Bachelor en Mathématiques

Mathématiques expérimentales

Semestre d'été 2021

La face cachée des matrices d'Hadamard et leurs applications

Auteurs :
Léa MICARD
Eva RAGAZZINI
Lara SUYS

Superviseur :
Guendalina PALMIROTTA

Table des matières

Introduction 3

1 Matrices d'Hadamard 4
- 1.1 Propriétés 4
- 1.2 Conjecture 5

2 Constructions 6
- 2.1 Produit de Kronecker 6
- 2.2 Construction de Sylvester 7
  - 2.2.1 Algorithme de construction 7
- 2.3 Construction de Paley 10
  - 2.3.1 Algorithme de construction 11
- 2.4 Visualisation des matrices 14

3 Matrices de Walsh 17

4 Correction d'erreurs 20
- 4.1 Théorie 20
- 4.2 Algorithme de correction 21

Conclusion 25

References 27

Introduction

Depuis leur découverte au 19e siècle par le mathématicien français Jacques Hadamard, les matrices d'Hadamard sont au coeur des recherches des mathématiciens. Pourtant il reste encore beaucoup de secrets à découvrir. Les matrices d'Hadamard sont utilisées dans de nombreux domaines tels que les codes correcteurs, les réalisations de plans d'analyse sensorielle et de plans d'expérience factoriels. Nous allons nous intéresser aux codes de correction d'erreurs.

En effet, lorsqu'on se demande comment les images reçues depuis l'espace arrivent sur Terre sans présenter d'erreurs de communications, on se rend compte que cela est possible grâce à ces codes-ci. Par exemple, en 1990, la NASA (*National Aeronautics and Space Administration*) a utilisé pour la mission Mariner 9 le code de Reed-Muller, en utilisant entre autres des matrices d'Hadamard. On peut en savoir plus sur l'histoire de la mission et du code dans [1].

Avant de nous plonger dans l'aventure, nous allons introduire les matrices d'Hadamard ainsi que leurs propriétés importantes dans la Section 1.

Ensuite dans la Section 2, nous allons présenter deux approches connues, Sylvester et Paley, pour construire et ensuite visualiser les matrices d'Hadamard.

Dans la Section 3, nous allons discuter brièvement des matrices de Walsh et de leur construction, qui nous aiderons par la suite.

Finalement dans la dernière Section 4, nous allons établir un code de correction d'erreur pour "décrypter" des messages.

On terminera ce projet par la section Conclusion où nous proposerons des idées et expliquerons les difficultés rencontrées.

1 Matrices d'Hadamard

Nous allons découvrir dans cette section les matrices d'Hadamard et leurs propriétés importantes.

Définition 1. Une matrice carrée $\mathcal{H}_n$ est dite d'Hadamard d'ordre $n \in \mathbb{N}$, si tous ses coefficients sont 1 et -1 et qu'elle satisfait aux deux propriétés suivantes :

(i) $\mathcal{H}_n \cdot \mathcal{H}_n^T = \mathcal{H}_n^T \cdot \mathcal{H}_n = n \cdot I$, où I est la matrice identité de taille n et $\mathcal{H}_n^T$ la transposée de $\mathcal{H}_n$.

(ii) Les lignes ainsi que les colonnes sont orthogonales entre elles.

Exemple 2. Les matrices d'Hadamard d'ordre 2, respectivement 4 :

- $\mathcal{H}_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$,
- $\mathcal{H}_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}$.

Par le calcul on voit directement que $\mathcal{H}_4$ (de même pour $\mathcal{H}_2$) respecte les propriétés (i) et (ii) de la Déf 1. En effet, en multipliant $\mathcal{H}_4$ par sa transposée $\mathcal{H}_4^T$, on obtient :

$$\underbrace{\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}}_{\mathcal{H}_4} \cdot \underbrace{\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}}_{\mathcal{H}_4^T} = \begin{pmatrix} 4 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 4 \end{pmatrix} = 4 \cdot I.$$

Pour l'orthogonalité, nous le vérifions simplement avec un produit scalaire en prenant les lignes et les colonnes deux-à-deux et vérifiant que le résultat est nul. Par exemple l'orthogonalité entre la première et la deuxième colonne est :

$$\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & -1 & 1 & -1 \end{pmatrix} = 1 \cdot 1 + 1 \cdot (-1) + 1 \cdot 1 + 1 \cdot (-1) = 0.$$

On a donc bien un produit scalaire nul, signifiant que les colonnes sont orthogonales. On répète le calcul de la même manière aux colonnes restantes.

