, 11] i no s'han modificat.

5.1 Codi extern

5.1.1 Matrius Hadamard i conversió de codis

`HadamardMatrixToBinary(H)`

```
/*********************************************************************
HadamardMatrixToBinary: AlgMatElt -> AlgMatElt
Given a Hadamard matrix H, returns the corresponding Hadamard binary matrix.
This function is the inverse of HadamardBinaryToMatrix().
*********************************************************************/
```

`HadamardBinaryToMatrix(H)`

```
/*********************************************************************
HadamardBinaryToMatrix: AlgMatElt -> AlgMatElt
Given a binary Hadamard matrix H, returns the corresponding Hadamard matrix.
This function is the inverse of HadamardMatrixToBinary().
*********************************************************************/
```

`HadamardMatrixToCode(H)`

```
/*************************************************************************
HadamardMatrixToCode : AlgMatElt -> [ModTupFldElt]
Given a Hadamard matrix H, returns the corresponding
Hadamard code. The code is represented as a list of binary vectors
of length n. This function is the inverse of HadamardCodeToMatrix().
*************************************************************************/
```

`HadamardCodeToMatrix(C)`

```
/*************************************************************************
HadamardCodeToMatrix: [ModTupFldElt] -> AlgMatElt
Given a Hadamard code represented as a list of binary vectors of length n,
returns the corresponding normalized Hadamard matrix of degree n. This function
is the inverse of HadamardMatrixToCode().
*************************************************************************/
```

`IsHadamardCode(C)`

```
/*************************************************************************
IsHadamardCode: [ModTupFldElt] -> BoolElt
Returns true if and only if C is a Hadamard code.
*************************************************************************/
```

5.1.2 Invariants de codis Hadamard

Les primeres funcions, que són les que tenen com a únic paràmetre d'entrada el codi C, representat com una llista de vectors binaris; són generals en el sentit que podem servir per a qualsevol codi binari, no cal que sigui un codi binari Hadamard. Això és perquè implementen i utilitzen dos propietats estructurals de l'àlgebra lineal: el rang i la dimensió del nucli. Estem parlant de les funcions: `RankZ2(C)`, `KernelZ2(C)`, `DimensionOfKernelZ2(C)` i `InvariantsRankKernelZ2(C)`.

`RankZ2(C)`

```
/***********************************************************************
RankZ2: [ModTupFldElt] -> RngIntElt
Given a code C represented as a list of binary vectors of length n,
returns its rank. The rank of a code C is the dimension of the linear
span of C, <C>, over GF(2).
***********************************************************************/
```

`KernelZ2(C)`

Aquesta nova versió de `KernelZ2(C)` és més ràpida i requereix menys memòria perquè retorna el nucli com a subespai vectorial. Les proves de rendiment d'aquesta funció ho certifiquen. Efectivament, donat que per definició el nucli és un espai vectorial; podem aprofitar la representació interna i les funcions de que disposa el MAGMA per treballar amb espais vectorials en el mòdul d'àlgebra lineal.

Comentem de passada que aquesta funció sempre ha suposat que el vector tot zeros forma part del codi C, dit d'una altre manera, estem suposant que el nucli està inclòs en el codi. Això implica dues coses importants:

1. La construcció del nucli es redueix en aquest cas a buscar possibles vectors en el propi codi tal que $c + C = C$.

2. Si dóna la casualitat que la llista de vectors binaris que formen el codi C també té estructura d'espai vectorial, llavors el codi C, el nucli $K(C)$ i el generat del codi $\langle C \rangle$ coincideixen.

Per això mirem, primer de tot, si la llista de vectors binaris que formen el codi C és un subespai vectorial. Si és així, el retornen transformat com subespai vectorial i ja hem acabat. La manera de mirar-ho és comprovant que el codi com a conjunt tindria el mateix nombre d'elements que el seu propi subespai vectorial generat.

Construïm el primer nucli a partir del vector tot zeros que hauria per hipòtesi d'estar inclòs en el codi. Acte seguit, si el vector tot uns pertany al codi, llavors mirem si també pertany al nucli comprovant que $1 + C = C$.

Si el vector tot uns és del nucli quedaran inclosos automàticament tots els complementaris.

Engrandim el nucli, afegint més vectors c de C si compleixen que $c+C = C$. Per això sumem cada vector del codi, que no estigui ja en el nucli per combinacions lineals anteriors, amb cadascun dels vectors del propi codi. A cada nou vector del nucli trobat, reconstruïm el subespai vectorial nucli combinant-lo linealment amb el subespai vectorial del nou vector del nucli.

Totes les funcions que criden a la funció `KernelZ2(C)` les haurem de revisar perquè ara rebran un espai vectorial enlloc d'una subllista de vectors binaris del codi, que és el que retornava la versió anterior.

```
/***************************************************************************

KernelZ2: [ModTupFldElt] -> ModTupFld
Given a code C represented as a list of binary vectors of length n and such
that the zero vector belongs to C, returns its kernel as a VectorSubspace.
Then the kernel of C are the codewords v such that v+C=C.

***************************************************************************/

KernelZ2:=function (C)

    if ((Type(C) eq SeqEnum) and (ElementType(C) eq ModTupFldElt)) then
        if (VectorSpace(GF(2),Degree(C[1]))!0 in C) then
            n:=Degree(C[1]); V:=VectorSpace(GF(2),n);

            // If C is a subspace then return it as the kernel
            if (#Set(C) eq 2^(Dimension(sub<V|C>))) then
                return sub<V|C>;
            end if;

            // first kernel is the vectorsubspace with only Zero and maybe the One vector
            ZeroVector:=V!0; OneVector:=V![1:i in [1..n]]; numElt:=#(C); isKernel:= true;
            if (OneVector in C) then
                for j in [1..numElt] do
                    aux:=OneVector+C[j];
                    isKernel:=aux in C;
                    if (not isKernel) then
                        break;
                    end if;
                end for;
                if (isKernel) then
                    kernel:=sub<V|[ZeroVector,OneVector]>;
```

```
                else
                    kernel:=sub<V|ZeroVector>;
                end if;
            else
                kernel:=sub<V|ZeroVector>;
            end if;

            // add the others possible vectors of C for the kernel
            for i in [1..numElt] do
                if C[i] notin kernel then
                    isKernel:= true;
                    for j in [1..numElt] do
                        if C[j] notin kernel then
                            aux:=C[i]+C[j];
                            isKernel:=aux in C;
                            if (not isKernel) then
                                break;
                            end if;
                        end if;
                    end for;
                    if (isKernel) then
                        kernel:=sub<V|[kernel,sub<V|C[i]>]>; // posar generadors
                    end if;
                end if;
            end for;
        else
            error "Runtime error in 'KernelZ2': ZeroVector not in code";
        end if;
    else
        error "Runtime error in 'KernelZ2': Argument 1 is not a code";
    end if;
    return kernel;
end function;
```

`DimensionOfKernelZ2(C)`

S'ha modificat aquesta funció `DimensionOfKernelZ2(C)` com a conseqüència d'haver modificat la funció `KernelZ2(C)`. Ja que ara simplement aquesta funció avalua directament la dimensió de l'espai vectorial retornat per la funció `KernelZ2(C)`, mentre que abans havia de calcular prèviament el subespai vectorial generat de la llista nucli per trobar la dimensió.

```
/***************************************************************************

DimensionOfKernelZ2: [ModTupFldElt] -> RngIntElt
Given a code C represented as a list of binary vectors of length n,
returns the dimension of its kernel. The code C must contain the zero
vector to assure that its kernel is a linear subspace of C over GF(2).

***************************************************************************/

DimensionOfKernelZ2:=function (C)

//Returns the dimension of kernel of this code
if((Type(C)eq SeqEnum) and (ElementType(C) eq ModTupFldElt)) then
                return Dimension(KernelZ2(C));
else
                error "Runtime error in 'DimensionOfKernelZ2':
                Argument 1 is not a code";
end if;

end function;
```

InvariantsRankKernelZ2(C)

```
/***************************************************************************
InvariantsRankKernelZ2: [ModTupFldElt] -> [RngIntElt]
Given a code C represented as a list of binary vectors of length n,
returns its rank and dimension of the kernel.
***************************************************************************/
```

HadamardThreeInvariants(H)

```
/***************************************************************************
HadamardThreeInvariants: AlgMatElt -> [RngIntElt]
Given a Hadamard matrix H, returns the invariants 4-profile, rank and
dimension of the kernel.
***************************************************************************/
```

`ExistsHadamardRankKernel(n,r,k)`

```
/***********************************************************************
ExistsHadamardRankKernel: RngIntElt,RngIntElt,RngIntElt -> BoolElt
Given positive integers n, r and k, returns true if there exists a
Hadamard code (or equivalently a Hadamard matrix) of length n with
rank r and kernel of dimension k.
When n is not a power of two, returns also false if we do not know
whether or not there exists a Hadamard code with these parameters.
***********************************************************************/
```

Invariant SHDE

Aquest invariant proposat per Kai-Tai Fang i Gennian Ge, es basa en una propietat anomenada Enumerador de la Distància Simètrica de Hamming ("Symmetric Hamming Distance Enumerator" en anglès i "SHDE" com acrònim). De fet la invariant és la distribució de freqüències dels valors d'aquesta propietat per totes les projeccions de k columnes de n d'una matriu de Hadamard H d'ordre n i per un número a positiu. Les funcions públiques són:

- `HadamardInvariantSHDE(H:a:=3.1415926,k:=4)`
- `HadamardInequivalentMatricesSHDE(H1,H2:a:=3.1415926)`
- `HadamardClassificationSHDE(n:a:=3.1415926,k:=4,lr:="",file:="")`

Totes elles criden a la funció auxiliar principal `SHDEDistribution(a,k,H)`, que a la vegada crida a las seves funcions auxiliars:

- `SHDkDistribution(kcol,H)`
- `SHDkEnumerator(a,kcol,H)`
- `nextComb(n,k,c)`

`SHDkDistribution(kcol,H)`

Sigui *kcol* una llista concreta de columnes i notem k com el nombre de columnes de la llista ($k = \#kcol$), aleshores aquesta funció calcula la distribució de freqüències de la Distància Simètrica de Hamming d'una matriu de Hadamard H d'ordre n per les k columnes de n guardada en la llista *kcol*. Retorna doncs un vector de freqüències de $k/2 + 1$ components, perquè la Distància Simètrica de Hamming té un interval de valors que va de 0 fins a $k/2$, i cada component és el nombre de parelles de files distintes de la matriu de Hadamard amb el mateix valor de Distància Simètrica de Hamming sobre aquesta llista *kcol* de k columnes.

Si cridem a la funció amb la llista única de totes les n columnes, $kcol := [1..n]$ i $k = n$, aleshores calcula la distribució de freqüències de la Distància Simètrica de Hamming de la matriu Hadamard. El vector de freqüències retornat, anomenat *distribució de distància de* H, té $n/2 + 1$ components perquè la Distància Simètrica de Hamming té un interval de 0 a $n/2$ (Per més detalls veure la secció 2.5 del capítol 2).

En l'article dels xinesos Kai-Tai Fang i Gennian Ge [5], per simplificar el càlcul, es relaciona la Distància Simètrica de Hamming amb el valor absolut del producte escalar de les files en les components que pertanyen a la llista *kcol*.

```
/*****************************************************************************

SHDkDistribution: [RngIntElt],AlgMatElt -> [RngIntElt]
Given a list of columns kcol (k=\#kcol) and given a Hadamard matrix H, returns
the Symmetric Hamming Distance distribution of H over the k-dimensional column
projection, (S0(H),...,Sk/2(H)), where Si(H) is the number of pairs of two distinct
rows whose Symmetric Hamming Distance over the given k-dimensional column projection
is i.

*****************************************************************************/

SHDkDistribution:=function(kcol,H)
    k:=#kcol; n:=NumberOfColumns(H);
    S:=[0:d in [0..(k div 2)]];
    for i:=1 to n-1 do
      for j:=i+1 to n do
```

```
        // Absolute value of the inner product |<Uk,Vk>|
        UkVk:=0;for l in kcol do UkVk:=UkVk+(H[i][l]*H[j][l]);end for;UkVk:=Abs(UkVk);
        // symmetric distance d in terms of the inner product
        d:=(k-UkVk) div 2;
        S[d+1]:=S[d+1]+1;
      end for;
    end for;
    return S;
end function;
```

`SHDkEnumerator(a,kcol,H)`

Sigui $kcol$ una llista concreta de columnes i notem k com el nombre de columnes de la llista ($k = \#kcol$), aleshores aquesta funció calcula l'Enumerador de la Distància Simètrica de Hamming d'una matriu de Hadamard H d'ordre n per les k columnes de n guardada en la llista $kcol$.

Aquest Enumerador de la Distància Simètrica de Hamming és un número, que s'obté ponderant les components del vector distribució de freqüències de la Distància Simètrica de Hamming per la mateixa projecció de k columnes, amb les potències d'un nombre positiu a, a^i i a^{k-i}, on i pren els diferents valors de la Distància Simètrica de Hamming. En aquest article [5], un valor aproximat del nombre pi, 3.1415926, es assignat al paràmetre a.

Si cridem a la funció amb la llista única de totes les n columnes, $kcol := [1..n]$ i $k = n$, aleshores calcula l'Enumerador de la Distància Simètrica de Hamming de la matriu Hadamard, i el número obtingut s'anomena l'enumerador de distància de H.

```
/**************************************************************************

SHDkEnumerator: RngIntElt,[RngIntElt],AlgMatElt -> FldReElt
Given a ponderator positive number a, a list of columns kcol (k=\#kcol)
and a Hadamard matrix H, returns the Symmetric Hamming Distance Enumerator
of H over the k-dimensional column projection:
Bak(H)=Sum(i=0..k/2) Si(H)(a^(i)+a^(k-i)), where Si(H) is the Symmetric
Hamming Distance distribution of H over the k-dimensional column
projection.

**************************************************************************/
```

```
SHDkEnumerator:=function(a,kcol,H)

    // k-column distance enumerator for a given k-column projection
    k:=#kcol; S:=SHDkDistribution(kcol,H);
    Bak:=0;
        for d:=0 to (k div 2) do
            Bak:=Bak+(S[d+1]*((a^(d))+(a^(k-d))));
        end for;
    return Bak;
end function;
```

nextComb(n,k,c)

Si considerem les combinacions de n elements agafats de k en k, aquesta funció retorna la combinació següent a la combinació entrada c seguint l'ordre creixent. Entrant-li la llista buida retorna la primera combinació, i entrant-li la última retorna la llista buida.

Mostrarem un exemple:

```
> nextComb(4,2,[]);
[ 1, 2 ]
> nextComb(4,2,[ 1, 2 ]);
[ 1, 3 ]
> nextComb(4,2,[ 3, 4 ]);
[]

/*****************************************************************************

nextComb: RngIntElt,RngIntElt,[RngIntElt] -> [RngIntElt]
Given positive integers n, k (k<=n) and a list of an especific k-combination of n elements
c, returns the next combination. If c is an empty list, returns the first combination,
and if c is the last combination, returns the empty list.

*****************************************************************************/

nextComb:=function(n,k,c)
    if IsEmpty(c) then
        return [i:i in [1..k]];
    elif (k eq 1) then
        if (c[k] ge 1) and (c[k] le n-1) then
            return [c[k]+1];
        else
            return [];
        end if;
```

```
    else
        next:=false;
        while (not next) do
            c[k]:=c[k]+1;
            if (c[k] le n) then
                next:=true;
            else
                npc:=$$(n,k-1,Prune(c));
                if (IsEmpty(npc)) then
                    c:=[];
                    next:=true;
                else
                    c:=npc cat [npc[#npc]];
                end if;
            end if;
        end while;
        return c;
    end if;
end function;
```

`SHDEDistribution(a,k,H)`

Aquesta funció calcula la distribució de la part entera de l'Enumerador de la Distància Simètrica de Hamming, ponderada amb el nombre positiu a, d'una matriu de Hadamard H d'ordre n per totes les projeccions de k columnes. Segons l'article [5], aquesta distribució és manté invariant a les permutacions i negacions de files o columnes d'una matriu de Hadamard.

Al realitzar el càlcul per totes les col·leccions de k columnes de n, obtenim molts valors de la part entera de l'Enumerador, alguns valors estan repetits i per tant caldrà comptabilitzar-los en forma de distribució de freqüències.

La funció retorna doncs una llista de parelles. Els primers elements de les parelles són els diferents valors de la part entera de l'Enumerador de la Distància Simètrica de Hamming de la matriu de Hadamard per totes les projeccions de k columnes, i els segons elements de les parelles són les respectives freqüències d'aquests valors.

En el programa implementat, per cada una de les combinacions de k columnes de n, calculem la part entera de l'Enumerador de la Distància Simètrica de Hamming i incrementem en un el comptador del seu valor. I

si és el primer aparegut d'aquest valor, creem la parella [valor, comptador] amb el comptador a un.

És necessari la utilització de la funció auxiliar `nextComb(n,k,c)` que genera la següent combinació a partir de l'anterior, d'aquesta manera, iterant-la, ens dóna una a una totes les combinacions sense esgotar la memòria. Això és el que passaria si les volguéssim guardar en local [1] amb la comanda MAGMA `Subsets(Set([1..n]),k)` per un n relativament gran respecte k. Pensem que, per exemple, el número de col·leccions diferents de 8 columnes d'una matriu de Hadamard d'ordre 32 és 10.518.300. Ara bé, és més ràpid llegir la següent combinació d'una llista de combinacions ja construïda, sempre que es pugui guardar en local, que cridar cada cop a una funció auxiliar per generar-la.

Adoptem doncs el criteri d'estalviar memòria, millor dit de salvar-nos d'un exhauriment de memòria, a canvi d'augmentar el temps de càlcul.

```
/***************************************************************************

SHDEDistribution: AlgMatElt,RngIntElt,RngIntElt -> [[RngIntElt,RngIntElt]]
Given a ponderator positive number a, a positive integer k and a Hadamard matrix H
returns the values of the integer part of the Symmetric Hamming Distance Enumerator of
H over all k-dimensional columns projections and their frequencies.

***************************************************************************/

SHDEDistribution:=function(a,k,H)
    n:=NumberOfColumns(H);
    FiBak:=[];
    kcol:=nextComb(n,k,[]);
    while (kcol ne []) do
        // integer part of the k-column distance enumerator for a given k-column projection
        iBak:=Floor(SHDkEnumerator(a,kcol,H));
        listed:=false;
        for i in [1..#FiBak] do
            if (FiBak[i][1] eq iBak) then
                FiBak[i][2]:=FiBak[i][2]+1;
                listed:=true;
                break;
            end if;
        end for;
```

[1] Per local volem dir en la partició de memòria assignada al procés.

```
        if (not listed) then
            Append(~FiBak,[iBak,1]);
        end if;
        kcol:=nextComb(n,k,kcol);
    end while;
    Sort(~FiBak);
    return FiBak;
end function;
```

`HadamardInvariantSHDE(H : a:=3.1415926 ,k:=4)`

Aquesta funció retorna només les freqüències de la distribució de la part entera de l'Enumerador de la Distància Simètrica de Hamming, ponderada amb el nombre positiu a, d'una matriu de Hadamard H d'ordre n per totes les projeccions de k columnes.

Després de comprovar la correció dels paràmetres d'entrada, crida a la funció `SHDEDistribution:=function(a,k,H)` i s'agafa la segona part de les llistes de parelles [valor,comptador]. És a dir, ens retorna les freqüències de la distribució de l'Enumerador de la Distància Simètrica de Hamming d'una matriu de Hadamard H d'ordre n per totes les projeccions de k columnes.

Retorna el mateix tipus de sortida que la funció `HadamardInvariant(H)` que calcula la invariant 4-profile, és a dir, una llista de freqüències ordenada de forma creixent pel valor freqüènciat.

Comprovarem més endavant que la funció `HadamardInvariantSHDE(H:a,k)`, per $a = 3.1415926$ i $k = 4$, dóna les mateixes freqüències que la funció `HadamardInvariant(H)` del 4-profile.

Assignem el números 3.1415926 i 4 com valors per defecte dels paràmetres d'entrada a i k respectivament.

```
/************************************************************************
HadamardInvariantSHDE: AlgMatElt: FldReElt, RngIntElt  -> [RngIntElt]
Given a Hadamard matrix H, a ponderator positive number a and a positive
integer k, returns only the frequencies of the Symmetric Hamming Distance
Enumerator distribution of H over all k-dimensional columns projections.
Invariant proposed by Kai-Tai Fang and Gennian Ge. It defaults a to 3.1415926
and k to 4.
************************************************************************/
```

```
HadamardInvariantSHDE:=function(H : a:=3.1415926 ,k:=4)
    if ((Type(a) eq FldReElt) and (a gt 0)) then
        if ((Type(k) eq RngIntElt) and (k gt 0)) then
    if((Type(H) eq AlgMatElt) and IsHadamard(H)) then
                if (k le NumberOfColumns(H)) then
                    FiBak:=SHDEDistribution(a,k,H);
                    FG:=[]; for i:=1 to #FiBak do Append(~FG,FiBak[i][2]); end for;
            return FG;
                else
                    error "Runtime error in 'HadamardInvariantSHDE':
    Argument 3 is greater than the number of columns of Hadamard matrix";
                end if;
            else
                error "Runtime error in 'HadamardInvariantSHDE':
Argument 1 is not a Hadamard matrix";
            end if;
        else
            error "Runtime error in 'HadamardInvariantSHDE':
            Argument 3 is not greater than 0";
        end if;
    else
        error "Runtime error in 'HadamardInvariantSHDE':
        Argument 2 is not greater than 0";
    end if;
end function;
```

`HadamardInequivalentMatricesSHDE(H1,H2 :  a:=3.1415926)`

Aquesta funció implementa l'algorisme que els xinesos Kai-Tai Fang i Gennian Ge relaten en el seu article [5], basant-se en la seva conclusió de que dos matrius de Hadamard $H1$ i $H2$ d'ordre n són inequivalents quan existeix una k per la qual no tenen la mateixa distribució de la part entera de l'Enumerador de la Distància Simètrica de Hamming, ponderada amb el mateix nombre positiu a.

Començant per $k = 4$, la funció repeteix el càlcul de la distribució de la part entera de l'Enumerador de la Distància Simètrica de Hamming per les dos matrius, incrementant la k en 2 a cada repetició, fins que deixen de ser iguals o bé hem arribat al valor màxim de la k possible, $k/2$. En cas de

que deixin de ser iguals per alguna k sense d'haver sobrepassat el seu llindar, sortim del bucle marcant la inequivalència, en cas contrari sortim sense saber res de la inequivalència d'aquestes dues matrius per aquest algorisme.

La k s'incrementa en 2 a cada repetició, perquè el valor de la magnitud en dos iteracions consecutives quasi té el mateix efecte a l'hora de distingir la inequivalència de les matrius de Hadamard.

Per defecte assignem al paràmetre d'entrada a el valor 3.1415926.

```
/*****************************************************************************

HadamardInequivalentMatricesSHDE: AlgMatElt, AlgMatElt: RngIntElt -> BoolElt
Given two Hadamard matrices H1, H2 and a ponderator positive integer a, returns true
if they are inequivalent and false if is not possible to prove their inequivalence by
this method. It defaults a to 3.1415926.

*****************************************************************************/

HadamardInequivalentMatricesSHDE:=function(H1,H2 : a:=3.1415926)
    if ((Type(H1) eq AlgMatElt) and IsHadamard(H1)) then
        if ((Type(H2) eq AlgMatElt) and IsHadamard(H2)) then
            n:=NumberOfColumns(H1);
            if (n ne NumberOfColumns(H2)) then
                if ((Type(a) eq FldReElt) and (a gt 0)) then
                    Inequivalence:=false; k:=2;
                    repeat
                        k:=k+2;
                        FBak1:=SHDEDistribution(a,k,H1);
                        FBak2:=SHDEDistribution(a,k,H2);
                        if (FBak1 ne FBak2) then
                            Inequivalence:=true;
                            break;
                        end if;
                    until (k eq (n div 2));
                    return Inequivalence;
                else
                    error "Runtime error in 'HadamardInequivalentMatricesSHDE':
                    Argument 3 is not greater than 0";
                end if;
            else
                error "Runtime error in 'HadamardInequivalentMatricesSHDE':
                different order of the Hadamard matrices";
            end if;
        else
            error "Runtime error in 'HadamardInequivalentMatricesSHDE':
            Argument 2 is not a Hadamard matrix";
```

```
        end if;
    else
        error "Runtime error in 'HadamardInequivalentMatricesSHDE':
        Argument 1 is not a Hadamard matrix";
    end if;
end function;
```

`HadamardClassificationSHDE(n:a:=3.1415926,k:=4,lr:="",file:="")`

Aquesta funció classifica les matrius Hadamard d'ordre n de la base de dades Hadamard ubicada en l'arrel de la llibreria lr segons la distribució de la part entera de l'Enumerador de la Distància Simètrica de Hamming, ponderada amb el mateix nombre positiu a, per totes les projeccions de k columnes. Retorna la llista de les distribucions (valors de l'enumerador i freqüències) amb les matrius que donen la mateixa distribució. La llista també és guarda en un fitxer amb nom per defecte `HadClaSHDE<n>`. Per defecte: a pren el valor 3.1415926, k pren el valor 4 i lr l'arrel de la llibreria de la base de dades Hadamard estàndard del MAGMA.

La classificació regeix quan el nombre d'elements de la llista és igual al nombre de matrius d'ordre n que estem classificant, llavors cada distribució té una sola matriu amb aquest valor.

```
/*****************************************************************************************

HadamardClassificationSHDE:RngIntElt:RngIntElt,RngIntElt,MonStgElt,MonStgElt->[[[[RngIntElt]]]]
Given an order n, a ponderator positive number a, a positive integer k, a library root lr and
a filename file, returns and saves in the file the list of the values of the integer part of
the Symmetric Hamming Distance Enumerator over all k-dimensional columns projections and their
frequencies of all the Hadamard matrices of order n from the hadamard database of the library
root. The classification is perfect when the number of elements of the list equals number of
Hadamard matrices. It defaults a to 3.1415926, k to 4, lr to the standard Magma HadamardDatabase
and file to HadClaSHDE<n>.

*****************************************************************************************/

HadamardClassificationSHDE:=function(n:a:=3.1415926,k:=4,lr:="",file:="")
    if (lr eq "") then
        D:=HadamardDatabase();
    else
        stdlibroot:=GetLibraryRoot();
```

```
        SetLibraryRoot(lr);
        D:=HadamardDatabase();
        SetLibraryRoot(stdlibroot);
    end if;
    if (file eq "") then file:="HadClaSHDE" cat IntegerToString(n); end if;

    // United list of |Bak| values for all the Hadamard matrices
     NFiBak:=[];
    for num:=1 to NumberOfMatrices(D,n) do
        H:=Matrix(D,n,num);
        FiBak:=SHDEDistribution(a,k,H);
        listed:=false;
        for i in [1..#NFiBak] do
            if (NFiBak[i][1] eq FiBak) then
                Append(~NFiBak[i][2],[n,num]);
                listed:=true;
                break;
            end if;
        end for;
        if (not listed) then
            Append(~NFiBak,[FiBak,[[n,num]]]);
        end if;
    end for;
    fprintf file,"n=%o  a=%o  k=%o  lr=%o\n",n,a,k,lr;
    for i in [1..#NFiBak] do
        fprintf file,"%o\n",NFiBak[i];
    end for;
    return NFiBak;
end function;
```

5.1.3 Construccions de les matrius de Hadamard

`HadamardKroneckerSylvester(t)`

```
/***************************************************************************
HadamardKroneckerSylvester: RngIntElt -> AlgMatElt
Given a positive integer t, returns the Hadamard-Sylvester matrix of
degree n=2^t.
***************************************************************************/
```

`HadamardKronecker(H1,H2)`

```
/************************************************************************
HadamardKronecker: AlgMatElt,AlgMatElt -> AlgMatElt
Given two Hadamard matrices H1 and H2 of degree n, returns a Hadamard
matrix of degree 2n. This matrix is constructed with the Kronecker
product S x [H1,H2], where S is the Hadamard-Sylvester matrix of degree 2.
************************************************************************/
```

`HadamardKroneckerPermutation(H1,H2,g)`

```
/************************************************************************
HadamardKroneckerPermutation: AlgMatElt,AlgMatElt,GrpPermElt -> AlgMatElt
Given two Hadamard matrices H1 and H2 of degree n and a permutation g in
Sym(n), returns a Hadamard matrix of degree 2n. This matrix is
constructed with the Kronecker product S x [H1,g(H2)], where S is the
Hadamard-Sylvester matrix of degree 2 and g(H2) is H2 with the columns
permuted by g.
************************************************************************/
```

`SwitchCode(C,S,x)`

```
/************************************************************************
SwitchCode:[ModTupFldElt],[ModTupFldElt],ModTupFldElt -> [ModTupFldElt]
Given a code C represented as a list of binary vectors, a subset S of C
and a codeword x, returns a code where the codewords of S are
substituted by the binary vectors of S+x.
************************************************************************/
```

`FindPermutation(H2,n)`

```
/************************************************************************
FindPermutation: AlgMatElt, RngIntElt -> GrpPermElt
Given a matrix H2 and a number of columns returns the positions of the
columns that have to be permuted.
************************************************************************/
```

`FindVectorNotInKernel(C,V,w)`

```
/***************************************************************************
FindVectorNotInKernel: [ModTupFldElt],ModTupFld,ModTupFld -> ModTupFldElt
Given a code C, a vector space V and a kernel returns a vector that is
not in the kernel.
***************************************************************************/
```

`ListConstruction(n,d)`

```
/***************************************************************************
ListConstruction: RngIntElt,RngIntElt -> [RngIntElt]
Given n and d returns a binary list of length n with d divisions.
***************************************************************************/
```

`HadamardRankKernelNPower2(n,r,k)`

```
/***************************************************************************
HadamardRankKernelNPower2: RngIntElt,RngIntElt,RngIntElt -> AlgMatElt
Given integers n(=2^t), r and k returns a Hadamard matrix that have length n,
rank r and kernel k.
***************************************************************************/
```

`HadamardRankKernelNNotPower2(n,r,k)`

Aquesta funció auxiliar permet obtenir una matriu de Hadamard d'ordre n per a cada parella de rang r i dimensió de nucli k, però tan sols per aquelles matrius que l'ordre no és una potència estricta de dos, és a dir: $n = 2^t \cdot s$ ($t \geq 2$), on $s \neq 1$ senar.

La funció utilitza unes construccions fonamentades en el treball sobre el rang i el nucli dels codis binaris Hadamard de longitud $n = 2^t \cdot s$ (s senar) de [9]. I l'algorisme que s'ha implementat, basat en aquestes construccions, està descrit en la subsecció 2.6.4 del capítol 2. Destaquem tres aspectes:

- La funció és bàsicament recursiva, construeix en la majoria dels casos la matriu actual a partir de matrius d'ordre la meitat i així successivament fins arribar a buscar la matriu d'ordre $4s$. Si aquesta no la tenim a la base de dades Hadamard del MAGMA, no es podrà construir la matriu Hadamard d'ordre n i la funció retorna un missatge d'error.

- Es diferencien el casos $n = 4s$ i $n = 8s$ perquè és regeixen per teoremes especials de rang i dimensió del nucli en les matrius Hadamard d'ordre no potència de dos. A més el cas $n = 8s$ és el punt d'inflexió del camí recursiu, si bé es construeix a partir del $n = 4s$ buscat a la base de dades Hadamard del MAGMA com ja s'ha comentat. Així doncs el cas $n = 4s$ està especificat per si es demana de forma directa a la funció.

- Per la resta de casos s'apliquen 5 construccions diferents de matrius Hadamard, la majoria recursives. Si $H^n_{r,k}$ és una matriu Hadamard d'ordre n, rang r i dimensió del nucli k tenim que:

  1. [$*$] Aplica el producte de Kronecker amb matrius Hadamard:

  $$S \otimes H^{n/2}_{r-1,k-1}.$$

  2. [$\circ$] Aplica el producte de Kronecker amb matrius Hadamard:

  $$S \otimes [H^{n/2}_{r-1,1}, H^{n/2}_{r-1,2}].$$

  3. [$\bullet$] Aplica la tècnica del switching.
  4. [$\star$] Aplica el producte de Kronecker amb dues matrius Hadamard més una permutació π:

  $$S \otimes [H^{n/2}_{r-2,k}, \pi(H^{n/2}_{r-2,k})].$$

  5. [$\diamond$] Transposada d'una matriu Hadamard $H^n_{r,1}$.

Aquesta funció generalitza i per tant inclou totes les funcions especials per casos que s'havien fet en el projecte anterior en relació a la construcció de matrius Hadamard del mateix tipus (veure [11]). Hem tret doncs del fitxer `hadamardfile.m` les funcions següents:

- $s = 3$ i $t = 2, 3, 4, 5$:
  - `HadamardRankKernelN12(r,k)`

  - HadamardRankKernelN24(r,k)
  - HadamardRankKernelN48(r,k)
  - HadamardRankKernelN96(r,k)
  - HadamardRankKernelN192(r,k)

- $s = 5$ i $t = 2, 3, 4, 5, 6$:
  - HadamardRankKernelN20(r,k)
  - HadamardRankKernelN40(r,k)
  - HadamardRankKernelN80(r,k)
  - HadamardRankKernelN160(r,k)
  - HadamardRankKernelN320(r,k)

- $s = 7$ i $t = 2, 3, 4, 5$:
  - HadamardRankKernelN28(r,k)
  - HadamardRankKernelN56(r,k)
  - HadamardRankKernelN112(r,k)

```
/*************************************************************************
HadamardRankKernelNNotPower2: RngIntElt,RngIntElt,RngIntElt -> AlgMatElt
Given integers n(=2^t.s), r and k returns a Hadamard matrix
that have length n, rank r and kernel k.
*************************************************************************/

function HadamardRankKernelNNotPower2(n,r,k)

    t,s:=Valuation(n,2);

    ////////////////////////////////////////////////////////////////
    // Distinguishes the first two cases of the sequence (4s i 8s)  //
    // in whatever other case recursion calls are used              //
    // case 4s     : r=4s-1, k=1 (only one)                         //
    // case 8s     : r=4s, k=1,2 (two)                              //
    // other case  : white0, aster6, rombo, black0, aster5          //
    ////////////////////////////////////////////////////////////////

    case n:
        when (4*s):
            return Matrix(HadamardDatabase(),n,1);
        when (8*s):
            m4s:=Matrix(HadamardDatabase(),(4*s),1);
```

```
    if (k eq 1) then
        return HadamardKroneckerPermutation(m4s,m4s,FindPermutation(m4s,2));
    else
        return HadamardKronecker(m4s,m4s);
    end if;
else:
    // white0 single case: H(n,r(=rmin),1) = S x [H(n/2,r-1,1),H(n/2,r-1,2)]
    rmin:=4*s+t-3;
    if ((r eq rmin) and (k eq 1)) then
        return HadamardKronecker($$(n div 2,rmin-1,1),$$(n div 2,rmin-1,2));

    // aster6 case: H(n,r,k) = S x H(n/2,r-1,k-1)
    elif (ExistsHadamardRankKernel(n div 2,r-1,k-1)) then
        return KroneckerProduct(HadamardKroneckerSylvester(1),$$(n div 2,r-1,k-1));

    // rombo case: H(n,r,2) = Transpose(H(n,r,1))
    elif ((r ge ((n div 4)+2)) and (k eq 2)) then
        return Transpose(HadamardNormalize($$(n,r,1)));

    // black0 single case: H(n,r(=n/4+2),3) = switching code of H(n,r-1,3)
    elif ((r eq ((n div 4)+2)) and (k eq 3)) then
        C:=HadamardMatrixToCode($$(n,((n div 4)+1),3));
        V:=VectorSpace(GF(2),n);kernel:=KernelZ2(C);
        Basis:=ExtendBasis([V![1:i in [1..n]]],kernel);
        v1v2:=V![Basis[2][i]*Basis[3][i]:i in [1..n]];
        x:=FindVectorNotInKernel(C,V,kernel);
        S:=[v+x:v in kernel];
        return HadamardCodeToMatrix(SwitchCode(C,S,v1v2));

    // aster5 case: permutations H(n,r,1)=Sx[H(n/2,r-2,1),perm(H(n/2,r-2,1))]
    elif (k eq 1) then
        rper1:=rmin+1;rper2:=(n div 4)+2;
        if ((r ge rper1) and (r lt rper2)) then
            H:=$$((n div 2),(r-2),1);
            columns:=2;
            perm:=FindPermutation(H,columns);
            return HadamardKroneckerPermutation(H,H,perm);

        elif (r ge rper2) then
            H:=$$((n div 2),(rper2-2),1);
            columns:=r-(rper2-2);
            perm:=FindPermutation(H,columns);
            return HadamardKroneckerPermutation(H,H,perm);
        end if;
    else
        error "Runtime error in 'HadamardRankKernelNNotPower2':
               lost in recursion by other cases";
```

```
            end if;

    end case;

end function;
```

`HadamardAllRankKernel(n:  swint:=false)`

Aquesta funció retorna una llista de matrius Hadamard d'ordre n, n'hi ha una per cada possible valor de rang r i dimensió del nucli k compatibles amb l'ordre. Com a opció les pot retornar convertides a enters, mitjançant la funció `HadamardMatrixToInteger(H)` si activem el paràmetre opcional *swint*.

Aquesta funció en relació a la versió anterior (veure [10, 11]):

- Millora la part que l'ordre n no és potència de dos. Doncs abans, en aquest suposit, es cridava a les funcions específiques per casos. Per tant només es podien construir les matrius Hadamard per certs valors de t i s esmentats anteriorment en els comentaris de la funció `HadamardRankKernelNNotPower2(n,r,k)`. Ara es podrà construir qualsevol matriu Hadamard d'ordre no potència de dos sempre que el seu ordre primigeni $n = 4s$ estigui a la base de dades Hadamard del MAGMA.

- L'opció de convertir la sortida de la llista de matrius Hadamard a llista d'enters.

Després de comprovar que l'ordre és 1, 2 o múltiple de 4; esbrina en quin dels dos possibles escenaris estem: O bé l'ordre és una potència de dos ($n = 2^t$, $t \geq 0$), o bé l'ordre no és una potència de dos ($n = 2^t \cdot s$, $t \geq 2$ i $s \neq 1$ senar).

- Si l'ordre és una potència de dos ($n = 2^t$, $t \geq 0$):

  - Tracta apart el cas lineal, que té el rang i la dimensió del nucli igual a $t + 1$ i sempre existeix per qualsevol valor de n potència de dos. Aquest cas inclou els ordres 1 i 2 completament.

- Tracta apart el cas d'ordre 16 ($t = 4$) per generar les quatre matrius de codis Hadamard no lineals que té:

$$(r, k) \in \{(6, 3), (7, 2), (8, 1), (8, 2)\}$$

- Per els altres casos ($t > 4$) aplica els límits en el rang i la dimensió del nucli per a la construcció de matrius Hadamard d'un ordre potència de dos donat:

$$\begin{cases} t + 2 \leq r \leq 2^{t+1-k} + k - 1 & \text{si} \quad 3 \leq k \leq t - 1 \\ t + 3 \leq r \leq 2^{t-1} & \text{si} \quad 1 \leq k \leq 2 \end{cases} \tag{5.1}$$

- En qualsevol cas cridem a `HadamardRankKernelNPower2(n,r,k)`

- Si l'ordre no és una potència de dos ($n = 2^t \cdot s$, $t \geq 2$ i $s \neq 1$ senar):
  - Comprova que l'ordre $4s$ existeix a la base de dades Hadamard del MAGMA, preguntant-li el nombre de matrius que té d'ordre $4s$ amb la funció `NumberOfMatrices(HadamardDatabase(),4*s)`.
  - Tracta apart el cas $n = 4s$ ($t = 2$), perquè és un cas especial que té el rang i la dimensió del nucli igual a $4s - 1$ i 1 respectivament.
  - Per els altres casos $n = 2^t \cdot s$ ($t > 2$), aplica els límits en el rang i la dimensió del nucli per a la construcció de matrius Hadamard d'un ordre no potència de dos donat:

$$4s + t - 3 \leq r \leq \begin{cases} 2^{t+1-k} \cdot s + k - 1 & \text{si} \quad 3 \leq k \leq t - 1 \\ 2^{t-1} \cdot s & \text{si} \quad 1 \leq k \leq 2 \end{cases} \tag{5.2}$$

  - En qualsevol cas cridem a `HadamardRankKernelNNotPower2(n,r,k)`

```
/*****************************************************************************************

HadamardAllRankKernel: RngIntElt : BoolElt -> [AlgMatElt] (optionally [RngIntElt])
Given a positive interger n, returns a list of Hadamard matrices of degree n. There is
one Hadamard matrix with rank r and kernel of dimension k, for each possible pair (r,k).
Optionally be returned to integer conversion if boolean parameter swint is set to true,
it defaults to false.

*****************************************************************************************/

HadamardAllRankKernel:=function(n: swint:=false)

    if ((Type(n) eq RngIntElt) and (n gt 0)) then
        if ((n eq 1) or (n eq 2) or (IsDivisibleBy(n,4))) then

            t,s:=Valuation(n,2);

            //case power of 2
            if(s eq 1)then
                //for all n it exists a Hadamard Matrix with rank and kernel equal to t+1
                H:=HadamardRankKernelNPower2(n,t+1,t+1);
                if (not swint) then Hseq:=[H]; else Hseq:=[HadamardMatrixToInteger(H)]; end if;

                // case n=1
                if t eq 0 then
                    return Hseq;
                elif t eq 4 then
                //for t = 4 this matrices are defined
                    H:=HadamardRankKernelNPower2(16,6,3);
                    if swint then H:=[HadamardMatrixToInteger(H)]; end if;
                    Append(~Hseq,H);
                    H:=HadamardRankKernelNPower2(16,7,2);
                    if swint then H:=[HadamardMatrixToInteger(H)]; end if;
                    Append(~Hseq,H);
                    H:=HadamardRankKernelNPower2(16,8,2);
                    if swint then H:=[HadamardMatrixToInteger(H)]; end if;
                    Append(~Hseq,H);
                    H:=HadamardRankKernelNPower2(16,8,1);
                    if swint then H:=[HadamardMatrixToInteger(H)]; end if;
                    Append(~Hseq,H);
                else
                    for k:=1 to 2 do
                        for r:=t+3 to 2^(t-1) do
                            //t+3 <= r <= 2^(t-1) if 1 <= k <= 2
                            H:=HadamardRankKernelNPower2(n,r,k);
                            if swint then H:=[HadamardMatrixToInteger(H)]; end if;
                            Append(~Hseq,H);
```

```
                    end for;
                end for;
                for k:=3 to t-1 do
                    for r:=t+2 to 2^(t+1-k)+k-1 do
                        //t+2 <= r <= 2^(t+1-k)+k-1 if 3 <= k <= t-1
                        H:=HadamardRankKernelNPower2(n,r,k);
                        if swint then H:=[HadamardMatrixToInteger(H)]; end if;
                        Append(~Hseq,H);
                    end for;
                end for;
            end if;

        //case 4s
        elif (NumberOfMatrices(HadamardDatabase(),4*s) ge 1) then
            Hseq:=[];
            case n:
                when (4*s):
                    H:=HadamardRankKernelNNotPower2(n,n-1,1);
                    if swint then H:=[HadamardMatrixToInteger(H)]; end if;
                    Append(~Hseq,H);
                else:
                    rmin:=(4*s)+t-3;
                    for k:=1 to t-1 do
                        if (k le 2) then
                            rmax:=2^(t-1)*s;
                        else
                            rmax:=(2^(t+1-k)*s)+k-1;
                        end if;
                        for r:=rmin to rmax do
                            H:=HadamardRankKernelNNotPower2(n,r,k);
                            if swint then H:=[HadamardMatrixToInteger(H)]; end if;
                            Append(~Hseq,H);
                        end for;
                    end for;
                end case;
        else
            error "Runtime error in 'HadamardAllRankKernel': First Hadamard matrix of sequence\\
                   is not in Database";
        end if;
    else
        error "Runtime error in 'HadamardAllRankKernel': n must be 1, 2 or multiple of 4";
    end if;
 else
    error "Runtime error in 'HadamardAllRankKernel': Argument 1 is not greater than 0";
end if;
 return Hseq;
end function;
```

`HadamardAllRankKernelFile(n : swint:=false)`

Aquesta funció és la mateixa que l'anterior `HadamardAllRankKernel(n)`, però com el seu nom indica retorna un fitxer.