1.1 Propriétés

Pour introduire les propriétés des matrices d'Hadamard, nous avons d'abord besoin d'introduire quelques notions, dont l'inégalité d'Hadamard.

Soit $a_i \in \mathbb{N}, \forall i \in \{1, ..., n\}$ et $X = \begin{pmatrix} a_1 & \cdots & a_n \end{pmatrix}^T$. On note par $\|X\|_2 = \left(\sum |a_i|^2\right)^{\frac{1}{2}}$ la norme Euclidienne dans $\mathbb{R}^2$.

Définition 3. Soit M une matrice carrée dont les vecteurs colonnes sont $X_1, ..., X_n$.
L'inégalité d'Hadamard est définie par :

$$|\det(M)| \leq \|X_1\|_2 \cdots \|X_n\|_2 \left(= \prod_{i=1}^{n} \|X_i\|_2 \right),$$

où $\det(\cdot)$ est le déterminant de M.

Théorème 4. On sait par [2] que si M est une matrice d'ordre n, dont les coefficients sont 1 et -1, alors son déterminant est majoré par $n^{n/2}$,

$$\det(M) \leq n^{n/2}.$$

Donc en remplaçant M par la matrice d'Hadamard on obtient le résultat suivant.

Proposition 5. [2]. Une matrice réelle M d'ordre n, dont tous les éléments $u_{ij} \in \mathbb{R}$ sont bornés tels que $|M_{ij}| \leq 1, \forall i, j \in \mathbb{N}$ est dite d'Hadamard si et seulement si elle atteint l'égalité d'Hadamard, c'est-à-dire

$$|\det(M)| = n^{n/2}.$$

Proposition 6. La transposée d'une matrice d'Hadamard est également une matrice d'Hadamard.

Démonstration. Soit $\mathcal{H}_n$ une matrice d'Hadamard d'ordre n et on note $\mathcal{H}_n^T$ sa transposée .
On sait par l'algèbre linéaire que le déterminant d'une matrice est le même que celui de sa transposée. Donc par Prop 5, le résultat suit directement. □

1.2 Conjecture

Quand on parle des matrices d'Hadamard, on peut se demander quel est l'ordre maximal des matrices que l'on peut construire. Rappelons que l'ordre d'une matrice est la taille d'une matrice carrée. Jusqu'à maintenant on sait que l'ordre d'une matrice d'Hadamard est nécessairement $1, 2$ ou un multiple de 4. On peut retrouver la preuve dans [3].
Une des conjectures principales, attribuée à Raymond Paley, est la suivante :

"Pour tout n multiple de 4, il existe une matrice d'Hadamard d'ordre n."

A l'heure actuelle, le plus petit ordre multiple de 4 pour lequel aucune matrice d'Hadamard est connue est de 668. Auparavant, on ne connaissait pas la matrice de taille 764, mais elle a été trouvée en 2008 par le mathématicien Dragomir DOKOVI , [4].
Nous pouvons nous renseigner sur d'autres conjectures intéressantes dans [5].

2 Constructions

Pour construire des matrices d'Hadamard, on dispose de plusieurs outils. Nous allons en expliquer quelques uns, ainsi que l'algorithme qui va avec. Nous commencerons par expliquer et utiliser la construction de Sylvester, une construction par itération utilisant le produit de Kronecker. Nous expliquerons ensuite la construction de Paley, une construction plus moderne reposant sur l'utilisation des corps finis.

2.1 Produit de Kronecker

Nous commençons par introduire la notion de produit de Kronecker $\otimes$ dont nous aurons besoin pour expliquer la construction de Sylvester.