Es a dir, guarda en un fitxer amb nom "HadamardMatrix<n>" una llista de matrius Hadamard d'ordre n, n'hi ha una per cada possible valor de rang r i dimensió del nucli k compatibles amb l'ordre. Opcionalment les pot convertir a enters, mitjançant la funció `HadamardMatrixToInteger(H)`, si activem el paràmetre opcional *swint*.

```
/***************************************************************************************

HadamardAllRankKernelFile: RngIntElt : BoolElt
Given a positive interger n, saves in a file called "HadamardMatrix<n>" Hadamard matrices of
degree n. There is one Hadamard matrix with rank r and kernel of dimension k, for each possible
pair (r,k). Optionally be returned to integer conversion if boolean parameter swint is set to
true, it defaults to false.
***************************************************************************************/

HadamardAllRankKernelFile:=procedure(n : swint:=false)

if ((Type(n) eq RngIntElt) and (n gt 0)) then
    if ((n eq 1) or (n eq 2) or (IsDivisibleBy(n,4))) then

      t,s:=Valuation(n,2);
      FileName:="HadamardMatrix" cat IntegerToString(n);
      if swint then regint:="["; end if;

        //case power of 2
        if(s eq 1)then
            //for all n it exists a Hadamard Matrix with rank and kernel equal to t+1
            H:=HadamardRankKernelNPower2(n,t+1,t+1);
            if (not swint) then
              PrintFile(FileName,"\n Rank: " cat IntegerToString(t+1) cat\\
               ", Kernel Dimension: " cat IntegerToString(t+1));
              PrintFile(FileName,H);
            else
              PrintFile(FileName,regint);
              regint:=IntegerToString(HadamardMatrixToInteger(H)) cat ",";
            end if;

            // case n=1
            if t eq 0 then

            elif t eq 4 then
```

```
//for t = 4 this matrices are defined

     H:=HadamardRankKernelNPower2(16,6,3);
     if (not swint) then
      PrintFile(FileName,"\n Rank: " cat IntegerToString(6) cat\\
       ", Kernel Dimension: " cat IntegerToString(3));
      PrintFile(FileName,H);
     else
      PrintFile(FileName,regint);
      regint:=IntegerToString(HadamardMatrixToInteger(H)) cat ",";
     end if;
     H:=HadamardRankKernelNPower2(16,7,2);
     if (not swint) then
      PrintFile(FileName,"\n Rank: " cat IntegerToString(7) cat\\
       ", Kernel Dimension: " cat IntegerToString(2));
      PrintFile(FileName,H);
     else
      PrintFile(FileName,regint);
      regint:=IntegerToString(HadamardMatrixToInteger(H)) cat ",";
     end if;
     H:=HadamardRankKernelNPower2(16,8,2);
     if (not swint) then
      PrintFile(FileName,"\n Rank: " cat IntegerToString(8) cat\\
       ", Kernel Dimension: " cat IntegerToString(2));
      PrintFile(FileName,H);
     else
      PrintFile(FileName,regint);
      regint:=IntegerToString(HadamardMatrixToInteger(H)) cat ",";
     end if;
     H:=HadamardRankKernelNPower2(16,8,1);
     if (not swint) then
      PrintFile(FileName,"\n Rank: " cat IntegerToString(8) cat\\
       ", Kernel Dimension: " cat IntegerToString(1));
      PrintFile(FileName,H);
     else
      PrintFile(FileName,regint);
      regint:=IntegerToString(HadamardMatrixToInteger(H)) cat ",";
     end if;

 else
     for k:=1 to 2 do
         for r:=t+3 to 2^(t-1) do
             //t+3 <= r <= 2^(t-1) if 1 <= k <= 2
             H:=HadamardRankKernelNPower2(n,r,k);
             if (not swint) then
              PrintFile(FileName,"\n Rank: " cat IntegerToString(r) cat\\
               ", Kernel Dimension: " cat IntegerToString(k));
```

```
                PrintFile(FileName,H);
               else
                PrintFile(FileName,regint);
                regint:=IntegerToString(HadamardMatrixToInteger(H)) cat ",";
               end if;
           end for;
       end for;
       for k:=3 to t-1 do
           for r:=t+2 to 2^(t+1-k)+k-1 do
               //t+2 <= r <= 2^(t+1-k)+k-1 if 3 <= k <= t-1
               H:=HadamardRankKernelNPower2(n,r,k);
               if (not swint) then
                PrintFile(FileName,"\n Rank: " cat IntegerToString(r) cat\\
                 ", Kernel Dimension: " cat IntegerToString(k));
                PrintFile(FileName,H);
               else
                PrintFile(FileName,regint);
                regint:=IntegerToString(HadamardMatrixToInteger(H)) cat ",";
               end if;
           end for;
       end for;
   end if;

//case 4s
elif (NumberOfMatrices(HadamardDatabase(),4*s) ge 1) then
   case n:
       when (4*s):
           H:=HadamardRankKernelNNotPower2(n,n-1,1);
           if (not swint) then
            PrintFile(FileName,"\n Rank: " cat IntegerToString(n-1) cat\\
             ", Kernel Dimension: " cat IntegerToString(1));
            PrintFile(FileName,H);
           else
            PrintFile(FileName,regint);
            regint:=IntegerToString(HadamardMatrixToInteger(H)) cat ",";
           end if;

       else:
           rmin:=(4*s)+t-3;
           for k:=1 to t-1 do
               if (k le 2) then
                   rmax:=2^(t-1)*s;
               else
                   rmax:=(2^(t+1-k)*s)+k-1;
               end if;
               for r:=rmin to rmax do
                   H:=HadamardRankKernelNNotPower2(n,r,k);
```

```
                        if (not swint) then
                         PrintFile(FileName,"\n Rank: " cat IntegerToString(r) cat\\
                          ", Kernel Dimension: " cat IntegerToString(k));
                         PrintFile(FileName,H);
                        else
                         PrintFile(FileName,regint);
                         regint:=IntegerToString(HadamardMatrixToInteger(H)) cat ",";
                        end if;
                    end for;
                end for;
            end case;
        else
            error "Runtime error in 'HadamardAllRankKernelFile': First Hadamard matrix of sequence\\
                   is not in Database";
        end if;
    else
        error "Runtime error in 'HadamardAllRankKernelFile': n must be 1,2 or multiple of 4";
    end if;
else
    error "Runtime error in 'HadamardAllRankKernelFile':
    Argument 1 is not greater than 0";
end if;
if swint then
  PrintFile(FileName,Substring(regint,1,#regint-1));
  PrintFile(FileName,"]");
end if;
end procedure;
```

`HadamardRankKernel(n,r,k: swint:=false)`

Donats l'ordre n, el rang r i la dimensió del nucli k, aquesta funció retorna una matriu Hadamard amb aquestes característiques. És a dir, retorna una matriu Hadamard d'ordre n, en la qual el seu codi té rang r i la dimensió del nucli és k. Integra la funcionalitat de les funcions auxiliars `HadamardRankKernelNPower2(n,r,k)` pel cas de $n = 2^t$, $t \geq 0$ i `HadamardRankKernelNNotPower2(n,r,k)` pel cas de $n = 2^t \cdot s$, $t \geq 2$ i $s \neq 1$ senar. L'algorisme utilitzat és el següent:

- Comprova la correcció dels valors dels paràmetres d'entrada. És a dir, si l'ordre n, el rang r i la dimensió del nucli k són enters positius.

- Comprova si existeix una matriu Hadamard amb aquest ordre n , rang r i dimensió del nucli k mitjançant la funció ExistsHadamardRankKernel(n,r,k).

- Comprova si es troba en el cas potència de dos amb `t,s:=Valuation(n,2)`, que ens calcula l'exponent t i el factor senar s quan descomposem n en potències de 2.

- Si és potència de dos crida a la funció `HadamardRankKernelNPower2(n,r,k)`, sinó a la funció `HadamardRankKernelNNotPower2(n,r,k)`; en passant-les hi els mateixos paràmetres.

```
/*************************************************************************************************

HadamardRankKernel: RngIntElt,RngIntElt,RngIntElt:BoolElt->AlgMatElt (optionally RngIntElt)
Given positive integers n, r and k, returns a Hadamard matrix of degree n such that its
corresponding Hadamard code of length n has rank r and kernel of dimension k. Optionally to
integer conversion if boolean parameter swint is set to true, it defaults to false.

*************************************************************************************************/

HadamardRankKernel:=function(n,r,k: swint:=false)
    if ((Type(n) eq RngIntElt) and (n gt 0)) then
        if((Type(r) eq RngIntElt) and (r gt 0)) then
            if((Type(k) eq RngIntElt) and (k gt 0)) then

                if (ExistsHadamardRankKernel(n,r,k)) then

                    t,s:=Valuation(n,2);
                    if (s eq 1) then
                       // NPower2-case (n=2^(t))
                       H:=HadamardRankKernelNPower2(n,r,k);
                    elif (NumberOfMatrices(HadamardDatabase(),4*s) ge 1) then
                       // NNotPower2-case (n=2^(t).s): the first 4s must exist in DB
                       H:=HadamardRankKernelNNotPower2(n,r,k);
                      else
                           error "Runtime error in 'HadamardRankKernel':
                           First Hadamard matrix of sequence is not in Database";
                    end if;

                else
                    error "Runtime error in 'HadamardRankKernel':
                    Impossible construction with these parameters";
                end if;
```

```
            else
                error "Runtime error in 'HadamardRankKernel':
                Argument 3 is not greater than 0";
            end if;
        else
            error "Runtime error in 'HadamardRankKernel':
            Argument 2 is not greater than 0";
        end if;
    else
        error "Runtime error in 'HadamardRankKernel':
       Argument 1 is not greater than 0";
    end if;

    if (not swint) then return H; else return HadamardMatrixToInteger(H); end if;

end function;
```

`HadamardRandomMatrix(n: swint:=false)`

Donat un ordre n, aquesta funció retorna una matriu Hadamard qualsevol d'aquest ordre, amb el rang r i dimensió del nucli k arbitraris però correctes i compatibles amb l'ordre n. L'algorisme utilitzat és el següent:

- Comprova que l'ordre n sigui un enter positiu 1, 2 o un múltiple de 4.
- Comprova si té matrius Hadamard d'aquest ordre n a la base de dades Hadamard del MAGMA mitjançant la funció `NumberOfMatrices(D,n)`.
- Si en té de matrius a la base de dades, llavors en retorna una qualsevol fent un `Random(1,NumberOfMatrices(HadamardDatabase(),n))` sobre el nombre de matrius d'aquest ordre que té guardades en la base de dades Hadamard del MAGMA.
- Si no en té de matrius Hadamard a la base de dades del MAGMA, llavors l'ha de construir. Per fer-ho, comprova si es troba en el cas potència de dos mitjançant `t,s:=Valuation(n,2)`.
  - Si l'ordre és una potència de dos ($n = 2^t$, $t \geq 0$):

* Busca una dimensió del nucli k arbitrària entre $\{1, 2, \ldots, t-1, t+1\}$ (alerta, el valor t està exclòs) amb `k:=Random(Setseq(k: k in Exclude([1..t+1],t)));`
* En el cas lineal ($k = t+1$) no hi ha arbitrarietat, el rang té el mateix valor que la dimensió del nucli.
* Per els altres casos aplica els límits en el rang i la dimensió del nucli per a la construcció de matrius Hadamard d'un ordre potència de dos. En aquest sistema de dos inequacions, la inequació que s'aplica per la determinació del rang depèn del valor de la dimensió del nucli (com si fos la variable independent):

$$\begin{cases} t+2 \leq r \leq 2^{t+1-k} + k - 1 & \text{si} \quad 3 \leq k \leq t-1 \\ t+3 \leq r \leq 2^{t-1} & \text{si} \quad 1 \leq k \leq 2 \end{cases} \tag{5.3}$$

  Obtindrem un rang r arbitrari i compatible, "randomitzant" entre el limits inferior i superior de la inequació corresponent al valor de la dimensió del nucli que hem "randomitzat" abans.

– Si l'ordre no és una potència de dos ($n = 2^t \cdot s$):

* Comprova que el cas $n = 4s$ existeix a la base de dades Hadamard del MAGMA, preguntant-li el nombre de matrius que té d'ordre $4s$ amb la funció `NumberOfMatrices(HadamardDatabase(),4*s)`.
* Per el cas concret $n = 4s$ ($t = 2$), no hi ha arbitrarietat, el rang i la dimensió del nucli són iguals a $4s - 1$ i 1 respectivament.
* Per els altres casos $n = 2^t \cdot s$ ($t > 2$): Busca una dimensió del nucli k arbitrària entre $\{1, \ldots, t-1\}$. I aplica els límits en el rang i la dimensió del nucli per a la construcció de matrius Hadamard d'un ordre no potència de dos. En aquest sistema, la inequació que s'aplica per determinar el rang depèn també

de la dimensió del nucli:

$$4s+t-3 \leq r \leq \begin{cases} 2^{t+1-k}\cdot s+k-1 & \text{si} \quad 3 \leq k \leq t-1 \\ 2^{t-1}\cdot s & \text{si} \quad 1 \leq k \leq 2 \end{cases} \tag{5.4}$$

Obtindrem un rang r arbitrari i compatible, "randomitzant" entre el limits inferior i superior de la inequació corresponent al valor de la dimensió del nucli que hem "randomitzat" abans.

- En qualsevol cas, al final, cridem a la funció `HadamardRankKernel(n,r,k)` per tal de construir la matriu de Hadamard d'ordre n amb el rang r i dimensió del nucli k escollits arbitràriament.

```
/***************************************************************************

HadamardRandomMatrix: RngIntElt : BoolElt -> AlgMatElt (optionally RngIntElt)
Given positive integer n, returns a random Hadamard matrix of degree n.
Optionally to integer conversion if boolean parameter swint is set to true,
it defaults to false.

***************************************************************************/

HadamardRandomMatrix:=function(n: swint:=false)
     if ((Type(n) eq RngIntElt) and (n gt 0)) then
        if ((n eq 1) or (n eq 2) or (IsDivisibleBy(n,4))) then

            D:=HadamardDatabase(); NOM:=NumberOfMatrices(D,n);
            if (NOM ge 1) then
                H:=Matrix(D,n,Random(1,NOM));
            else
                t,s:=Valuation(n,2);

                //case power of 2
                if (s eq 1) then
                    k:=Random(Setseq({k: k in Exclude([1..t+1],t)}));
                    // lineal case
                    if (k eq (t+1)) then
                        r:=k;
                    //t+3 <= r <= 2^(t-1) if 1 <= k <= 2
                    elif ((k ge 1) and (k le 2)) then
                        r:=Random(t+3,2^(t-1));
                    //t+2 <= r <= 2^(t+1-k)+k-1 if 3 <= k <= t-1
```

```
                    else
                        r:=Random(t+2,2^(t+1-k)+k-1);
                    end if;

                // case not power of 2, the first (4s) must exist in DB
                elif (NumberOfMatrices(D,4*s) ge 1) then
                    //t=2, r=4s-1, k=1 (case 4s)
                    if (t eq 2) then
                        r:=4*s-1;k:=1;
                    else
                        k:=Random(1,t-1);
                        //t>=3, r=4s+t-3..n/2, k=1,2
                        if ((k ge 1) and (k le 2)) then
                            r:=Random(4*s+t-3,2^(t-1)*s);
                        //t>3, r=4s+t-3..2^(t+1-k)s+k-1, k>2
                        else
                            r:=Random(4*s+t-3,2^(t+1-k)*s+k-1);
                        end if;
                    end if;
                else
                    error "Runtime error in 'HadamardRandomMatrix': First Hadamard matrix of sequence\\
                           is not in Database";
                end if;
                H:=HadamardRankKernel(n,r,k);
            end if;

        else
            error "Runtime error in 'HadamardRandomMatrix': n must be 1, 2 or multiple of 4";
        end if;
    else
        error "Runtime error in 'HadamardRandomMatrix': Argument is not greater than 0";
    end if;

    if (not swint) then return H; else return HadamardMatrixToInteger(H); end if;

end function;
```

5.1.4 Altres funcions

`CCGHadamardDatabase()`

```
/*****************************************************************************************
CCGHadamardDatabase: : MonStgElt -> DB
Returns the database of Hadamard matrices.
*****************************************************************************************/
```

5.2 Pas de codi extern a *Package*

Hem de muntar una llibreria interna o *package* amb totes les funcions implementades, i actualitzar el manual Hadamard de l'usuari (capitol 8) amb les funcions implementades que són públiques o d'usuari.

Mostrem un exemple d'una d'aquestes funcions d'usuari, explicarem els canvis que s'han de fer per passar-la al *package*, i finalment la mostrem ja convertida per al *package*.

1. En la declaració de la funció hem de canviar la paraula reservada `function` per `intrinsic`, i també al final de la funció `end function` per `end intrinsic`.

2. Hem de treure el codi que comprova la correcció dels paràmetres d'entrada, ja que posarem els seus tipus de variable acompanyant als arguments de la funció després de "::", i també el tipus que la funció retorna després de "->".

3. Adjuntem una breu explicació de la funció entre claus "{...}".

4. Per les possibles condicions inicials o requeriments en el valors dels paràmetres d'entrada, s'ha de extreure el codi corresponent del programa i posar-lo en les comandes:

   - `require condició:  missatge_error;`
   - `requirerange variable, limit_inferior, limit_superior;`
   - `requirege variable, limit_inferior;`

```
intrinsic HadamardInvariantSHDE(H::AlgMatElt : a:=3.1415926 ,k:=4) -> SeqEnum[RngIntElt]
{Given a Hadamard matrix H, a ponderator positive number a and a positive integer k,
returns only the frequencies of the Symmetric Hamming Distance Enumerator
distribution of H over all k-dimensional columns projections. Invariant
proposed by Kai-Tai Fang and Gennian Ge. It defaults a to 3.1415926 and k to 4.}

    require IsHadamard(H):"Argument is not a Hadamard matrix";
    requirege k, 1;
    require k le NumberOfColumns(H):"Argument 3 is greater than the number of columns of Hadamard matrix";
    require ((Type(a) eq FldReElt) and (a gt 0)):"Argument 2 is not greater than 0";
    FiBak:=SHDEDistribution(a,k,H);
```

```
    FG:=[]; for i:=1 to #FiBak do Append(~FG,FiBak[i][2]); end for;
    return FG;
end intrinsic;
```

En el procés de correcció dels errors al passar les funcions d'usuari a intrínsics del nou `package` utilitzarem aquestes comandes:

- Per carregar el *package* s'utilitza la comanda Attach: `Attach("hadamard.m");`. Al fer això es crearan automàticamnent dos fitxers: `hadamard.dat` i `hadamard.sig`.
- I per descarregar-lo s'utilitza la comanda `Dettach;`.
- Després d'haver depurat el *package* posarem la comanda `freeze;` al començament del fitxer per indicar al MAGMA que la versió ja és definitiva i que no caldrà recompilar-lo cada cop que fem el `Attach`.

Tots els *packages* es troben en el directori `/package` de la distribució de MAGMA. Per instal·lar-hi el nou `package` d'aquest projecte cal seguir els passos següents:

1. Crear un subdirectori /Hadamard.
2. Copiar a aquesta carpeta els fitxers `hadamard.m` i `hadamard.sig`.
3. Crear el fitxer d'especificacions `hadarmard.spec`. Aquest fitxer només conté la informació següent:
   {
   hadamard.m
   }
4. Modificar el fitxer `/package/spec` per afegir la informació del package Hadamard
   {
   +hadamard.spec
   }

5.3 Test de proves

Per tal de comprovar la correció i la fiabilitat de les funcions implementades, hem dissenyat una sèrie de tests. En el projecte s'han realitzat dos tipus de tests: tests unitaris i tests d'integració.

5.3.1 Test unitari

El test unitari serveix per validar un mòdul, fent passar el test per totes les línies de codi. No només hem de comprovar que la resposta de la funció és correcta quan la cridem amb els paràmetres correctes, sinó també que els errors són els esperats. Per tant, hem de dissenyar-lo per tots el casos possibles, correctes i erronis.

Hem implementat test unitaris bastant exhaustius per cadascuna de les vuit funcions.

1. `KernelZ2(C)` : test unitari `unittestKernelZ2.m`
2. `HadamardInvariantSHDE(H)` : `unittestHadamardInvariantSHDE.m`
3. `HadamardInequivalentMatricesSHDE(H1,H2)` : `unittestHadamardInequivalentMatricesSHDE.m`
4. `HadamardRankKernelNNotPower2(n,r,k)` : `unittestHadamardRankKernelNNotPower2.m`
5. `HadamardRankKernel(n,r,k)` : `unittestHadamardRankKernel.m`
6. `HadamardRandomMatrix(n)` : `unittestHadamardRandomMatrix.m`
7. `HadamardAllRankKernelFile(n)` : `unittestHadamardAllRankKernelFile.m`
8. `HadamardAllRankKernel(n)` : `unittestHadamardAllRankKernel.m`

També hem implementat un procediment general `unittestHadamard(option)` que engloba tots els tests unitaris de cadascuna de les vuit funcions. Triant l'opció pertinent podrem realitzar el test unitari de la funció que vulguem o de totes (opció 0). També, si volguéssim, podrem guardar els resultats a un fitxer.

Per fer-nos una idea i no carregar excessivament de codi MAGMA la memòria, mostrem només el procediment `unittestHadamardRankKernelNNotPower2`. Aquest i tots els altres tests unitaris estan gravats al CD que acompanya la memòria.

```
print "HadamardRankKernelNNotPower2 ...";

//Correct case: n=4s (12,20,28,36,44,52,...,63)
print "\n    4s case ...";
//read pause,"    press <Enter> to continue";
for s:=3 to 63 by 2 do
    n:=4*s;
    print "\n    H(" cat IntegerToString(n) cat "," cat IntegerToString(n-1) cat\\
         "," cat IntegerToString(1) cat ")";
    //read pause,"    press <Enter> to continue";
    HadamardRankKernelNNotPower2(n,n-1,1);
end for;

//Correct case: n=8s (24,40,56,72,88,104,...,31)
print "\n    8s case ...";
//read pause,"press <Enter> to continue";
for s:=3 to 31 by 2 do
    n:=8*s;
    print "\n    H(" cat IntegerToString(n) cat "," cat IntegerToString(n div 2) cat\\
         "," cat IntegerToString(1) cat ")";
    HadamardRankKernelNNotPower2(n,(n div 2),1);
    print "\n    H(" cat IntegerToString(n) cat "," cat IntegerToString(n div 2) cat\\
         "," cat IntegerToString(2) cat ")";
    HadamardRankKernelNNotPower2(n,(n div 2),2);
end for;

//Correct case: white0 single: H(n,r(=rmin),1) = S x [H(n/2,r-1,1),H(n/2,r-1,2)]
print "\n    H(n,r(=rmin),1) = S x [H(n/2,r-1,1),H(n/2,r-1,2)] case ...";
//read pause,"press <Enter> to continue";
n:=48;
while (n le 256) do
    t,s:=Valuation(n,2);
    if ((s ne 1) and (t gt 3)) then
        print "\n    H(" cat IntegerToString(n) cat "," cat IntegerToString(4*s+t-3) cat\\
             "," cat IntegerToString(1) cat ")";
        HadamardRankKernelNNotPower2(n,(4*s+t-3),1);
    end if;
    n:=n+4;
end while;

//Correct case: aster6 case H(n,r,k) = S x H(n/2,r-1,k-1)
print "\n    H(n,r,k) = S x H(n/2,r-1,k-1) case ...";
//read pause,"press <Enter> to continue";
n:=24;
while (n le 128) do
    t,s:=Valuation(n,2);
    if ((s ne 1) and (t ge 3)) then
       rmin:=(4*s)+t-3;
       for k:=1 to t-1 do
         if (k le 2) then rmax:=2^(t-1)*s; else rmax:=(2^(t+1-k)*s)+k-1; end if;
         for r:=rmin to rmax do
             if (ExistsHadamardRankKernel(n,r,k)) then
                 print "\n    H(" cat IntegerToString(2*n) cat "," cat IntegerToString(r+1) cat\\
                     "," cat IntegerToString(k+1) cat ")";
                 HadamardRankKernelNNotPower2((2*n),(r+1),(k+1));
             end if;
         end for;
       end for;
    end if;
    n:=n+4;
end while;

//Correct case: rombo case H(n,r,2) = Transpose(H(n,r,1))
print "\n    H(n,r,2) = Transpose(H(n,r,1)) case ...";
```

```
//read pause,"press <Enter> to continue";
n=48;
while (n le 256) do
    t,s:=Valuation(n,2);
    if ((s ne 1) and (t gt 3)) then
       rmin:=(4*s)+t-3; rmax:=2^(t-1)*s;
         for r:=rmin to rmax do
             if (r ge ((n div 4)+2)) then
                 print "\n    H(" cat IntegerToString(n) cat "," cat IntegerToString(r) cat\\
                      "," cat IntegerToString(2) cat ")";
                 HadamardRankKernelNNotPower2(n,r,2);
             end if;
         end for;
    end if;
    n:=n+4;
end while;

//Correct case: black0 single case H(n,r(=n/4+2),3) = switching code of H(n,r-1,3)
print "\n    H(n,r(=n/4+2),3) = switching code of H(n,r-1,3) case ...";
//read pause,"press <Enter> to continue";
n:=48;
while (n le 256) do
    t,s:=Valuation(n,2);
    if ((s ne 1) and (t gt 3)) then
       rmin:=(4*s)+t-3; rmax:=(2^(t-2)*s)+2;
         for r:=rmin to rmax do
             if (r eq ((n div 4)+2)) then
                 print "\n    H(" cat IntegerToString(n) cat "," cat IntegerToString(r) cat\\
                      "," cat IntegerToString(3) cat ")";
                 HadamardRankKernelNNotPower2(n,r,3);
             end if;
         end for;
    end if;
    n:=n+4;
end while;

//Correct case: aster5 permutations H(n,r,1)=Sx[H(n/2,r-2,1),perm(H(n/2,r-2,1))]
print "\n    permutations H(n,r,1)=Sx[H(n/2,r-2,1),perm(H(n/2,r-2,1))] case ...";
//read pause,"press <Enter> to continue";
n:=48;
while (n le 256) do
    t,s:=Valuation(n,2);
    if ((s ne 1) and (t gt 3)) then
       rmin:=(4*s)+t-3; rmax:=2^(t-1)*s;
         for r:=rmin to rmax do
                rper1:=rmin+1; rper2:=(n div 4)+2;
                if (r ge rper1) then
                    print "\n    H(" cat IntegerToString(n) cat "," cat IntegerToString(r) cat\\
                         "," cat IntegerToString(1) cat ")";
                    HadamardRankKernelNNotPower2(n,r,1);
                end if;
         end for;
    end if;
    n:=n+4;
end while;

//Error case: lost in recursion, n=260 (4s with s=65, not in HadamardDatabase)
print "\n    lost in recursion, n=260 (4s with s=65, not in HadamardDatabase) case ...";
//read pause,"press <Enter> to continue";
print "\n    H(" cat IntegerToString(260) cat "," cat IntegerToString(259) cat "," cat IntegerToString(1) cat ")";
HadamardRankKernelNNotPower2(260,259,1);
```

5.3.2 Test d'integració

El test d'integració verifica la integració dels diferents mòduls que participen en les funcions, des de les auxiliars més importants fins arribar a les públiques o d'usuari. És per això que hem implementat un test d'integració per cadascuna d'aquestes tres funcions importants:

1. `HadamardRankKernelNPower2(n,r,k)` : `testHadamardRankKernelNNotPower2.m`
2. `HadamardRankKernelNNotPower2(n,r,k)` : `testHadamardRankKernelNNotPower2.m`
3. `HadamardRankKernel(n,r,k)` : `testHadamardRankKernel.m`

Mostrarem només el codi del procediment `testHadamardRankKernel.m` que verifica la integració de la funció `HadamardRankKernel(n,r,k)` que, de fet, es com si féssim també un test de integració de les altres, perquè són cridades per la primera. Tots els procediments tests de integració estan gravats al CD que acompanya la memòria.

En el test `testHadamardRankKernel(nmax)` comprovem totes les matrius Hadamard de tots els orders n fins a l'ordre màxim entrat $nmax$, que pot construir la funció `HadamardRankKernel(n,r,k)` per tots els rangs r i dimensions del nucli possibles k. I, a més, comprovem que la matriu retornada sigui realment de rang r i dimensió del nucli k. La única limitació serà els recursos de càlcul i de memòria de que disposi el nostre procés.

Els resultats els podrem guardar a un fitxer que per defecte té el nom "testHadRanKer<nmax>.txt".

```
/***************************************************************************
testHadamardRankKernel: RgnIntElt (,MonStgElt)
Given a positive integer as the maximun order, tests in a file the Hadamard matrices
until this order generated by the general function 'HadamardRankKernel'.
***************************************************************************/
testHadamardRankKernel:=procedure(nmax: file:="testHadRanKer" cat IntegerToString(nmax) cat ".txt")

fprintf file,"testing HadamardRankKernel ... \n";

n:=1;
while (n le nmax) do
    t,s:=Valuation(n,2);
    if (s eq 1) then

    //for all n=2^t it exists a Hadamard Matrix with rank and kernel equal to t+1
    // includes completely the only lineal orders (n=1,2,4,8)
```

```
H:=HadamardRankKernel(n,t+1,t+1);
H2:=HadamardMatrixFromInteger(HadamardRankKernel(n,t+1,t+1: swint:=true),n);
fprintf file,"  n=%o, r=%o, k=%o  ->  ",IntegerToString(n),IntegerToString(t+1),IntegerToString(t+1);
fprintf file,"%o ",InvariantsRankKernelZ2(HadamardMatrixToCode(H2)) eq [t+1, t+1];
fprintf file,"%o \n",H eq H2;

  if t lt 4 then

  elif t eq 4 then
  //for t = 4 this matrices are defined
    H:=HadamardRankKernel(n,6,3);
    H2:=HadamardMatrixFromInteger(HadamardRankKernel(n,6,3: swint:=true),n);
    fprintf file,"  n=%o, r=%o, k=%o  ->  ",IntegerToString(n),IntegerToString(6),IntegerToString(3);
    fprintf file,"%o ",InvariantsRankKernelZ2(HadamardMatrixToCode(H2)) eq [6, 3];
    fprintf file,"%o \n",H eq H2;

    H:=HadamardRankKernel(n,7,2);
    H2:=HadamardMatrixFromInteger(HadamardRankKernel(n,7,2: swint:=true),n);
    fprintf file,"  n=%o, r=%o, k=%o  ->  ",IntegerToString(n),IntegerToString(7),IntegerToString(2);
    fprintf file,"%o ",InvariantsRankKernelZ2(HadamardMatrixToCode(H2)) eq [7, 2];
    fprintf file,"%o \n",H eq H2;

    H:=HadamardRankKernel(n,8,2);
    H2:=HadamardMatrixFromInteger(HadamardRankKernel(n,8,2: swint:=true),n);
    fprintf file,"  n=%o, r=%o, k=%o  ->  ",IntegerToString(n),IntegerToString(8),IntegerToString(2);
    fprintf file,"%o ",InvariantsRankKernelZ2(HadamardMatrixToCode(H2)) eq [8, 2];
    fprintf file,"%o \n",H eq H2;

    H:=HadamardRankKernel(n,8,1);
    H2:=HadamardMatrixFromInteger(HadamardRankKernel(n,8,1: swint:=true),n);
    fprintf file,"  n=%o, r=%o, k=%o  ->  ",IntegerToString(n),IntegerToString(8),IntegerToString(1);
    fprintf file,"%o ",InvariantsRankKernelZ2(HadamardMatrixToCode(H2)) eq [8, 1];
    fprintf file,"%o \n",H eq H2;

  else
     for k:=1 to 2 do
       for r:=t+3 to 2^(t-1) do
       //t+3 <= r <= 2^(t-1) if 1 <= k <= 2
         H:=HadamardRankKernel(n,r,k);
         H2:=HadamardMatrixFromInteger(HadamardRankKernel(n,r,k: swint:=true),n);
         fprintf file,"  n=%o, r=%o, k=%o  ->  ",IntegerToString(n),IntegerToString(r),IntegerToString(k);
         fprintf file,"%o ",InvariantsRankKernelZ2(HadamardMatrixToCode(H2)) eq [r, k];
         fprintf file,"%o \n",H eq H2;
       end for;
     end for;
     for k:=3 to t-1 do
       for r:=t+2 to 2^(t+1-k)+k-1 do
       //t+2 <= r <= 2^(t+1-k)+k-1 if 3 <= k <= t-1
         H:=HadamardRankKernel(n,r,k);
         H2:=HadamardMatrixFromInteger(HadamardRankKernel(n,r,k: swint:=true),n);
         fprintf file,"  n=%o, r=%o, k=%o  ->  ",IntegerToString(n),IntegerToString(r),IntegerToString(k);
         fprintf file,"%o ",InvariantsRankKernelZ2(HadamardMatrixToCode(H2)) eq [r, k];
         fprintf file,"%o \n",H eq H2;
       end for;
     end for;
  end if;

elif (NumberOfMatrices(HadamardDatabase(),4*s) ge 1) then

  case n:
    when (4*s):
        H:=HadamardRankKernel(n,n-1,1);
        H2:=HadamardMatrixFromInteger(HadamardRankKernel(n,n-1,1: swint:=true),n);
        fprintf file,"  n=%o, r=%o, k=%o  ->  ",IntegerToString(n),IntegerToString(n-1),IntegerToString(1);
```

```
            fprintf file,"%o ",InvariantsRankKernelZ2(HadamardMatrixToCode(H2)) eq [n-1, 1];
            fprintf file,"%o \n",H eq H2;

        else:
            rmin:=(4*s)+t-3;
            for k:=1 to t-1 do
              if (k le 2) then rmax:=2^(t-1)*s; else rmax:=(2^(t+1-k)*s)+k-1; end if;
              for r:=rmin to rmax do
                  H:=HadamardRankKernel(n,r,k);
                  H2:=HadamardMatrixFromInteger(HadamardRankKernel(n,r,k: swint:=true),n);
                  fprintf file,"  n=%o, r=%o, k=%o  ->  ",IntegerToString(n),IntegerToString(r),IntegerToString(k);
                  fprintf file,"%o ",InvariantsRankKernelZ2(HadamardMatrixToCode(H2)) eq [r, k];
                  fprintf file,"%o \n",H eq H2;
              end for;
            end for;
        end case;

    end if;

    if (n eq 1) then n:=n+1;
    elif (n eq 2) then n:=n+2;
    else n:=n+4;
    end if;

end while;

end procedure;
```

5.4 Proves de fiabilitat

Provar que les noves funcions donen els resultats correctes, és a dir, que les sortides són les que s'esperen obtenir en cada cas; ja ho hem fet en els tests de proves unitàries. Ara provem que els resultats obtinguts coincideixen amb els que s'obtenien amb les versions anteriors de les mateixes funcions.

5.4.1 Proves de fiabilitat de la funció `KernelZ2(C)`

Comprovarem que la nova funció `KernelZ2(C)` construeix els mateixos nuclis que la seva versió anterior. Per això hem realitzat aquest procediment que veurem a continuació anomenat `ProvaFiabilitatKernelZ2`. Aquest procediment es repassa tota la base de dades Hadamard estàndard del MAGMA i per cada matriu Hadamard comprova els nuclis i les seves dimensions del seu corresponent codi Hadamard per cada una de les dos versions de la funció `KernelZ2`.

```
//Fiability test of  KernelZ2
load "KernelZ2Ori.m";
load "hadamarfile.m";
    FileName:="fiatestKernelZ2.txt";
    SetLogFile(FileName);
    printf "\n Fiability Test KernelZ2 over HadamardDatabase\n";
    D:=HadamardDatabase();
    for n in Degrees(D) do
        V:=VectorSpace(GF(2),n); ZeroVector:=V!0;
        for m:=1 to NumberOfMatrices(D,n) do
            C:=HadamardMatrixToCode(Matrix(D,n,m));
            if (ZeroVector in C) then
                K:=KernelZ2(C); KA:=KernelZ2Ori(C); KAsub:=sub<V|KA>;
                ka:=Dimension(KAsub); k:=Dimension(K); r:=RankZ2(C);
                printf "n=%o,m=%o,#C=%o,r=%o,k=%o: %o,%o\n",n,m,#C,r,k,(#KA eq #K),(KAsub eq K);
            end if;
        end for;
    end for;
    UnsetLogFile();
```

Un exemple de la sortida de comprovació que s'obté d'aquest procediment és:

```
Fiability Test KernelZ2 over HadamardDatabase
n=1,m=1,#C=2,r=1,k=1: true,true
n=2,m=1,#C=4,r=2,k=2: true,true
n=4,m=1,#C=8,r=3,k=3: true,true
```

```
n=8,m=1,#C=16,r=4,k=4: true,true
n=12,m=1,#C=24,r=11,k=1: true,true
n=16,m=1,#C=32,r=5,k=5: true,true
n=16,m=2,#C=32,r=8,k=1: true,true
n=16,m=3,#C=32,r=8,k=2: true,true
n=16,m=4,#C=32,r=6,k=3: true,true
n=16,m=5,#C=32,r=7,k=2: true,true
n=20,m=1,#C=40,r=19,k=1: true,true
.
.
.
n=208,m=1,#C=416,r=53,k=3: true,true
n=212,m=1,#C=424,r=211,k=1: true,true
n=216,m=1,#C=432,r=108,k=2: true,true
n=220,m=1,#C=440,r=219,k=1: true,true
n=224,m=1,#C=448,r=112,k=1: true,true
n=228,m=1,#C=456,r=227,k=1: true,true
n=232,m=1,#C=464,r=116,k=2: true,true
n=236,m=1,#C=472,r=235,k=1: true,true
n=240,m=1,#C=480,r=120,k=1: true,true
n=244,m=1,#C=488,r=243,k=1: true,true
n=248,m=1,#C=496,r=124,k=2: true,true
n=252,m=1,#C=504,r=251,k=1: true,true
n=256,m=1,#C=512,r=9,k=9: true,true
```

5.4.2 Proves de fiabilitat de les funcions que implementen la invariant SHDE de Kai-Tai Fang i Gennian Ge

En l'article dels xinesos Kai-Tai Fang i Gennian Ge [5] no només es definia aquesta nova invariant, sinó que per demostrar la seva sensibilitat en la classificació de les matrius Hadamard, també es parlava dels resultats dels càlculs fets amb aquest invariant sobre les 60 matrius de Hadamard inequivalents d'ordre 24 i també sobre les 192 d'ordre 36, per diferents valors de la k (ja que el càlcul es fa per totes les col·leccions diferents de k columnes de n) i agafant

el número a com a 3.1415926. Aquests càlculs estaven tabul·lats en un altre article seu, més antic, del qual els autors feien esment [4] i que nosaltres no disposarem, fins que amablement ens el van fer arribar per correu electrònic.

Aquests resultats que els autors comentaven en l'article [5] i les taules dels càlculs de l'article [4], per les matrius Hadamard d'ordre 24, ens serviran per provar la fiabilitat de les funcions implementades d'aquest invariant.

Les nostres proves es fonamenten en dos resultats a comprovar que s'esmenten en el primer article:

1. Hi ha 35 classes inequivalents de matrius de Hadamard d'ordre 24 quan es calcula la invariant per $k = 4$. I, de resultes del càlcul, han quedat 38 matrius sense separar.