Définition 7. On définit le produit de Kronecker $\otimes$ pour des matrices de taille 2 comme suivant, $\forall\, a,b,c,d,e,f,g,h \in \mathbb{R}$, on a :

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}_{2\times 2} \otimes \begin{pmatrix} e & f \\ g & h \end{pmatrix}_{2\times 2} = \begin{pmatrix} a.\begin{pmatrix} e & f \\ g & h \end{pmatrix} & b.\begin{pmatrix} e & f \\ g & h \end{pmatrix} \\ c.\begin{pmatrix} e & f \\ g & h \end{pmatrix} & d.\begin{pmatrix} e & f \\ g & h \end{pmatrix} \end{pmatrix} = \begin{pmatrix} a\,e & a\,f & b\,e & b\,f \\ a\,g & a\,h & b\,g & b\,h \\ c\,e & c\,f & d\,e & d\,f \\ c\,g & c\,h & d\,g & d\,h \end{pmatrix}_{4\times 4}.$$

On peut ensuite généraliser cette définition à des matrices de taille quelconque.
Soient A et B deux matrices de taille $m \times n$, respectivement $p \times q$, avec $m,n,p,q \in \mathbb{R}$, non-nécessairement distingués. Par le produit de Kronecker, on obtient une matrice de taille $mp \times nq, \forall\, a_{11},\ldots,a_{mn},b_{11},\ldots,b_{pq} \in \mathbb{R}$, on a :

$$\begin{aligned} A \otimes B &= \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}_{m\times n} \otimes \begin{pmatrix} b_{11} & \cdots & b_{1q} \\ \vdots & \ddots & \vdots \\ b_{p1} & \cdots & b_{pq} \end{pmatrix}_{p\times q} \\ &= \begin{pmatrix} a_{11}.\begin{pmatrix} b_{11} & \cdots & b_{1q} \\ \vdots & \ddots & \vdots \\ b_{p1} & \cdots & b_{pq} \end{pmatrix} & \cdots & a_{1n}.\begin{pmatrix} b_{11} & \cdots & b_{1q} \\ \vdots & \ddots & \vdots \\ b_{p1} & \cdots & b_{pq} \end{pmatrix} \\ \vdots & \ddots & \vdots \\ a_{m1}.\begin{pmatrix} b_{11} & \cdots & b_{1q} \\ \vdots & \ddots & \vdots \\ b_{p1} & \cdots & b_{pq} \end{pmatrix} & \cdots & a_{mn}.\begin{pmatrix} b_{11} & \cdots & b_{1q} \\ \vdots & \ddots & \vdots \\ b_{p1} & \cdots & b_{pq} \end{pmatrix} \end{pmatrix} \\ &= \begin{pmatrix} a_{11}\,b_{11} & \cdots & a_{11}\,b_{1q} & \cdots & a_{1n}\,b_{11} & \cdots & a_{1n}\,b_{1q} \\ a_{11}\,b_{p1} & \cdots & a_{11}\,b_{pq} & \cdots & a_{1n}\,b_{p1} & \cdots & a_{1n}\,b_{pq} \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{m1}\,b_{p1} & \cdots & a_{m1}\,b_{pq} & \cdots & a_{mn}\,b_{p1} & \cdots & a_{mn}\,b_{pq} \\ a_{m1}\,b_{p1} & \cdots & a_{m1}\,b_{pq} & \cdots & a_{mn}\,b_{p1} & \cdots & a_{mn}\,b_{pq} \end{pmatrix}_{mp\times nq}. \end{aligned}$$

Exemple 8. Soient $A = \begin{pmatrix} 1 & 2 \\ 4 & 5 \end{pmatrix}_{2\times 2}$ et $B = \begin{pmatrix} 5 & 7 \\ 3 & 1 \\ 2 & 6 \end{pmatrix}_{3\times 2}$. Par le produit de Kronecker, on obtient une matrice de taille 6×4 :

$$\begin{pmatrix} 1 & 2 \\ 4 & 5 \end{pmatrix} \otimes \begin{pmatrix} 5 & 7 \\ 3 & 1 \\ 2 & 6 \end{pmatrix} = \begin{pmatrix} 1. \begin{pmatrix} 5 & 7 \\ 3 & 1 \\ 2 & 6 \end{pmatrix} & 2. \begin{pmatrix} 5 & 7 \\ 3 & 1 \\ 2 & 6 \end{pmatrix} \\ 4. \begin{pmatrix} 5 & 7 \\ 3 & 1 \\ 2 & 6 \end{pmatrix} & 5. \begin{pmatrix} 5 & 7 \\ 3 & 1 \\ 2 & 6 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 5 & 7 & 10 & 14 \\ 3 & 1 & 6 & 2 \\ 2 & 6 & 4 & 12 \\ 20 & 28 & 25 & 35 \\ 12 & 4 & 15 & 5 \\ 8 & 24 & 10 & 30 \end{pmatrix}.$$