2. Aplicant el càlcul de la invariant amb $k = 6$ per les 38 inseparables i només comprovant la seva freqüència per el valor 34953, podrem identificar finalment les 60 matrius Hadamard inequivalents d'ordre 24.

`HadamardClassificationSHDE(24:file:=“HadClaSHDE24.txt”)` classifica les 60 matrius de Hadamard inequivalents d'ordre 24 per la $k = 4$ i $a = 3.1415926$, la seva sortida és ordenada per obtenir la Taula 5.1, que ens mostra millor les 35 classes d'equivalència que s'han creat i les matrius que pertanyen a cada classe.

- Observem que hi han 35 valors de distribució SHDE diferents amb els números de matrius d'ordre 24 associats, corresponents a les 35 classes inequivalents que identifica.

- Si comptem les matrius en les distribucions SHDE que tenen més d'una matriu associada, en resulten 38 matrius no separades. Aquestes són: 1, 10, 51, 52, 57, 58, 59, 60, 4, 8, 5, 7, 11, 12, 13, 40, 15, 38, 16, 20, 22, 31, 55, 18, 45, 19, 53, 26, 35, 28, 44, 29, 30, 37, 39, 42, 32 i 50.

Taula 5.1: Classificació de les 60 matrius Hadamard d'ordre 24 i $k = 4$.

[Floor($SHDE_{a=3.1415926,k=4}$), frequency], ...	matrices
[9410, 6600],[9495, 3960],[10168, 66]	1, 10, 51, 52, 57, 58, 59, 60
[9410, 6960],[9495, 3420],[9747, 216],[10168, 30]	2
[9410, 7260],[9495, 2970],[9747, 396]	3
[9410, 7140],[9495, 3150],[9747, 324],[10168, 12]	4, 8
[9410, 7200],[9495, 3060],[9747, 360],[10168, 6]	5, 7
[9410, 7080],[9495, 3240],[9747, 288],[10168, 18]	6
[9410, 6072],[9495, 4554]	9
[9410, 6696],[9495, 3752],[9747, 160],[10168, 18]	11, 12
[9410, 6720],[9495, 3700],[9747, 200],[10168, 6]	13, 40
[9410, 6528],[9495, 3956],[9747, 136],[10168, 6]	14
[9410, 6564],[9495, 3918],[9747, 132],[10168, 12]	15, 38
[9410, 6504],[9495, 4008],[9747, 96],[10168, 18]	16, 20, 22, 31, 55
[9410, 6600],[9495, 3880],[9747, 128],[10168, 18]	17
[9410, 6480],[9495, 4060],[9747, 56],[10168, 30]	18, 45
[9410, 6576],[9495, 3932],[9747, 88],[10168, 30]	19, 53
[9410, 6624],[9495, 3828],[9747, 168],[10168, 6]	21
[9410, 6384],[9495, 4168],[9747, 56],[10168, 18]	23
[9410, 6360],[9495, 4200],[9747, 48],[10168, 18]	24
[9410, 6408],[9495, 4136],[9747, 64],[10168, 18]	25
[9410, 6432],[9495, 4104],[9747, 72],[10168, 18]	26, 35
[9410, 6480],[9495, 4020],[9747, 120],[10168, 6]	27
[9410, 6708],[9495, 3726],[9747, 180],[10168, 12]	28, 44
[9410, 6492],[9495, 4014],[9747, 108],[10168, 12]	29, 30, 37, 39, 42
[9410, 6528],[9495, 3996],[9747, 72],[10168, 30]	32, 50
[9410, 6504],[9495, 3988],[9747, 128],[10168, 6]	33
[9410, 6516],[9495, 3982],[9747, 116],[10168, 12]	34
[9410, 6552],[9495, 3944],[9747, 112],[10168, 18]	36
[9410, 6444],[9495, 4078],[9747, 92],[10168, 12]	41
[9410, 6672],[9495, 3804],[9747, 120],[10168, 30]	43
[9410, 6336],[9495, 4212],[9747, 72],[10168, 6]	46
[9410, 6348],[9495, 4206],[9747, 60],[10168, 12]	47
[9410, 6456],[9495, 4072],[9747, 80],[10168, 18]	48
[9410, 6432],[9495, 4094],[9747, 88],[10168, 12]	49
[9410, 6528],[9495, 3976],[9747, 104],[10168, 18]	54
[9410, 6408],[9495, 4116],[9747, 96],[10168, 6]	56

El procediment `HadamardTableSHDE6024B46(a)` reprodueix la taula núm. 1 de l'article [4] per identificar totes les 60 matrius de Hadamard d'ordre 24 combinant la invariant SHDE per $k = 4$ i $k = 6$. Usa les funcions implementades `HadamardClassificationSHDE(24,a,k)` per $k = 4, 6$ i $a = 3.1415926$.

```
/*****************************************************************************************
HadamardTableSHDE6024B46:RngIntElt
Given a positive number a, saves in a file called "HadTabSHDE6024B46" the table of
the integer part of the values of the distance enumerator of H over all 4-dimensional and
6-dimensional columns |Ba4| and |Ba6| respectively of the 60 Hadamard matrices of order 24.
*****************************************************************************************/
HadamardTableSHDE6024B46:=procedure(a)
    NFiBa46:=HadamardClassificationSHDE(24:a:=a,k:=4);
    NFiBa6:=HadamardClassificationSHDE(24:a:=a,k:=6);
    for i in [1..#NFiBa46] do
        NFiBa46[i][3]:=[];
        if (#NFiBa46[i][2] gt 1) then
            for j in [1..#NFiBa46[i][2]] do
                for l in [1..#NFiBa6] do
                    if (NFiBa6[l][2][1] eq NFiBa46[i][2][j]) then

                        listed:=false;
                        for m in [1..#NFiBa6[l][1]] do
                            if (NFiBa6[l][1][m][1] eq 34953) then
                                Append(~NFiBa46[i][3],NFiBa6[l][1][m]);
                                listed:=true;
                                break;
                            end if;
                        end for;
                        if (not listed) then
                            Append(~NFiBa46[i][3],[34953,0]);
                        end if;

                    end if;
                end for;
            end for;
        else
            Append(~NFiBa46[i][3],[34953,999994]);
        end if;
    end for;
    T6024B46:=[];
    for i in [1..#NFiBa46] do
        Bk4Bk6m:=[0,0,0,0,0,0];
        for j in [1..#NFiBa46[i][1]] do
            case NFiBa46[i][1][j][1]:
                when 9410:
                    Bk4Bk6m[4]:=NFiBa46[i][1][j][2];
                when 9495:
                    Bk4Bk6m[3]:=NFiBa46[i][1][j][2];
                when 9747:
                    Bk4Bk6m[2]:=NFiBa46[i][1][j][2];
                    //10168
                else:
                    Bk4Bk6m[1]:=NFiBa46[i][1][j][2];
            end case;
        end for;
        for j in [1..#NFiBa46[i][2]] do
            Bk4Bk6m[6]:=NFiBa46[i][2][j][2];
            Bk4Bk6m[5]:=NFiBa46[i][3][j][2];
            Append(~T6024B46,Bk4Bk6m);
        end for;
    end for;
    Sort(~T6024B46);
    FileName:="HadTabSHDE6024B46";
    PrintFile(FileName,"\n HadamardTableSHDE6024B46");
    PrintFile(FileName,a);
    PrintFile(FileName,"CLASSIFICATION OF THE HADAMARD MATRICES");
    PrintFile(FileName,"Table 1. FBa,k(H) for the 60 Hadamard matrices of order 24");
    PrintFile(FileName,"No [Ba,4]=10168 [Ba,4]=9747 [Ba,4]=9495 [Ba,4]=9410 [Ba,6]=34953");
```

```
    for i in [1..#T6024B46] do
        Bk4Bk6m:=Rotate(T6024B46[i],1);
        PrintFile(FileName,Bk4Bk6m);
    end for;
end procedure;
```

Mostrem un extracte de la taula guardada en el fitxer `HadTabSHDE6024B46.txt`:

```
 HadamardTableSHDE6024B46
3.14159260000000000000000000000
CLASSIFICATION OF THE HADAMARD MATRICES
Table 1. FBa,k(H) for the 60 Hadamard matrices of order 24
No [Ba,4]=10168 [Ba,4]=9747 [Ba,4]=9495 [Ba,4]=9410 [Ba,6]=34953
[ 9, 0, 0, 4554, 6072, 999994 ]
[ 3, 0, 396, 2970, 7260, 999994 ]
[ 46, 6, 72, 4212, 6336, 999994 ]
[ 56, 6, 96, 4116, 6408, 999994 ]
[ 27, 6, 120, 4020, 6480, 999994 ]
[ 33, 6, 128, 3988, 6504, 999994 ]
[ 14, 6, 136, 3956, 6528, 999994 ]
[ 21, 6, 168, 3828, 6624, 999994 ]
[ 13, 6, 200, 3700, 6720, 6312 ]
[ 40, 6, 200, 3700, 6720, 6370 ]
[ 5, 6, 360, 3060, 7200, 11440 ]
[ 7, 6, 360, 3060, 7200, 11488 ]
.
.
.
[ 1, 66, 0, 3960, 6600, 0 ]
[ 59, 66, 0, 3960, 6600, 1080 ]
[ 52, 66, 0, 3960, 6600, 1096 ]
[ 10, 66, 0, 3960, 6600, 1100 ]
[ 58, 66, 0, 3960, 6600, 1116 ]
[ 51, 66, 0, 3960, 6600, 1120 ]
[ 57, 66, 0, 3960, 6600, 1164 ]
[ 60, 66, 0, 3960, 6600, 1184 ]
```

En La Taula 5.2 mostrem la classificació total de les 60 matrius de Hadamard inequivalents d'ordre $n = 24$ per la $k = 4, 6$ i $a = 3.1415926$. La hem obtingut del fitxer sencer gravat pel procediment `HadamardTableSHDE6024B46`, resumit a dos columnes i tret el valor convingut 999994.

- Observem que amb només un valor de la distribució SHDE per $k = 6$ és suficient per separar les 38 matrius que ens quedaven sense identificar.

Taula 5.2: Classificació de les 60 matrius Hadamard d'ordre 24 i $k = 4, 6$.

No	[Ba,4] 10168	[Ba,4] 9747	[Ba,4] 9495	[Ba,4] 9410	[Ba,6] 34953	No	[Ba,4] 10168	[Ba,4] 9747	[Ba,4] 9495	[Ba,4] 9410	[Ba,6] 34953
90	0	0	4554	6072		35	18	72	4104	6432	4308
3	0	396	2970	7260		26	18	72	4104	6432	4310
46	6	72	4212	6336		48	18	80	4072	6456	
56	6	96	4116	6408		16	18	96	4008	6504	3176
27	6	120	4020	6480		20	18	96	4008	6504	3872
33	6	128	3988	6504		55	18	96	4008	6504	3940
14	6	136	3956	6528		31	18	96	4008	6504	4020
21	6	168	3828	6624		22	18	96	4008	6504	4860
13	6	200	3700	6720	6312	54	18	104	3976	6528	
40	6	200	3700	6720	6370	36	18	112	3944	6552	
5	6	360	3060	7200	11440	17	18	128	3880	6600	
7	6	360	3060	7200	11488	12	18	160	3752	6696	5464
47	12	60	4206	6348		11	18	160	3752	6696	5624
49	12	88	4094	6432		6	18	288	3240	7080	
41	12	92	4078	6444		18	30	56	4060	6480	2808
37	12	108	4014	6492	4134	45	30	56	4060	6480	2880
30	12	108	4014	6492	4176	32	30	72	3996	6528	3366
29	12	108	4014	6492	4278	50	30	72	3996	6528	3378
42	12	108	4014	6492	4918	53	30	88	3932	6576	3600
39	12	108	4014	6492	4924	19	30	88	3932	6576	3864
34	12	116	3982	6516		43	30	120	3804	6672	
15	12	132	3918	6564	4704	2	30	216	3420	6960	
38	12	132	3918	6564	4720	1	66	0	3960	6600	0
28	12	180	3726	6708	5976	59	66	0	3960	6600	1080
44	12	180	3726	6708	6012	52	66	0	3960	6600	1096
4	12	324	3150	7140	10416	10	66	0	3960	6600	1100
8	12	324	3150	7140	10512	58	66	0	3960	6600	1116
24	18	48	4200	6360		51	66	0	3960	6600	1120
23	18	56	4168	6384		57	66	0	3960	6600	1164
25	18	64	4136	6408		60	66	0	3960	6600	1184

Aquest valor és el 34953, i per això és el únic que hem posat a la taula de $k = 6$ darrera dels altres de $k = 4$.

- Quan en alguna matriu, el valor 34953 pren la freqüència 999994, aquesta no és una freqüència. Amb això, el nostre procediment indica que no s'ha calculat perquè les freqüències per $k = 4$ són suficients per

identificar la matriu en aquest cas.

- En canvi, hi han 38 matrius on el valor 34953 pren una freqüència significativa, que serveix per identificar les matrius que no havien quedat separades per les freqüències de $k = 4$. Per exemple: Les matrius 1, 59, 52, 10, 58, 51, 57 i 60 comparteixen la mateixa distribució SHDE per $k = 4$, que són els valors 10168, 9747, 9495, 9410 amb freqüències 66, 0, 3960, 6600 respectivament. Només el valor 34953 de $k = 6$ agafa una freqüència diferent per cada matriu d'aquest cas.

- Aquesta taula coincideix amb la taula 1 de l'article [4]. L'única diferència rau en la numeració de les matrius. La nostra taula segueix la numeració de la base de dades Hadamard estàndard del MAGMA, i la seva segueix la ordenació de la pàgina web d'on van descarregar-se les matrius d'ordre 24.

5.4.3 Proves de fiabilitat de la funció que implementa la construcció de matrius Hadamard d'ordre $n = 2^t \cdot s$

Es tracta ara de provar que les matrius Hadamard construïdes per alguns ordres amb la funció auxiliar `HadamardRankKernelNNotPower2(n,r,k)` son iguals o, si més no, equivalents a les generades amb les funcions per casos concrets de $n = 2^t \cdot s$ ($t \geq 2$ i $s \neq 1$ senar) de la versió anterior, com per exemple:

- `HadamardRankKernelN12(r,k)`, `HadamardRankKernelN24(r,k)`, ...
- `HadamardRankKernelN20(r,k)`, `HadamardRankKernelN20(r,k)`, ...
- `HadamardRankKernelN28(r,k)`, `HadamardRankKernelN56(r,k)`, ...
- etc.

Amb el procediment `fiatestHadamardRankKernelNNotPower2` exhaurim totes les proves de fiabilitat possibles per els ordres no potències de dos $n = 12, 24, 48, 96, 192, 20, 40, 80, 160, 320, 28, 56, 112$. Hem de tenir cura d'executar les funcions de la versió anterior amb el seu entorn. És a dir, quan

aquestes criden a la funció `KernelZ2(C)` per construir alguna de les seves matrius per la tècnica del *switching*, ens hem d'assegurar que criden a la que retorna el nucli com a tipus llista.

Com mostrem en la sortida d'aquest procediment, es curiós comprovar que les matrius del mateix ordre construïdes per la funció actual amb les construïdes per les altres funcions de la versió anterior no tenen perquè ser iguals, ni tampoc perquè ser equivalents Hadamard (és per això que ensenyo les dues condicions). Ara bé, tenen el mateix rank i dimensió del nucli (per exemple el casos $n = 48$, $r = 18, 19, 20$ i $k = 1$).

```
/*******************************************************************************
fiatestHadamardRankKernelNNotPower2:
Fiability tests of function 'HadamardRankKernelNNotPower2' with especial functions
for 12, 24, 48, 96, 192, 20, 40, 80, 160, 320, 28, 56, 112 orders.
*******************************************************************************/
fiatestHadamardRankKernelNNotPower2:=procedure()

  for n in [12, 24, 48, 96, 192, 20, 40, 80, 160, 320, 28, 56, 112] do
      t,s:=Valuation(n,2);
      if (n eq 4*s) then
          H1:=HadamardRankKernelNNotPower2(n,n-1,1);
          IRK1:=InvariantsRankKernelZ2(HadamardMatrixToCode(H1));
          case n:
              when 12: H2:=HadamardRankKernelN12(n-1,1);
              when 20: H2:=HadamardRankKernelN20(n-1,1);
              else:    H2:=HadamardRankKernelN28(n-1,1);
          end case;
          if (IRK1 eq InvariantsRankKernelZ2(HadamardMatrixToCode(H2))) then
              printf " n=%o,r=%o,k=%o: %o,%o\n",n,n-1,1,H1 eq H2,IsHadamardEquivalent(H1,H2: A1:="Leon");
          else
              printf " n=%o,r=%o,k=%o: Different Rank-Kernel Error\n",n,n-1,1;
          end if;
      else
          rmin:=(4*s)+t-3;
          for k:=1 to t-1 do
              if (k le 2) then
                  rmax:=2^(t-1)*s;
              else
                  rmax:=(2^(t+1-k)*s)+k-1;
              end if;
              for r:=rmin to rmax do
                  H1:=HadamardRankKernelNNotPower2(n,r,k);
                  IRK1:=InvariantsRankKernelZ2(HadamardMatrixToCode(H1));
                  case n:
                      when 24:  H2:=HadamardRankKernelN24(r,k);
                      when 48:  H2:=HadamardRankKernelN48(r,k);
                      when 96:  H2:=HadamardRankKernelN96(r,k);
                      when 192: H2:=HadamardRankKernelN192(r,k);
                      when 40:  H2:=HadamardRankKernelN40(r,k);
                      when 80:  H2:=HadamardRankKernelN80(r,k);
                      when 160: H2:=HadamardRankKernelN160(r,k);
                      when 320: H2:=HadamardRankKernelN320(r,k);
                      when 56:  H2:=HadamardRankKernelN56(r,k);
                      else:     H2:=HadamardRankKernelN112(r,k);
                  end case;
                 if (IRK1 eq InvariantsRankKernelZ2(HadamardMatrixToCode(H2))) then
```

```
                printf " n=%o,r=%o,k=%o: %o,%o\n",n,r,k,H1 eq H2,IsHadamardEquivalent(H1,H2: Al:="Leon");
            else
                printf " n=%o,r=%o,k=%o: Different Rank-Kernel Error\n",n,n-1,1;
            end if;
        end for;
    end for;
  end if;
 end for;
end procedure;
```

Un troç de la sortida que dóna aquest procediment és:

```
n=12,r=11,k=1: true,true
n=24,r=12,k=1: false,true
n=24,r=12,k=2: true,true
n=48,r=13,k=1: false,true
n=48,r=14,k=1: false,true
n=48,r=15,k=1: false,true
n=48,r=16,k=1: false,true
n=48,r=17,k=1: false,true
n=48,r=18,k=1: false,false
n=48,r=19,k=1: false,false
n=48,r=20,k=1: false,false
     .
     .
     .
```

5.5 Proves de rendiment

Les proves de rendiment s'han enfocat per veure com la nova versió de la funció `KernelZ2(C)` que retorna el nucli com un subespai vectorial, és més ràpida. Per això creem el procediment `TimesKernelZ2BigNoLinear(order,rep)`.

El procediment construeix codis binaris C no lineals a partir de codis lineals quaternaris sobre $Z4$ arbitraris $C4$ mitjançant la funció `GrayMapImage(C4)`. Ens assegurem que aquesta no generarà un codi binari lineal, testant amb la funció `HasLinearGrayMapImage(C4)`. Per un sèrie d'ordres menors que *order*, i després de comprovar que el vector tot zeros pertany al codi binari C, calculem els temps de creació dels dos nuclis amb la repetició *rep* perquè aquest càlcul no és sempre el mateix ja que depèn de la CPU de la màquina a més CPU's més ràpides, més repeticions). Ensenyem els temps calculats i els acumulem per treure al final un total. Aquests càlculs també es guarden en el fitxer "TempsKernelZ2BigNoLinear<order>-<rep>.txt".

```
timesKernelZ2BigNoLinear:=procedure(order,rep)
    FileName:="timesKernelZ2BigNoLinear" cat IntegerToString(order) cat "-" cat IntegerToString(rep) cat ".txt";
    T1:=0; T2:=0;
    SetLogFile(FileName);
    printf "timesKernelZ2BigNoLinear:   order=%o  rep=%o\n",order,rep;
    printf "%5o%5o%5o%10o%10o%10o\n","n","#ker","k","times_old","times_new","decrement";
    Z4:=IntegerRing(4); n4max:=order div 2;
      for n4:=1 to n4max do
        for k4:=1 to n4 do
            C4:=RandomLinearCode(Z4,n4,k4);
            if (not HasLinearGrayMapImage(C4)) then
                C:=GrayMapImage(C4);
                n:=Degree(C[1]);
                if n le order then
                    V:=VectorSpace(GF(2),n); ZeroVector:=V!0;
                    if (ZeroVector in C) then
                        t:=Cputime();
                        for i:=1 to rep do
                            kernelori:=KernelZ2Ori(C);
                        end for;
                        t1:=Cputime(t); T1:=T1+t1;
                        t:=Cputime();
                        for i:=1 to rep do
                            kernel:=KernelZ2(C);
                        end for;
                        t2:=Cputime(t); T2:=T2+t2;
                        num:=#kernel; k:=Dimension(kernel);
                        printf "%5o%5o%5o%10o%10o%10o\n",n,#kernel,Dimension(kernel),t1,t2,t1-t2;
                    end if;
                end if;
            end if;
        end for;
    end for;
    //print "\nTotal"; T1; T2; T1-T2;
```

```
    printf "%15o%10o%10o%10o\n","Total  ",T1,T2,T1-T2;
    UnsetLogFile();
end procedure;
```

Si executem aquest procediment fins l'ordre 18 i sense repetició, obtenim aquesta taula :

```
TempsKernelZ2BigNoLinear:   order=18  rep=1
    n #ker    k times_old times_new decrement
    8    8    3     0.000     0.000     0.000
   10   16    4     0.000     0.000     0.000
   10   64    6     0.120     0.010     0.110
   12    8    3     0.010     0.000     0.010
   12   32    5     0.070     0.020     0.050
   12  256    8     7.080     0.870     6.210
   14    8    3     0.000     0.000     0.000
   14   64    6     0.120     0.020     0.100
   14  128    7     4.600     1.580     3.020
   14  256    8     6.270     0.410     5.860
   16    4    2     0.000     0.000     0.000
   16   16    4     0.000     0.000     0.000
   16   32    5     0.060     0.010     0.050
   16   32    5     0.860     0.230     0.630
   16  128    7    45.570     4.620    40.950
   16  256    8  1485.260   196.130  1289.130
   18    4    2     0.000     0.000     0.000
   18    8    3     0.000     0.000     0.000
   18   16    4     0.070     0.060     0.010
   18   32    5     0.940     0.330     0.610
   18  128    7    46.610     6.160    40.450
   18  256    8  1587.830   309.280  1278.550
```

En ella, es mostra clarament que la nova versió de la funció `KernelZ2(C)` és més ràpida. En general quant més gran és la longitud del codi binari no lineal, més és l'avantatge (decrement en micro-segons de CPU) que en treu en la construcció dels nuclis. I fixada una longitud de codi binari no lineal, quant més gran és la dimensió del nucli a construir, és a dir quants més elements té el nucli, més gran és també el decrement.

Ho hem provat expressament amb els codis no lineals, perquè amb els codis lineals no paga la pena provar-ho, segur que és més ràpida. Doncs en les primeres línies de codi de la nova funció, quan detecta que la llista de vectors binaris que formen el codi C té estructura d'espai vectorial, retornen ràpidament el subespai vectorial generat de la llista com a nucli. No perd el temps aplicant la definició de nucli, i, per tant, fent una doble iteració exhaustiva provant tots els vectors del codi, sumant cada vector del codi amb tots els vectors del codi comprovant si el resultat és un vector també del codi ($c + C = C$).

Capítol 6

Resultats

Un cop fet el disseny, millora i desenvolupament del *package* a que es feia referència en els punts 6, 7, 8 i 9 dels objectius detallats del capítol 3. En aquest capítol veurem, analitzarem i obtindrem resultats, que corresponen als punts 10, 11 i 12 dels següents objectius detallats de l'esmentat capítol.

És per això que aquest capítol de resultats està dedicat a dos aspectes:

- Analitzar les invariants 4-profile, rang i dimensió del nucli i SHDE en la classificació de les matrius Hadamard.
- Ampliar la base de dades de matrius Hadamard com a conseqüència de la generalització de la funció que construeix matrius Hadamard de qualsevol ordre n per cada rang r i dimensió del nucli k.

6.1 Invariants: 4-profile, rang i nucli i SHDE

El procediment `HadamardFourInvariantsTable(n:file,lr)` compara les quatre invariants. Fixat l'ordre n, per cada invariant calcula el nombre de matrius Hadamard inequivalents d'aquest ordre que reconeix i el temps que triga en segons. Les característiques d'aquest procediment són:

- Calcula els temps de les invariants de totes les matrius d'un ordre n, així com el nombre de matrius inequivalents que reconeixen.

- Guarda els resultats en el fitxer `file` que per defecte s'anomena `FIT<n>`.
- Permet escollir la base de dades amb la que volem treballar mitjançant el `library root` *lr* que per defecte és el de la base de dades Hadamard estàndard del MAGMA.
- Si utilitzem la base de dades estàndard del MAGMA, sabem que totes les seves matrius de cadascun dels ordres són inequivalents. Per tant, en aquest cas, el número de matrius de cada ordre és un bon indicador de la sensibilitat de les invariants per classificar les matrius, doncs és la fita que les invariants han d'assolir.
- Per la invariant SHDE comencem per $k = 4$ i fem un tractament incremental. Mentre no haguem reconegut com inequivalents el mateix nombre de matrius de l'ordre tractat, o bé no haguem sobrepassat el límit teòric de la k que és $n/2$; anem incrementant la k de dos en dos i tornem a calcular la invariant SHDE per totes les matrius de l'ordre.

```
/*****************************************************************************************
HadamardFourInvariantsTable: RgnIntEl : MonStgElt : MonStgElt
Given a positive integer as the order, tabulates in a file the number of HadamardDatabase
matrices of the order from the Library Root recognized as inequivalent for each of the
four invariants (4-profile, rank and kernel, and Fang&Ge's SHDEDistribution with k-value).
*****************************************************************************************/

HadamardFourInvariantsTable:=procedure(n : file:="", lr:="")

    if (lr eq "") then
        D:=HadamardDatabase();
    else
        stdlibroot:=GetLibraryRoot();
        SetLibraryRoot(lr);
        D:=HadamardDatabase();
        SetLibraryRoot(stdlibroot);
    end if;
    if (file eq "") then file:="FIT" cat IntegerToString(n); end if;

    NOM:=NumberOfMatrices(D,n);
    printf "Checkpoint-FourInvariantsTable n=%o: start\n",n;
    fprintf file,"FourInvariantsTable  n=%o\n",IntegerToString(n);
    fprintf file,"  n  Mat 4-p r-k F&G <-k         t(4-p)         t(r-k)         t(F&G)\n";
    // 4-profile, rank-kernel and Fang-Ge(a=pi,k=4,6,...,n/2) invariants
```

```
    L4p:=[];Lrk:=[];LFG:=[]; t4p:=0;trk:=0;tFG:=0; a:=3.1415926; k:=4;
    if (NOM eq 1) then
        fprintf file,"%4o%4o%4o%4o%4o%4o%15o%15o%15o\n",n,NOM,1,1,1,k,t4p,trk,tFG;
    else
        for m:=1 to NOM do
            H:=Matrix(D,n,m); C:=HadamardMatrixToCode(H);
            t:=Cputime(); L4p:=L4p cat [HadamardInvariant(H)];       t4p:=t4p+Cputime(t);
            t:=Cputime(); Lrk:=Lrk cat [InvariantsRankKernelZ2(C)];  trk:=trk+Cputime(t);
            t:=Cputime(); LFG:=LFG cat [HadamardInvariantSHDE(H:a:=a,k:=k)];tFG:=tFG+Cputime(t);
            printf "Checkpoint-FourInvariantsTable n=%o: m=%o/%o k=%o\n",n,m,NOM,k;
        end for;
            neFG:=#Set(LFG);
            fprintf file,"%4o%4o%4o%4o%4o%4o%15o%15o%15o\n"
                    ,n,NOM,#Set(L4p),#Set(Lrk),neFG,k,t4p,trk,tFG;

            while ((neFG lt NOM) and (k lt (n div 2))) do
                LFG:=[]; k:=k+2; tFG:=0;
                for m:=1 to NOM do
                    H:=Matrix(D,n,m);
                    t:=Cputime(); LFG:=LFG cat [HadamardInvariantSHDE(H:a:=a,k:=k)];
                    tFG:=tFG+Cputime(t);
                    printf "Checkpoint-FourInvariantsTable n=%o: m=%o/%o k=%o\n",n,m,NOM,k;
                end for;
                neFG:=#Set(LFG);
                fprintf file,"%20o%4o%45o\n",neFG,k,tFG;
            end while;

        end if;
printf "Checkpoint-FourInvariantsTable n=%o: end",n;
end procedure;
```

Hem agrupat en la taula que es mostra a continuació, els resultats de tots els fitxers guardats pel procediment quan s'executa per alguns ordres de la base de dades estàndard del MAGMA en quasi tots els casos, doncs també hi ha una entrada a la taula calculada a partir d'unes matrius Hadamard d'ordre 32 proporcionades amablement per Ilias Kotsireas i Cristos Koukouvinos.

FourInvariantsTable

n	Mat	4-p	r-k	SHDE	<-k	t(4-p)	t(r-k)	t(SHDE)	
1	1	1	1	1	4	0	0	0	
2	1	1	1	1	4	0	0	0	
4	1	1	1	1	4	0	0	0	
8	1	1	1	1	4	0	0	0	
12	1	1	1	1	4	0	0	0	
16	5	4	5	4	4	0.000	0.020	5.440	
16				4	6			31.750	
16				5	8			62.960	
20	3	1	1	1	4	0.000	0.000	13.340	
20				3	6			140.750	
24	60	35	2	35	4	0.020	0.230	870.810	
24				60	6			14400.230	
28	487	60	1	60	4	0.190	3.090	18452.360	
28				487	6			443711.970	
32	23	18	13	18	4	0.010	0.190	1963.410	
32				22	6			67527.000	
32				23	8			948840.400	el job triga 11 dies!
KE32	10	8	7	8	4	0.000	0.030	854.860	10ineq32 I.Kotsireas i C.Koukouvinos
KE32				10	6			27702.790	
48	55	53	2	53	4	0.190	0.970	56787.600	

Conclusions que es dedueixen al examinar la taula `FourInvariantsTable`:

- La invariant 4-profile té la mateixa sensibilitat que la invariant SHDE per $a := 3.1415926, k := 4$ perquè les dues invariants calculen la mateixa distribució de freqüències, però la invariant SHDE tarda més temps perquè és més complexa de calcular. Per ser més precisos direm que, a diferència de la 4-profile, la invariant SHDE no reflexa la freqüència nul·la en la seva distribució de freqüències. Mirem per exemple aquesta sessió de MAGMA:

```
> H:=Matrix(HadamardDatabase(),16,1);
> HadamardInvariant(H);
[ 1680, 0, 140 ]
> HadamardInvariantSHDE(H : a:=3.1415926,k:=4);
[ 1680, 140 ]
```

- La invariant SHDE, incrementant la k, acaba sent sempre la més sensible; però també és la més costosa en temps de CPU.

- Nogensmenys hi ha l'excepció del cas d'ordre $n = 16$, la invariant rank i dimensió de nucli reconeix ràpidament les 5 matrius inequivalents d'aquest ordre, i mentre que la invariant 4-profile és queda en 4, la invariant SHDE ha d'arribar fins a la $k = 8$ per reconèixer-les totes, trigant, considerant només el càlcul de $k = 8$, de l'ordre de 10^6 vegades més!

- El cas d'order $n = 24$ és el que ens ha servit per comprovar la fiabilitat de la implementació de la invariant SHDE, al coincidir amb les taules de classificació d'aquest ordre que es donaven en els articles introductoris d'aquesta nova invariant [5, 4]. Recordem que per $k = 4$ ja en reconeix 35 matrius Hadamard inequivalents i per $k = 6$ totes les 60.

- El cas d'ordre $n = 32$ és aquell en que la combinació de les invariants 4-profile i rang i dimensió de nucli, amb `HadamardThreeInvariants`, reconeix una matriu Hadamard inequivalent més que la invariant SHDE per $k = 4$ (per més detalls veure el projecte [11]). Per identificar les 23 matrius d'aquest ordre que té la base de dades de matrius Hadamard estàndard del MAGMA va trigar 11 dies, i ho va assolir per un valor de $k = 8$.

- De les 10 matrius de Hadamard inequivalents d'ordre $n = 32$ proporcionades per Ilias Kotsireas i Cristos Koukouvinos [6], 8 es van reconèixer per la 4-profile i la SHDE per $k = 4$ com era d'esperar; la SHDE va reconèixer totes 10 només a la $k = 6$ següent.

Com exemple d'execució d'aquest procediment per una base de dades de matrius Hadamard que no sigui la estàndard, vàrem seleccionar 10 matrius inequivalents d'ordre $n = 32$ d'un grapat de fitxers de matrius Hadamard de diferents ordres que ens van proporcionar amablement Ilias Kotsireas i Cristos Koukouvinos. Aquestes 10 matrius Hadamard gregues són inequivalents entre elles, però només la meitat són inequivalents amb les de la base de dades estàndard (mala sort perquè hi han més de 66.000 matrius de Hadamard

inequivalents d'ordre $n = 32$), perquè al actualitzar la base de dades només admetia 5, com es pot veure en aquesta sessió de MAGMA.

```
> D:=HadamardDatabase();
> #D;
3491
> data:=HadamardDatabaseInformation(D);
> SetVerbose("HadamardDB",1);
> load "matrixfile32ineq10.m";
Loading "matrixfile32ineq10.m"
> UpdateHadamardDatabase(~data,S);
5 new matrices added
```

El fitxer `matrixfile32ineq10.m` és el mateix que vaig descarregar de la web de Ilias Kotsireas i Cristos Koukouvinos [6] amb les 10 matrius inequivalents d'ordre 32 en format llista d'enters carregat a la variable `order32ineq10`, però modificat per tal que ompli la variable `S` amb la llista de les matrius. Per això es va comentar el codi MAGMA originari que ve després de la llista i es va executar aquest:

```
S:=[HadamardMatrixFromInteger(i,32):i in order32ineq10];
```

```
order32ineq10 := \\
[ 17878192447832605036863095254629242828791987994639169972663784266536475666220\
3559490076587870970124834831691553683854621428370404477668582028504074677511871\
0770331954532028040269267648909035039053784571041656416840172014675825983024170\
95397846399881240357823602327971292259087509761535980580813975597832149375,
17878194599220980653495013283554546256777365596142976227928580102201951432252291\
7942857701844993992435857290977887509777289461752280178840469277486273948022130\
4467159200626323985228638354914516559843604701846891797084589286690617941766134\
0286950650851248559988394478038093007133848043203802741798814201926844 15,
17878200986416819546452838340910653163248065004322531581114148134733100209753 51\
4160183218665616038348793046059553361372020647834441142784896376036894700351838\
5722403477142550530629115152039158500336465694878289282989227822945169787661126\
2124849802691797397894047982735665459758403708261102595072109561212501 75,
17865024410680208239906204514698644841110176159428912972796654884620120951690 33\
7640749946615517914168662958440720864851438194097010433434508267687574445528189\
4314853941016339852385894651614934051253340659674235368276943956497028915812428\
1262102196957852022874246934496956048879245531401746708299338075308744 95,
17865027985165952941469746391337514927082461831439046953807626019345009903861 56\
3592458011838252483324467317553707506698482662484440945674354782858171099879504\
0753466605885626594084453111021792371295136632832260303217304045570515740348561\
1968706440201498182249875003274482834459300720915777847086332298783023 35,
17865032949264422749495947600980055175566253169112274581247018752816745495223 49\
5851925506494036940033972835344905840761316004891003809361201793317001394692920\
```

```
300406196495608635608824491275058864621134925038688296724897031185960040105645 8\
940848800083724584253227163751969891776255162430304349523169165819975295,
178595350720849539493018021907756833421996374064402946457689455481773203962278 4\
962985914478060897338438736498452908970267796809723355661421856954796923836886 1\
796690108430540703662123058957992751958785071794942031996705986883483485911031 9\
144683384799966809503738543996552943898087498245039280469321951574174335,
178595386214572156670136169638965833243656927801778386334794210876823543628014 3\
676724831065851668200783907414101837530552132463433855498157668169936042185331 7\
493758813490550131651098802066431733745331104890711313028520161359655131059953 7\
969854427019777349994473518982593605232895882347941647103396254818708095,
178782031378051951630847563698359565912334426058263378363789439703985759757860 7\
615403326172351301830116716788207248568716727254627355476716280311570089718686 1\
315636722456607048893098861605017115480161329255953939605661564205318942701016 4\
843333645473064237964641918106083587601715413610509952873116778481785215,
178650333259666675072719582433449086124686026806967182330274770010221672432387 2\
048299180283348899127568478623373507752898856821083268619064660216377476565677 9\
165154454304820612235379555991712959208894068304959892304665429303219541495573 4\
225713769501826360597856960035605032672773348269155536649341301958317695 ];

/***************************************************************************
n := #order32ineq10;
for i in [1..n-1] do
 for j in [i+1..n] do
  print i,j,IsHadamardEquivalent(HadamardMatrixFromInteger(order32ineq10[i],32)\\
               ,HadamardMatrixFromInteger(order32ineq10[j],32));
 end for;
end for;
***************************************************************************/
//To code the "matrixfile.m" that generates the list S of these Hadamard matrices when loaded:
// 1st. Delete, if exists, the '^M' characters behind '\' in the integer list.
// 2nd. Comment the magma code after the integer list and execute:
S:=[HadamardMatrixFromInteger(i,32):i in order32ineq10];
```

El procediment `HadamardDB32ineq10` crea la nova base de dades a partir de les matrius Hadamard d'una llista `S` que li ha preparat el programa anterior.

```
//************   HadamardDB32ineq10   ************************************
// new database creation with the "32ineq10" Cristos Koukouvinos's Hadamard matrices
// "matrixfile32ineq10.m" generates the list S of these Hadamard matrices when loaded
// previously we must create the directory "data" in our default directory

data:=HadamardDatabaseInformationEmpty();
SetVerbose("HadamardDB",1);
load "matrixfile32ineq10.m";
```

```
UpdateHadamardDatabase(~data,S);
WriteHadamardDatabase("~/data/hadamard",~data);
```

I a partir d'aquí ja es pot executar el procediment `HadamardFourInvariantsTable(n)` per la base de dades de 10 matrius Hadamard inequivalents d'ordre 32 de Ilias Kotsireas i Cristos Koukouvinos.

6.2 Construcció de matrius amb rang i nucli

En el projecte [10] s'implementà amb les funcions `HadamardAllRankKernel(n)` i `HadamardRankKernelNPower2(n,r,k)` un algorisme recurrent complet per a la construcció de matrius Hadamard de qualsevol ordre $n = 2^t$. Per cada ordre n es construeix tantes matrius, inequivalents, com rang r i dimensió del nucli k possibles i compatibles amb l'ordre. Aquest algorisme es troba en la pàgina 17 de [10] i es basa en les construccions descrites en l'article [8].

En aquest projecte, la nova `HadamardRankKernelNNotPower2(n,r,k)` completa la generalització de la funció principal `HadamardAllRankKernel(n)` perquè li atorga un nou algorisme recurrent que s'afegeix a l'anterior, i que li permet ara, a més a més, la construcció de matrius Hadamard de qualsevol ordre $n = 2^t \cdot s$ sempre que disposem del cas $n = 4s$ a la base de dades de matrius Hadamard del MAGMA . De la mateixa manera per cada ordre n es construeix tantes matrius, inequivalents, com rang r i dimensió del nucli k possibles i compatibles amb l'ordre. Aquest algorisme es troba en la subsecció 2.6.4, capítol 2, i es basa en les construccions descrites en l'article [9].

La construcció de noves matrius de Hadamard surt doncs de la explotació de l'algorisme recurrent per la construcció de matrius de Hadamard d'ordre $n = 2^t \cdot s$. L'ordre $n = 252$ $(252 = 4 \cdot 63)$ és el cas $n = 4s$ més gran que es troba actualment a la base de dades de matrius Hadamard del MAGMA. Per tant, a partir de l'actual base de dades, podem construir infinites matrius Hadamard d'ordre $\{n = 2^t \cdot s, t \geq 2, s \in \{3, 5, ..., 63\}\}$. Anem a veure, per alguns ordres, quantes d'aquestes matrius Hadamard, inequivalents, ja hi són

a la base de dades i quantes són noves.

En resum, podem incrementar la base de dades utilitzant només el rang i la dimensió del nucli, per el cas de les matrius d'ordre no potència de dos, sempre que el seu graó primigeni $n = 4s$ on $s \neq 1$ i senar estigui a la base de dades Hadamard i els recursos de càlcul del sistema siguin suficients.

El procediment `HadamardRankKernelNNotPower2Table(nmax:file)` construeix moltes matrius de Hadamard que no estan en la base de dades estàndard d'ordre $n = 2^t \cdot s$, amb la funció `HadamardRankKernelNNotPower2(n,r,k)`. Les característiques principals d'aquest procediment són:

- Busca les matrius de Hadamard per tots els orders que no són potències de dos, fins arribar a l'ordre màxim $nmax$.

- Filtra aquells ordres que el seu cas $n = 4s$ no estigui a la base de dades estàndard del MAGMA.

- Per cada ordre, comptabilitza les matrius "inicials" que són les que ja tenim a la base de dades, les "actuals" que són les que tenim més les que construeix, i la diferència entre les dues que és el "increment".