2.2 Construction de Sylvester

Le mathématicien James Joseph Sylvester est à l'origine des premiers exemples de matrices d'Hadamard. Sa méthode repose sur une construction itérative en partant des matrices d'ordre 1 et 2. Cette construction est basée sur la propriété suivante :

Proposition 9. [6]. Si $\mathcal{H}_n$ est une matrice d'Hadamard d'ordre n, alors $\begin{pmatrix} \mathcal{H}_n & \mathcal{H}_n \\ \mathcal{H}_n & -\mathcal{H}_n \end{pmatrix}$ est une matrice d'Hadamard d'ordre $2n$.

Par itération, on peut construire n'importe quelle matrice de taille 2^k, $k \in \mathbb{N}$, à l'aide du produit de Kronecker grâce à la formule suivante :

$$\mathcal{H}_{2^k} = \begin{pmatrix} \mathcal{H}_{2^{k-1}} & \mathcal{H}_{2^{k-1}} \\ \mathcal{H}_{2^{k-1}} & -\mathcal{H}_{2^{k-1}} \end{pmatrix} = \mathcal{H}_2 \otimes \mathcal{H}_{2^{k-1}}.$$

On observe que les matrices obtenues sont symétriques et de trace nulle. En effet, quand on regarde la matrice par bloc, on remarque que $\begin{pmatrix} \mathcal{H}_n & \mathcal{H}_n \\ \mathcal{H}_n & -\mathcal{H}_n \end{pmatrix} = \begin{pmatrix} \mathcal{H}_n & \mathcal{H}_n \\ \mathcal{H}_n & -\mathcal{H}_n \end{pmatrix}^T$, ce qui implique la symétrie. Quant à la trace, en regardant à nouveau les blocs, on remarque sur la diagonale que les valeurs du premier et deuxième bloc sont opposées, ce qui engendrera une trace nulle.

2.2.1 Algorithme de construction

Nous allons maintenant nous intéresser à l'algorithme de construction des matrices. Il s'agit d'un algorithme déjà existant en Python avec le module SciPy, nous allons donc expliquer son code source, provenant de [7]. Nous l'avons traduit et avons enlevé certaines parties non pertinentes. Tous les codes utilisés peuvent être trouvés dans [8].

Nous commençons par définir une fonction `hadamard(n, dtype=int)` qui a comme argument sa taille ainsi que son type (optionnel, qui peut par exemple correspondre à une matrice complexe). La première chose que fait cette fonction est de définir une variable `lg2` en fonction de la taille n choisie. Cette variable va correspondre au nombre de fois que nous devrons répéter la construction de Sylvester afin d'obtenir notre matrice d'Hadamard de taille n. Nous vérifions

ensuite si n est une puissance de 2, en élevant 2 à la puissance `lg2` et si $2^{\text{lg2}} \neq n$, alors le programme soulèvera une erreur ainsi qu'un message indiquant une taille incorrecte.