- Al nombre de matrius construïdes per rang i dimensió de nucli se li descompte el nombre de matrius "inicials" inequivalents per la invariant rang i dimensió de nucli, abans de sumar-les a les "actuals". Per exemple, en el cas $n = 72$, d'entrada tenim 105 matrius inicials a la base de dades que són inequivalents, però totes tenen el mateix rang i dimensió de nucli, per tant només hi ha 1 de inequivalent per aquest invariant. Mitjançant la funció `HadamardAllRankKernel(n)` per rang i dimensió de nucli en construïm 2, de les quals 1 ja la tenim a les "inicials". Per això el nombre de matrius "actuals" és $105 + (2 - 1) = 106$ i, per tant, el increment és $106 - 105 = 1$.

- Guarda la taula d'aquests acumuladors per cada ordre en el fitxer `file`, que per defecte s'anomena `HadRankKerNNotPow2Tab<nmax>`.

```
/*****************************************************************************************
HadamardRankKernelNNotPower2Table: RgnIntElt [,MonStgElt]
Given a positive integer n as the maximum order, tabulates in a file the new Hadamard n=s.2^(t)
matrices with diferent rang and kernel, that they are not in the Hadamard Database.
*****************************************************************************************/

HadamardRankKernelNNotPower2Table:=procedure(nmax:file:="HadRankKerNNotPow2Tab" cat IntegerToString(nmax))
D:=HadamardDatabase(); Ln:=Degrees(D);
printf "Checkpoint-HadamardRankKernelNNotPower2Table(%o): start\n",nmax;
// Number of Matrices calculation
T:=[];TInici:=0;TAct:=0;TInc:=0;
n:=12;
while (n le nmax) do
    t,s:=Valuation(n,2);
    if (s ne 1) then
      if (NumberOfMatrices(D,4*s) ge 1) then
        // Hadamard matrices in the HadamardDatabase
        if (n in Ln) then NOM:=NumberOfMatrices(D,n); else NOM:=0; end if;
        Inici:=NOM; TInici:=TInici+Inici;
        // Hadamard matrices by Rank-Kernel construction not in HadamardDatabase
        Lrk:=[];
        for m:=1 to NOM do
            Lrk:=Lrk cat [InvariantsRankKernelZ2(HadamardMatrixToCode(Matrix(D,n,m)))];
        end for;
        Act:=#HadamardAllRankKernel(n);
        if (Act gt #Set(Lrk)) then
            Act:=NOM+(Act-#Set(Lrk));
        end if;
        TAct:=TAct+Act; Inc:=Act-Inici; TInc:=TInc+Inc;
        Append(~T,[n,Inici,Act,Inc]);
        printf "Checkpoint-HadamardRankKernelNNotPower2Table(%o): n=%o Inici=%o Act=%o Inc=%o\n"
                ,nmax,n,Inici,Act,Inc;
      end if;
    end if;
    n:=n+4;
end while;
printf "Checkpoint-HadamardRankKernelNNotPower2Table(%o): Total TInici=%o TAct=%oTInc=%o\n"
        ,nmax,TInici,TAct,TInc;
printf "Checkpoint-HadamardRankKernelNNotPower2Table(%o):   end\n",nmax;
Sort(~T);
fprintf file,"HadamardRankKernelNNotPower2Table hasta %o\n",IntegerToString(nmax);
fprintf file,"    n     Inici      Act       Inc\n";
for i in [1..#T] do
    fprintf file,"%5o%10o%10o%10o\n",T[i][1],T[i][2],T[i][3],T[i][4];
end for;
fprintf file,"Total%10o%10o%10o\n",TInici,TAct,TInc;
end procedure;
```

La Taula 6.1 resumeix la sortida del procediment executat fins l'ordre 720, ja que només apareixen els ordres que mostren un increment del nombre de matrius.

Taula 6.1: Construcció matrius de Hadamard fins $n = 720$.

n	Inici	Act	Inc	Tinc	n	Inici	Act	Inc	n	Inici	Act	Inc
72	105	106	1	1	280	0	2	2	456	0	2	2
80	1	42	41	42	288	0	253	253	464	0	234	234
96	1	85	84	126	296	0	2	2	472	0	2	2
104	1	2	1	127	304	0	154	154	480	0	421	421
112	2	59	57	184	312	0	2	2	488	0	2	2
120	3	4	1	185	320	0	359	359	496	0	250	250
136	2	3	1	186	328	0	2	2	504	0	2	2
152	1	2	1	187	336	0	170	170	528	0	266	266
160	1	141	140	327	344	0	2	2	544	0	477	477
168	1	2	1	328	352	0	309	309	560	0	282	282
176	2	91	89	417	360	0	2	2	576	0	647	647
184	1	2	1	418	368	0	186	186	592	0	298	298
192	1	215	214	632	376	0	2	2	608	0	533	533
200	1	2	1	633	384	0	488	488	624	0	314	314
208	1	106	105	738	392	0	2	2	640	0	816	816
216	1	2	1	739	400	0	202	202	656	0	330	330
224	1	197	196	935	408	0	2	2	672	0	589	589
232	1	2	1	936	416	0	365	365	688	0	346	346
240	1	122	121	1057	424	0	2	2	704	0	791	791
248	1	2	1	**1058**	432	0	218	218	720	0	362	362
264	0	2	2		440	0	2	2				
272	0	138	138		448	0	503	503	**Tot.**	**129**	**11520**	**11391**

Al examinar la Taula 6.1 podem extreure els següents resultats:

- Donat que la base de dades de matrius Hadamard del MAGMA arriba actualment fins l'ordre $n = 256$, en la taula a partir de l'ordre $n = 264$ cap endavant totes les matrius construïdes per rang i dimensió del nucli són increment net.

- Per contra, fixem-nos en l'ordre $n = 80$ (és el tercer ordre de la seqüència del $4s = 20, 40, 80, ...$) del qual la base de dades de matrius Hada-

mard del MAGMA disposa de 1 matriu, que naturalment té un rang i dimensió de nucli concret. Les matrius construïdes d'ordre $n = 80$ per rang i dimensió del nucli són 42, per tant hem aconseguit 41 noves matrius Hadamard que juntament amb la 1 de la base de dades fan un total de 42 matrius Hadamard inequivalents per l'ordre 80.

- Si fem el mateix plantejament per tots els ordres de la taula que ja tenen matrius a la base de dades de matrius Hadamard del MAGMA, obtenim 1.058 noves matrius Hadamard inequivalents que es podrien incorporar a la actual base de dades.

- Tan sols amb aquesta execució hem aconseguit un total de 11.391 noves matrius de Hadamard.

Capítol 7

Conclusions

7.1 Conclusions

En aquest projecte hem assolit els següents objectius:

1. Hem estudiat les matrius i codis Hadamard.
2. Hem estudiat les invariants 4-profile, rang i dimensió del nucli.
3. Hem estudiat la proposta d'una nova invariant més sensible a la inequivalència de matrius Hadamard proposada pels xinesos Kai-Tai Fang i Gennian Ge.
4. Hem estudiar els treballs dels dos projectistes anteriors.
5. Hem après a utilitzar el MAGMA.
6. Hem après a llençar processos de grans càlculs seguint un protocol d'us.
7. Hem optimitzat la funció `KernelZ2(C)` aprofitant la seva estructura d'espai vectorial.
8. Hem implementat una nova invariant per a la detecció de matrius Hadamard no equivalents: la distribució de l'enumerador de la distància simètrica de Hamming, SHDE.

9. Hem implementat una funció que construeix matrius Hadamard per a qualsevol ordre $n = 2^t \cdot s$ on $s \neq 1$ i senar, per cada rang i una dimensió de nucli vàlides.

10. Hem implementat procediments per executar tests de fiabilitat i rendiment.

11. Hem actualitzat el manual Hadamard d'usuari amb les noves funcions i les modificades.

12. Hem actualitzat els exemples d'ús de les funcions.

13. Hem actualitzat la llibreria.

14. Hem actualitzat el package.

15. Hem analitzat les invariants 4-profile, rang i dimensió del nucli, i SHDE.

16. Hem ampliat la base de dades del MAGMA construint noves matrius Hadamard d'ordre $n = 2^t \cdot s$ utilitzant les funcions implementades i tenint en compte el rang i la dimensió del nucli, fins el $4s$ més gran que es troba actualment a la base de dades Hadamard del MAGMA.

17. I finalment, hem redactat la memòria.

Les futures línies de continuació per a aquest projecte podrien ser:

- Optimitzar encara més la funció `KernelZ2(C)` tenint en compte les propietats de codi lineal del nucli: la idea és alhora que es construeix el nucli, anar buidant del codi a examinar els cosets del nucli per reduir el camp de la iteració exhaustiva.

- Fer que la funció que construeix les matrius Hadamard per qualsevol ordre $n = 2^t \cdot s$ i per un rang i una dimensió de nucli pugui buscar la d'ordre $4s$, que és el primer graó de la cadena, en qualsevol `library root`, sense està restringit a la base de dades Hadamard estàndard del MAGMA.

- Implementar altres construccions conegudes com la construcció de Paley, la construcció de Williamson o els arrays de Baumert-Hall [7].

Capítol 8

Handbook of MAGMA Functions

8.1 Introduction

A *Hadamard matrix* is an $n \times n$ matrix of $+1$'s and -1's such that every pair of rows and every pair of columns differ in exactly $n/2$ places. Two such matrices are considered equivalent if one can be transformed into the other by performing row swaps, column swaps, row negations or column negations. The problem of deciding whether two Hadamard matrices are equivalent is hard.

A *binary Hadamard matrix* is a $n \times n$ Hadamard matrix, where the $+1$'s are replaced by 0's and the -1's by 1's. A *Hadamard code* is a binary $(n, 2n, n/2)$-code consisting of the rows of a binary Hadamard matrix and their complements.

Two structural properties of non-linear codes are the rank and kernel. The *rank* of a binary code C, $r = rank(C)$, is simply the dimension of the linear span, $\langle C \rangle$, of C. The *kernel* of a binary code C is defined as $K(C) = \{x \in \mathbb{F}^n \mid x + C = C\}$, where $\mathbb{F} = \{0, 1\}$. If the zero word is in C, then $K(C)$ is a linear subspace of C. We will denote the dimension of the kernel of C by $k = ker(C)$. These parameters can be used to distinguish between non-equivalent Hadamard matrices, since equivalent ones have the same parameters r and k.

MAGMA contains several functions for Hadamard matrices. In section 8.2, new functions to work with them and the corresponding binary Hadamard matrices and Hadamard codes are given. In section 8.3, new functions to compute the invariants rank, r, and dimension of the kernel, k, for any non-linear code are presented, specifically they can be used to increase the number of invariants computed for Hadamard matrices in MAGMA. Finally, in section 8.4, functions to construct new Hadamard matrices from existing Hadamard matrices or codes, as well as from given invariants r and k, are described.

8.2 Hadamard matrices and codes converting

`HadamardMatrixToBinary(H)`

Given a Hadamard matrix H, returns the corresponding Hadamard binary matrix. This function is the inverse of `HadamardBinaryToMatrix()`.

`HadamardBinaryToMatrix(H)`

Given a binary Hadamard matrix H, returns the corresponding Hadamard matrix. This function is the inverse of `HadamardMatrixToBinary()`.

`HadamardMatrixToCode(H)`

Given a Hadamard matrix H, returns the corresponding Hadamard code. The code is represented as a list of binary vectors of length n. This function is the inverse of `HadamardCodeToMatrix()`.

`HadamardCodeToMatrix(C)`

Given a Hadamard code represented as a list of binary vectors of length n, returns the corresponding normalized Hadamard matrix of degree n. This function is the inverse of `HadamardMatrixToCode()`.

`IsHadamardCode(C)`

Returns `true` if and only if C is a Hadamard code.

Example E1

The following example converts a Hadamard matrix to a binary Hadamard matrix and to a Hadamard code.

```
> H:=Matrix([[1,1,1,1],[1,-1,1,-1],[1,1,-1,-1],[1,-1,-1,1]]);
> IsHadamard(H);
true
> Hb:=HadamardMatrixToBinary(H);
> Hb;
[0 0 0 0]
[0 1 0 1]
[0 0 1 1]
[0 1 1 0]
> H eq HadamardBinaryToMatrix(Hb);
true
> C:=HadamardMatrixToCode(H);
> C;
[
    (0 0 0 0),
    (1 1 1 1),
    (0 1 0 1),
    (1 0 1 0),
    (0 0 1 1),
    (1 1 0 0),
    (0 1 1 0),
    (1 0 0 1)
```

```
]
> IsHadamardCode(C);
true
> H eq HadamardCodeToMatrix(C);
true
```

8.3 Invariants of (Hadamard) codes

`RankZ2(C)`

Given a code C represented as a list of binary vectors of length n, returns its rank. The rank of a code C is the dimension of the linear span of C, $\langle C \rangle$, over $GF(2)$.

`KernelZ2(C)`

Given a code C represented as a list of binary vectors of length n and such that the zero vector belongs to C, returns its kernel as a VectorSubspace. Then the kernel of C are the codewords v such that $v + C = C$.

`DimensionOfKernelZ2(C)`

Given a code C represented as a list of binary vectors of length n, returns the dimension of its kernel. The code C must contain the zero vector to assure that its kernel is a linear subspace of C over $GF(2)$.

`InvariantsRankKernelZ2(C)`

Given a code C represented as a list of binary vectors of length n, returns its rank and dimension of the kernel.

`HadamardThreeInvariants(H)`

Given a Hadamard matrix H, returns the invariants 4-profile, rank and dimension of the kernel.

`ExistsHadamardRankKernel(n,r,k)`

Given positive integers n, r and k, returns `true` if there exists a Hadamard code (or equivalently a Hadamard matrix) of length n with rank r and kernel of dimension k. When n is not a power of two, returns also `false` if we do not know whether or not there exists a Hadamard code with these parameters.

`HadamardInvariantSHDE(H,a,k)`

Given a Hadamard matrix H, a ponderator positive number a and a positive integer k, returns only the frequencies of the Symmetric Hamming Distance Enumerator distribution of H over all k-dimensional columns projections. Invariant proposed by Kai-Tai Fang and Gennian Ge. It defaults a to 3.1415926 and k to 4.

`HadamardInequivalentMatricesSHDE(H1,H2 :a)`

Given two Hadamard matrices $H1$, $H2$ and a ponderator positive integer a, returns true if they are inequivalent and false if is not possible to prove their inequivalence by this method. It defaults a to 3.1415926.

`HadamardClassificationSHDE(n:a,k,lr,file)`

Given an order n, a ponderator positive number a, a positive integer k, a library root lr and a filename $file$, returns and saves in the file the list of the values of the integer part of the Symmetric Hamming Distance Enumerator over all k-dimensional columns projections and their frequencies of all the Hadamard matrices of order n from the hadamard database of the library root. The classification is perfect when the number of elements of the list equals number of Hadamard matrices. It defaults a to 3.1415926, k to 4, lr to the standard Magma HadamardDatabase and $file$ to HadClaSHDE<n>.

Example E2

In this example the rank, the kernel and the dimension of the kernel of a Hadamard code are computed. Kai-Tai Fang and Gennian Ge's SHDEDistribution invariant and inequivalent matrices algorism of Hadamard matrices are computed too. All using Hadamard matrices from a HadamardDatabase[1].

```
> D:=HadamardDatabase();
> H:=Matrix(D,12,1);
> H;
[ 1  1  1  1  1  1  1  1  1  1  1  1]
[ 1  1  1  1  1  1 -1 -1 -1 -1 -1 -1]
[ 1  1  1 -1 -1 -1  1  1  1 -1 -1 -1]
[ 1  1 -1  1 -1 -1  1 -1 -1  1  1 -1]
[ 1  1 -1 -1  1 -1 -1  1 -1  1 -1  1]
[ 1  1 -1 -1 -1  1 -1 -1  1 -1  1  1]
[ 1 -1 -1 -1  1  1  1  1 -1 -1  1 -1]
[ 1 -1  1  1 -1 -1 -1  1 -1 -1  1  1]
[ 1 -1  1 -1 -1  1  1 -1 -1  1 -1  1]
[ 1 -1 -1  1 -1  1 -1  1  1  1 -1 -1]
[ 1 -1 -1  1  1 -1  1 -1  1 -1 -1  1]
```

[1] For more information consult chapter 115 from [2].

```
[ 1 -1  1 -1  1 -1 -1 -1  1  1  1 -1]
> C:=HadamardMatrixToCode(H);
> RankZ2(C);
11
> KernelZ2(C);
Vector space of degree 12, dimension 1 over GF(2)
Generators:
(1 1 1 1 1 1 1 1 1 1 1 1)
Echelonized basis:
(1 1 1 1 1 1 1 1 1 1 1 1)
> DimensionOfKernelZ2(C);
1
> InvariantsRankKernelZ2(C);
[ 11, 1 ]
> HadamardThreeInvariants(H);
[ 495, 0, 11, 1 ]
> Inv1:=HadamardInvariant(Matrix(D,32,3));
> Inv2:=HadamardInvariant(Matrix(D,32,14));
> Inv1 eq Inv2;
true
> Inv3:=HadamardThreeInvariants(Matrix(D,32,3));
> Inv4:=HadamardThreeInvariants(Matrix(D,32,14));
> Inv3 eq Inv4;
false
> ExistsHadamardRankKernel(12,11,1);
true
> ExistsHadamardRankKernel(12,12,1);
false
> ExistsHadamardRankKernel(32,6,6);
true
> ExistsHadamardRankKernel(32,6,7);
false
> Inv5:=HadamardThreeInvariants(Matrix(D,20,2));
> Inv6:=HadamardThreeInvariants(Matrix(D,20,3));
> Inv5 eq Inv6;
true
> Inv7:=HadamardInvariantSHDE(Matrix(D,20,2):k:=6);
> Inv8:=HadamardInvariantSHDE(Matrix(D,20,3):k:=6);
```

```
> Inv7 eq Inv8;
false
> HadamardInequivalentMatricesSHDE(Matrix(D,20,2),Matrix(D,20,3));
true
> HadamardInequivalentMatricesSHDE(Matrix(D,20,2),Matrix(D,20,2));
false
```

8.4 Construction of (Hadamard) matrices

`HadamardKroneckerSylvester(t)`

Given a positive integer t, returns the Hadamard-Sylvester matrix of degree $n = 2^t$.

`HadamardKronecker(H1,H2)`

Given two Hadamard matrices $H1$ and $H2$ of degree n, returns a Hadamard matrix of degree $2n$. This matrix is constructed with the Kronecker product $S \otimes [H1, H2]$, where S is the Hadamard-Sylvester matrix of degree 2.

`HadamardKroneckerPermutation(H1,H2,g)`

Given two Hadamard matrices $H1$ and $H2$ of degree n and a permutation g in `Sym(n)`, returns a Hadamard matrix of degree $2n$. This matrix is constructed with the Kronecker product $S \otimes [H1, g(H2)]$, where S is the Hadamard-Sylvester matrix of degree 2 and $g(H2)$ is $H2$ with the columns permuted by g.

`SwitchCode(C,S,x)`

Given a code C represented as a list of binary vectors, a subset S of C and a codeword x, returns a code where the codewords of S are substituted by the binary vectors of $S + x$.

`HadamardAllRankKernel(n : swint)`

Given a positive interger n, returns a list of Hadamard matrices of degree n. There is one Hadamard matrix with rank r and kernel of dimension k, for each possible pair (r, k). Optionally be returned to integer conversion if boolean parameter *swint* is set to true, it defaults to false.

`HadamardAllRankKernelFile(n : swint)`

Given a positive interger n, saves in a file called "HadamardMatrix<n>"Hadamard matrices of degree n. There is one Hadamard matrix with rank r and kernel of dimension k, for each possible pair (r, k). Optionally be returned to integer conversion if boolean parameter *swint* is set to true, it defaults to false.

`HadamardRankKernel(n,r,k : swint)`

Given positive integers n, r and k, returns a Hadamard matrix of degree n such that its corresponding Hadamard code of length n has rank r and kernel of dimension k. Optionally to integer conversion if boolean parameter *swint* is set to *true*, it defaults to $false$.

`HadamardRandomMatrix(n : swint)`

Given positive interger n, returns a random Hadamard matrix of degree n. Optionally to integer conversion if boolean parameter *swint* is set to *true*, it defaults to $false$.