La construction va être faite par itération. Nous commençons par définir `H` comme étant une matrice d'Hadamard de taille 1, c'est-à-dire la matrice $\mathcal{H}_1 = (1)$. On utilise pour cela une donnée de type "array". Pour continuer, nous utilisons de façon directe la formule liant les matrices de taille 2^n et 2^{n-1} dans la construction de Sylvester. Nous utilisons pour cela deux fonctions de NumPy : `hstack` et `vstack`. La fonction `hstack` permet de stocker les tableaux (correspondant à nos matrices) de façon horizontale, en tant que colonnes, tandis que `vstack` nous permet de les stocker de façon verticale, en tant que rangs. En itérant la construction `lg2` fois, nous obtenons au final une matrice d'Hadamard de taille n.

```
def hadamard(n, dtype=int):
    """
    Construction d'une matrice d'Hadamard par la méthode de Sylvester.
    n doit être une puissance de 2
    Paramètres de la matrice
    ----------
    n : int
        L'ordre 'n' de la matrice doit être une puissance de 2.
    dtype : dtype, optional
        Type de données à construire (complexe par exemple)
    Données obtenues
    -------
    H : (n, n) ndarray
        La matrice d'Hadamard
    Exemples d'une matrice complexe et d'une matrice réelle
    --------
    >>> from scipy.linalg import hadamard
    >>> hadamard(2, dtype=complex)
    array([[ 1.+0.j,  1.+0.j],
           [ 1.+0.j, -1.-0.j]])
    >>> hadamard(4)
    array([[ 1,  1,  1,  1],
           [ 1, -1,  1, -1],
           [ 1,  1, -1, -1],
           [ 1, -1, -1,  1]])
    """

    #on commence par vérifier les conditions sur n
    if n < 1: #si n<1, on pose le logarithme égal à 0
        lg2 = 0
    else:
        lg2 = int(math.log(n, 2)) #sinon, on utilise le logarithme de n en base
    2 pour obtenir la taille de la matrice
    if 2 ** lg2 != n: #vérification que la taille choisie est une puissance de 2
     et qu'on obtient n, sinon on appelle une erreur avec un message
        raise ValueError("n doit être un entier positif, et n doit être une
    puissance de 2")

```

```
36     H = np.array([[1]], dtype=dtype) #on crée la matrice d'Hadamard de taille 1
    à l'aide de numpy
37
38     #Construction de Sylvester par itération
39     for i in range(0, lg2): #on itère lg2 fois pour obtenir notre matrice grâce
    à la construction de Sylvester
40         H = np.vstack((np.hstack((H, H)), np.hstack((H, -H)))) #on redéfinit la
    valeur de H: vstack vient ajouter un rang composé des matrices stockées avec
     hstack de façon verticale
41
42     return H #on obtient la matrice de taille 2 une fois que la boucle est finie
```

Listing 1 – Construction de Sylvester via SciPy.

Nous pouvons donner quelques exemples de résultats obtenus grâce à ce programme :

```
  3 print(hadamard(8))
  2 print(hadamard(16))
  1 print(hadamard(32))
58
:w !python
[[ 1  1  1  1  1  1  1  1]
 [ 1 -1  1 -1  1 -1  1 -1]
 [ 1  1 -1 -1  1  1 -1 -1]
 [ 1 -1 -1  1  1 -1 -1  1]
 [ 1  1  1  1 -1 -1 -1 -1]
 [ 1 -1  1 -1 -1  1 -1  1]
 [ 1  1 -1 -1 -1 -1  1  1]
 [ 1 -1 -1  1 -1  1  1 -1]]
[[ 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1]
 [ 1 -1  1 -1  1 -1  1 -1  1 -1  1 -1  1 -1  1 -1]
 [ 1  1 -1 -1  1  1 -1 -1  1  1 -1 -1  1  1 -1 -1]
 [ 1 -1 -1  1  1 -1 -1  1  1 -1 -1  1  1 -1 -1  1]
 [ 1  1  1  1 -1 -1 -1 -1  1  1  1  1 -1 -1 -1 -1]
 [ 1 -1  1 -1 -1  1 -1  1  1 -1  1 -1 -1  1 -1  1]
 [ 1  1 -1 -1 -1 -1  1  1  1  1 -1 -1 -1 -1  1  1]
 [ 1 -1 -1  1 -1  1  1 -1  1 -1 -1  1 -1  1  1 -1]
 [ 1  1  1  1  1  1  1  1 -1 -1 -1 -1 -1 -1 -1 -1]
 [ 1 -1  1 -1  1 -1  1 -1 -1  1 -1  1 -1  1 -1  1]
 [ 1  1 -1 -1  1  1 -1 -1 -1 -1  1  1 -1 -1  1  1]
 [ 1 -1 -1  1  1 -1 -1  1 -1  1  1 -1 -1  1  1 -1]
 [ 1  1  1  1 -1 -1 -1 -1 -1 -1 -1 -1  1  1  1  1]
 [ 1 -1  1 -1 -1  1 -1  1 -1  1 -1  1  1 -1  1 -1]
 [ 1  1 -1 -1 -1 -1  1  1 -1 -1  1  1  1  1 -1 -1]
 [ 1 -1 -1  1 -1  1  1 -1 -1  1  1 -1  1 -1 -1  1]]
[[ 1  1  1 ...  1  1  1]
 [ 1 -1  1 ... -1  1 -1]
 [ 1  1 -1 ...  1 -1 -1]
 ...
 [ 1 -1  1 ...  1 -1  1]
 [ 1  1 -1 ... -1  1  1]
 [ 1 -1 -1 ...  1  1 -1]]
```