Example E3

Here there are some examples that show how to construct Hadamard matrices using the Kronecker product and the Switching tecnique.

```
> H:=HadamardKroneckerSylvester(2);
> H;
[ 1  1  1  1]
[ 1 -1  1 -1]
[ 1  1 -1 -1]
[ 1 -1 -1  1]
> HadamardKronecker(H,H);
[ 1  1  1  1  1  1  1  1]
[ 1 -1  1 -1  1 -1  1 -1]
[ 1  1 -1 -1  1  1 -1 -1]
[ 1 -1 -1  1  1 -1 -1  1]
[ 1  1  1  1 -1 -1 -1 -1]
[ 1 -1  1 -1 -1  1 -1  1]
[ 1  1 -1 -1 -1 -1  1  1]
[ 1 -1 -1  1 -1  1  1 -1]
> g:=Random(Sym(4));
> g;
(1, 3, 2)
> HadamardKroneckerPermutation(H,H,g);
[ 1  1  1  1  1  1  1  1]
[ 1 -1  1 -1  1 -1  1 -1]
[ 1  1 -1 -1  1  1 -1 -1]
[ 1 -1 -1  1  1 -1 -1  1]
[ 1  1  1  1 -1 -1 -1 -1]
[-1  1  1 -1  1 -1 -1  1]
[ 1 -1  1 -1 -1  1 -1  1]
[-1 -1  1  1  1  1 -1 -1]
```

```
> V:=VectorSpace(GF(2),48);
> C:=HadamardMatrixToCode(HadamardRankKernel(48,13,3));
> w:=C[5];
> a:=[1:j in [1..12]];
> b:=[0:j in [1..36]];
> K:=KernelZ2(C);
> S:=[k+w : k in K];
> x:=V!Vector(a cat b);
> IsHadamardCode(SwitchCode(C,S,x));
true
```

Example E4

This example shows how construct new Hadamard matrices given only the degree n, all o one randomly; or given the degree n, rank r and kernel k, both to integer conversion optionally.

```
> H:=HadamardAllRankKernel(16);
> H;
[
    [ 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1]
    [ 1 -1  1 -1  1 -1  1 -1  1 -1  1 -1  1 -1  1 -1]
    [ 1  1 -1 -1  1  1 -1 -1  1  1 -1 -1  1  1 -1 -1]
    [ 1 -1 -1  1  1 -1 -1  1  1 -1 -1  1  1 -1 -1  1]
    [ 1  1  1  1 -1 -1 -1 -1  1  1  1  1 -1 -1 -1 -1]
    [ 1 -1  1 -1 -1  1 -1  1  1 -1  1 -1 -1  1 -1  1]
    [ 1  1 -1 -1 -1 -1  1  1  1  1 -1 -1 -1 -1  1  1]
    [ 1 -1 -1  1 -1  1  1 -1  1 -1 -1  1 -1  1  1 -1]
    [ 1  1  1  1  1  1  1  1 -1 -1 -1 -1 -1 -1 -1 -1]
    [ 1 -1  1 -1  1 -1  1 -1 -1  1 -1  1 -1  1 -1  1]
    [ 1  1 -1 -1  1  1 -1 -1 -1 -1  1  1 -1 -1  1  1]
    [ 1 -1 -1  1  1 -1 -1  1 -1  1  1 -1 -1  1  1 -1]
    [ 1  1  1  1 -1 -1 -1 -1 -1 -1 -1 -1  1  1  1  1]
    [ 1 -1  1 -1 -1  1 -1  1 -1  1 -1  1  1 -1  1 -1]
    [ 1  1 -1 -1 -1 -1  1  1 -1 -1  1  1  1  1 -1 -1]
    [ 1 -1 -1  1 -1  1  1 -1 -1  1  1 -1  1 -1 -1  1],

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

```
[ 1 -1  1 -1  1 -1  1 -1  1 -1  1 -1  1 -1  1 -1]
[ 1  1 -1 -1  1  1 -1 -1  1  1 -1 -1  1  1 -1 -1]
[ 1 -1 -1  1  1 -1 -1  1  1 -1 -1  1  1 -1 -1  1]
[ 1  1  1  1 -1 -1 -1 -1  1  1  1  1 -1 -1 -1 -1]
[ 1 -1  1 -1 -1  1 -1  1  1 -1  1 -1 -1  1 -1  1]
[ 1  1 -1 -1 -1 -1  1  1  1  1 -1 -1 -1 -1  1  1]
[ 1 -1 -1  1 -1  1  1 -1  1 -1 -1  1 -1  1  1 -1]
[ 1  1  1  1  1  1  1  1 -1 -1 -1 -1 -1 -1 -1 -1]
[ 1  1 -1 -1  1 -1  1 -1 -1 -1  1  1 -1  1 -1  1]
[ 1 -1  1 -1  1  1 -1 -1 -1  1 -1  1 -1 -1  1  1]
[ 1 -1 -1  1  1 -1 -1  1 -1  1  1 -1 -1  1  1 -1]
[ 1  1  1  1 -1 -1 -1 -1 -1 -1 -1 -1  1  1  1  1]
[ 1  1 -1 -1 -1  1 -1  1 -1 -1  1  1  1 -1  1 -1]
[ 1 -1  1 -1 -1 -1  1  1 -1  1 -1  1  1  1 -1 -1]
[ 1 -1 -1  1 -1  1  1 -1 -1  1  1 -1  1 -1 -1  1],

[ 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1]
[ 1 -1  1 -1  1 -1  1 -1  1 -1  1 -1  1 -1  1 -1]
[ 1  1 -1 -1  1  1 -1 -1  1  1 -1 -1  1  1 -1 -1]
[ 1 -1 -1  1  1 -1 -1  1  1 -1 -1  1  1 -1 -1  1]
[ 1  1  1  1 -1 -1 -1 -1  1  1  1  1 -1 -1 -1 -1]
[ 1 -1  1 -1 -1  1 -1  1  1 -1  1 -1 -1  1 -1  1]
[ 1  1 -1 -1 -1 -1  1  1  1  1 -1 -1 -1 -1  1  1]
[ 1 -1 -1  1 -1  1  1 -1  1 -1 -1  1 -1  1  1 -1]
[ 1  1  1  1  1  1  1  1 -1 -1 -1 -1 -1 -1 -1 -1]
[ 1 -1 -1  1 -1  1 -1  1 -1  1  1 -1 -1  1 -1  1]
[ 1 -1  1 -1  1  1 -1 -1 -1  1 -1  1 -1 -1  1  1]
[ 1  1 -1 -1  1 -1 -1  1 -1  1 -1  1  1 -1  1 -1]
[ 1  1  1  1 -1 -1 -1 -1 -1 -1 -1 -1  1  1  1  1]
[ 1 -1 -1  1 -1  1 -1  1 -1  1  1 -1  1 -1  1 -1]
[ 1 -1  1 -1 -1 -1  1  1 -1  1 -1  1  1  1 -1 -1]
[ 1  1 -1 -1 -1  1  1 -1 -1 -1  1  1  1 -1 -1  1],

[ 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1]
[ 1 -1  1 -1  1 -1  1 -1  1  1  1  1 -1 -1 -1 -1]
[ 1  1 -1 -1  1  1 -1 -1  1 -1  1 -1  1 -1  1 -1]
[ 1 -1 -1  1  1 -1 -1  1  1  1 -1 -1  1  1 -1 -1]
[ 1  1  1  1 -1 -1 -1 -1  1 -1 -1  1  1 -1 -1  1]
```

```
    [ 1 -1  1 -1 -1  1 -1  1  1 -1  1 -1 -1  1 -1  1]
    [ 1  1 -1 -1 -1 -1  1  1  1  1 -1 -1 -1 -1  1  1]
    [ 1 -1 -1  1 -1  1  1 -1  1 -1 -1  1 -1  1  1 -1]
    [ 1  1  1  1  1  1  1  1 -1 -1 -1 -1 -1 -1 -1 -1]
    [ 1 -1  1 -1  1 -1  1 -1 -1 -1 -1 -1  1  1  1  1]
    [ 1  1 -1 -1  1  1 -1 -1 -1  1 -1  1 -1  1 -1  1]
    [ 1 -1 -1  1  1 -1 -1  1 -1 -1  1  1 -1 -1  1  1]
    [ 1  1  1  1 -1 -1 -1 -1 -1  1  1 -1 -1  1  1 -1]
    [ 1 -1  1 -1 -1  1 -1  1 -1  1 -1  1  1 -1  1 -1]
    [ 1  1 -1 -1 -1 -1  1  1 -1 -1  1  1  1  1 -1 -1]
    [ 1 -1 -1  1 -1  1  1 -1 -1  1  1 -1  1 -1 -1  1],

    [ 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1]
    [ 1 -1  1 -1  1 -1  1 -1  1 -1  1 -1  1 -1  1 -1]
    [ 1  1 -1 -1  1  1 -1 -1  1  1 -1 -1  1  1 -1 -1]
    [ 1 -1 -1  1  1 -1 -1  1  1 -1 -1  1  1 -1 -1  1]
    [ 1  1  1  1 -1 -1 -1 -1  1  1  1  1 -1 -1 -1 -1]
    [ 1 -1  1 -1 -1  1 -1  1  1 -1  1 -1 -1  1 -1  1]
    [ 1  1 -1 -1 -1 -1  1  1  1  1 -1 -1 -1 -1  1  1]
    [ 1 -1 -1  1 -1  1  1 -1  1 -1 -1  1 -1  1  1 -1]
    [ 1  1  1  1  1  1  1  1 -1 -1 -1 -1 -1 -1 -1 -1]
    [ 1  1 -1  1 -1 -1  1 -1 -1 -1  1 -1  1  1 -1  1]
    [ 1  1  1 -1 -1  1 -1 -1 -1 -1 -1  1  1 -1  1  1]
    [ 1  1 -1 -1  1 -1 -1  1 -1 -1  1  1 -1  1  1 -1]
    [ 1 -1  1  1  1 -1 -1 -1 -1  1 -1 -1 -1  1  1  1]
    [ 1 -1 -1  1 -1  1 -1  1 -1  1  1 -1  1 -1  1 -1]
    [ 1 -1  1 -1 -1 -1  1  1 -1  1 -1  1  1  1 -1 -1]
    [ 1 -1 -1 -1  1  1  1 -1 -1  1  1  1 -1 -1 -1  1]
]
> Hi:=HadamardAllRankKernel(16:swint:=true);
> Hi;
[ 4775922363650773020980155034437111907893478331888744128983400465623127700275 2,
4775923738587072862556939259952851654687960235052287943164297794665310335795 2,
4505596137738920609433576384845494644806497347539438022759537482095158113075 2,
4775922363650795196475063808284663522708403532919374800558368408248720306995 2,
5136360539864806056020132603891703351165515952683931008213471087391391626035 2 ]
> for i in [1..#H]do
> InvariantsRankKernelZ2(HadamardMatrixToCode(H[i]));
```

```
> end for;
[ 5, 5 ]
[ 6, 3 ]
[ 7, 2 ]
[ 8, 2 ]
[ 8, 1 ]
> for i in [1..#Hi]do
> InvariantsRankKernelZ2(HadamardMatrixToCode(HadamardMatrixFromInteger(Hi[i],16)));
> end for;
[ 5, 5 ]
[ 6, 3 ]
[ 7, 2 ]
[ 8, 2 ]
[ 8, 1 ]
> HadamardRankKernel(16,8,1);
[ 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1]
[ 1 -1  1 -1  1 -1  1 -1  1 -1  1 -1  1 -1  1 -1]
[ 1  1 -1 -1  1  1 -1 -1  1  1 -1 -1  1  1 -1 -1]
[ 1 -1 -1  1  1 -1 -1  1  1 -1 -1  1  1 -1 -1  1]
[ 1  1  1  1 -1 -1 -1 -1  1  1  1  1 -1 -1 -1 -1]
[ 1 -1  1 -1 -1  1 -1  1  1 -1  1 -1 -1  1 -1  1]
[ 1  1 -1 -1 -1 -1  1  1  1  1 -1 -1 -1 -1  1  1]
[ 1 -1 -1  1 -1  1  1 -1  1 -1 -1  1 -1  1  1 -1]
[ 1  1  1  1  1  1  1  1 -1 -1 -1 -1 -1 -1 -1 -1]
[ 1  1 -1  1 -1 -1  1 -1 -1 -1  1 -1  1  1 -1  1]
[ 1  1  1 -1 -1  1 -1 -1 -1 -1 -1  1  1 -1  1  1]
[ 1  1 -1 -1  1 -1 -1  1 -1 -1 -1  1  1 -1  1  1 -1]
[ 1 -1  1  1  1 -1 -1 -1 -1  1 -1 -1 -1  1  1  1]
[ 1 -1 -1  1 -1  1 -1  1 -1  1  1 -1  1 -1  1 -1]
[ 1 -1  1 -1 -1 -1  1  1 -1  1 -1  1  1  1 -1 -1]
[ 1 -1 -1 -1  1  1  1 -1 -1  1  1  1 -1 -1 -1  1]
> HadamardRankKernel(16,8,1:swint:=true);
513636053986480605602013260389170335116551595268393100821347108739139162603 52
> HadamardRankKernel(16,8,1) eq\\
    HadamardMatrixFromInteger(HadamardRankKernel(16,8,1:swint:=true),16);
true
> HadamardRankKernel(20,19,1);
[ 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1]
```

```
[ 1  1  1  1  1  1  1  1  1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  1]
[ 1  1  1  1 -1 -1 -1 -1 -1  1  1  1  1  1 -1 -1 -1 -1 -1  1]
[ 1  1  1 -1  1 -1  1 -1 -1  1 -1 -1 -1  1  1 -1  1 -1  1 -1]
[ 1  1  1 -1 -1 -1 -1  1  1 -1  1 -1  1 -1  1  1  1 -1 -1 -1]
[ 1  1 -1  1 -1  1 -1  1 -1  1  1 -1 -1 -1 -1 -1  1  1  1 -1]
[ 1 -1  1  1  1 -1  1 -1 -1 -1  1  1 -1 -1 -1  1  1  1 -1 -1]
[ 1  1 -1  1  1  1 -1 -1 -1 -1 -1 -1  1  1  1  1 -1  1 -1 -1]
[ 1 -1  1  1 -1  1 -1 -1  1  1 -1  1 -1 -1  1  1 -1 -1  1 -1]
[ 1 -1 -1 -1  1  1 -1 -1  1  1  1 -1 -1  1 -1  1  1 -1 -1  1]
[ 1  1 -1 -1 -1 -1  1  1  1  1 -1  1 -1  1 -1  1 -1  1 -1 -1]
[ 1 -1 -1  1  1 -1 -1  1  1 -1 -1  1  1  1 -1 -1  1 -1  1 -1]
[ 1 -1  1 -1 -1  1  1 -1  1 -1  1 -1  1  1 -1 -1 -1  1  1 -1]
[ 1  1 -1 -1  1 -1 -1 -1  1 -1  1  1 -1 -1  1 -1 -1  1  1  1]
[ 1 -1 -1  1 -1 -1  1 -1  1  1 -1 -1  1 -1  1 -1  1  1 -1  1]
[ 1 -1  1 -1 -1  1 -1  1 -1 -1 -1  1 -1  1  1 -1  1  1 -1  1]
[ 1 -1  1 -1  1 -1 -1  1 -1  1 -1 -1  1 -1 -1  1 -1  1  1  1]
[ 1  1 -1 -1 -1  1  1 -1 -1 -1 -1  1  1 -1 -1  1  1 -1  1  1]
[ 1 -1 -1  1 -1 -1  1  1 -1 -1  1 -1 -1  1  1  1 -1 -1  1  1]
[ 1 -1 -1 -1  1  1  1  1 -1  1  1  1  1 -1  1 -1 -1 -1 -1 -1]
> HadamardRankKernel(20,19,1:swint:=true);
252239378352637154703509321900297875468821399039650803620528128562545460812 36/\\
     80370995571103980967787230602651441655971840
> HadamardRankKernel(20,19,1) eq\\
      HadamardMatrixFromInteger(HadamardRankKernel(20,19,1:swint:=true),20);
true
> HadamardRandomMatrix(16);
[ 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1]
[ 1  1  1  1  1  1  1  1 -1 -1 -1 -1 -1 -1 -1 -1]
[ 1  1  1  1 -1 -1 -1 -1  1  1  1  1 -1 -1 -1 -1]
[ 1  1  1  1 -1 -1 -1 -1 -1 -1 -1 -1  1  1  1  1]
[ 1  1 -1 -1  1  1 -1 -1  1  1 -1 -1  1  1 -1 -1]
[ 1  1 -1 -1  1  1 -1 -1 -1 -1  1  1 -1 -1  1  1]
[ 1  1 -1 -1 -1 -1  1  1  1  1 -1 -1 -1 -1  1  1]
[ 1  1 -1 -1 -1 -1  1  1 -1 -1  1  1  1  1 -1 -1]
[ 1 -1  1 -1  1 -1  1 -1  1 -1  1 -1  1 -1  1 -1]
[ 1 -1  1 -1  1 -1 -1  1 -1  1 -1  1 -1  1  1 -1]
[ 1 -1  1 -1 -1  1  1 -1 -1  1  1 -1 -1  1 -1  1]
[ 1 -1  1 -1 -1  1 -1  1  1 -1 -1  1  1 -1 -1  1]
```

```
[ 1 -1 -1  1  1 -1  1 -1  1 -1 -1  1 -1  1 -1  1]
[ 1 -1 -1  1  1 -1 -1  1 -1  1  1 -1  1 -1 -1  1]
[ 1 -1 -1  1 -1  1  1 -1 -1  1 -1  1  1 -1  1 -1]
[ 1 -1 -1  1 -1  1 -1  1  1 -1  1 -1 -1  1  1 -1]
> HadamardRandomMatrix(20:swint:=true);
252239378352637154703509321900297875468821399039650803620528128562545460812 36/\\
     80370995571103980967787230602651441655971840
```

8.5 Hadamard Database

`CCGHadamardDatabase()`

Returns the database of Hadamard matrices.

Bibliografia

[1] E. F. Assmus Jr. i J. D. Key, *Designs and their codes*, Cambridge University Press, 1992.

[2] W. Bosma, J. J. Cannon, *Handbook of Magma Functions*, Sydney: School of Mathematics and Statistics, University of Sydney (1995) <http://magma.maths.usyd.edu.au>.

[3] H. Evangelaras, C. Koukouvinos i J. Seberry, *Applications of Hadamard matrices*, Journal of Telecommunications and Information Technology (2003).

[4] K. T. Fang i Gennian Ge, *An efficient algorithm for the classification of Hadamard matrices*, Technical Report MATH-298, Hong Kong Baptist University, 2001.

[5] K. T. Fang i Gennian Ge, *A sensitive algorithm for detecting the inequivalence of Hadamard matrices*, Mathematics of computation, Vol. 73, No. 46 (2003) pp. 843-851.

[6] I. Kotsireas, C. Koukouvinos, *Hadamard ideals and inequivalent Hadamard matrices from two circulant submatrices*, updated March (2006) <http://www.cargo.wlu.ca/circulantSubmatrices/>.

[7] F. J. MacWilliams i N. J. Sloane, *The theory of Error-Correcting codes*, North-Holland, New York (1977).

[8] K. T. Phelps, J. Rifà i M. Villanueva, *Rank and Kernel of binary Hadamard codes*, IEEE Transactions on Information Theory, Vol. 51, No. 11 (2005) pp. 3931-3937.

[9] K. T. Phelps, J. Rifà i M. Villanueva, *Hadamard codes of length* $2^t s$ *(s odd). Rank and Kernel*, Lectures Notes in Computer Science. 3857 (2006), 328-337.

[10] M. Pujol, *Ranq i nucli de matrius Hadamard* $n = 2^t$ *en MAGMA*, Bellaterra (2006).

[11] L. Rodríguez, *Invariants de matrius Hadamard de mida* $n = 2^t s$ *en MAGMA*, Bellaterra (2006).

[12] E. Tressler, *A survey of the Hadamard Conjecture*, Blacksburg, Virginia (2004).

Firmat: Francesc Díez Aquilué

Bellaterra, 19 de Setembre de 2007

Resum

L'objectiu d'aquest projecte ha estat generalitzar e integrar la funcionalitat de dos projectes anteriors que ampliaven el tractament que oferia el MAGMA respecte a les matrius de Hadamard. Hem implementat funcions genèriques que permeten construir noves matrius Hadamard de qualsevol mida per cada rang i dimensió de nucli, i així ampliar la seva base de dades. També hem optimitzat la funció que calcula el nucli, i hem desenvolupat funcions que calculen la invariant *Symmetric Hamming Distance Enumerator* (SHDE) proposada per Kai-Tai Fang i Gennian Ge i que és més sensible per a la detecció de la no equivalència de les matrius Hadamard.

Resumen

El objetivo de este proyecto ha sido generalizar e integrar la funcionalidad de dos proyectos anteriores que ampliaban el tratamiento que ofrecía el MAGMA respecto a las matrices Hadamard. Hemos implementado funciones genéricas que permiten construir nuevas matrices Hadamard de cualquier orden para cada rango y dimensión de núcleo, y así ampliar su base de datos. También hemos optimizado la función que calcula el núcleo, y hemos desarrollado funciones que calculan el invariante *Symmetric Hamming Distance Enumerator* (SHDE) propuesta por Kai-Tai Fang y Gennian Ge que es más sensible en la detección de la no equivalencia de las matrices Hadamard.

Abstract

The aim of this project has been to generalize and to integrate the former two projects' functionality which extended MAGMA's treatement in relation to Hadamard matrices. We have implemented generic functions that allow us to construct new Hadamard matrices of any order for each possible pair of rank and dimension of kernel, and thus to extend its database. We have also optimized the function that computes the kernel, and we have developed functions that compute the Symmetric Hamming Distance Enumerator invariant proposed by Kai-Tai Fang and Gennian Ge, which is more sensitive for detecting the inequivalence of Hadamard matrices.