Figure 1 – Matrices $\mathcal{H}_8, \mathcal{H}_{16}, \mathcal{H}_{32}$ construites à l'aide de l'algorithme de Sylvester.

Ce sont des matrices d'Hadamard d'ordre 8, 16 et 32 (on ne peut ici visualiser entièrement l'ordre 32 sur l'écran).

Un inconvénient de cette méthode qu'on peut de suite remarquer est la limitation dans les tailles de matrices que l'on peut construire. En effet, nous sommes limités à des matrices dont la taille est une puissance de 2. C'est pour cette raison que nous allons ensuite étudier des méthodes plus générales, dont la construction de Paley.

2.3 Construction de Paley

La construction de Paley est une méthode qui permet de construire des matrices d'Hadamard à partir de corps finis. Raymond Paley a établi sa construction en 1933.

Rappel. Un corps K est un ensemble muni des opérations $(+, \times)$ telles que :

— $(K, +, \times)$ est un anneau commutatif unitaire.

— $(K\backslash\{0_K\}, \times)$ est un groupe.

On dit que le corps K est fini si son cardinal est fini.

Pour construire des matrices grâce à la méthode de Paley, on se place dans un corps fini F_q, où q est la puissance d'un nombre premier impair. La construction dépend ensuite de si q est congru à 1 ou à 3 modulo 4. Dans les deux cas, on fait appel au caractère quadratique d'un élément, aussi appelé symbole de Legendre, et à la matrice de Jacobsthal. On va donc les définir avant de les utiliser.

Définition 10. Soit p un nombre premier et F_p un corps fini. On dit qu'un élément $a \in \mathsf{F}_p$ est un carré parfait modulo p s'il existe $b \in \mathsf{F}_p$ tel que $a \equiv b^2 (\mathtt{mod\ p})$. Le caractère quadratique est donc défini par :

$$\chi(a) = \begin{cases} 1 & \text{si } a \text{ est un carré parfait modulo } p \text{ et } a \not\equiv 0(\mathtt{mod\ p}), \\ -1 & \text{si } a \text{ n'est pas un carré parfait modulo } p, \\ 0 & \text{si } a \equiv 0(\mathtt{mod\ p}). \end{cases}$$

En d'autres termes, il nous indique si $a \in \mathsf{F}_p$ est un carré parfait ou non.

Nous pouvons ensuite définir la matrice de Jacobsthal.

Définition 11. La matrice de Jacobsthal $\mathcal{Q}$ dans F_q est la matrice de taille $q \times q$ dont les coefficients sont déterminés par le caractère quadratique de la différence entre les index des rangs et des colonnes. Si on veut déterminer la valeur de $\mathcal{Q}_{a,b}$, où a est le rang et b la colonne, avec $a, b \in \mathsf{F}_q$, alors $\mathcal{Q}_{a,b} = \chi(a - b)$.

Pour construire ensuite une matrice d'Hadamard grâce à ces outils, on distingue deux cas :

— Si q est congru à $3(\mathtt{mod\ 4})$, alors :

$$\mathcal{H}_{q+1} = I + \begin{pmatrix} 0 & j^T \\ -j & \mathcal{Q} \end{pmatrix}$$

où I est la matrice identité de taille $q+1$, j est le vecteur colonne de taille q avec uniquement des 1. On obtient une matrice de taille $q + 1$.

— Si q est congru à $1(\mathtt{mod\ 4})$, on construit $\mathcal{H}_{2(q+1)}$ à partir de $\begin{pmatrix} 0 & j^T \\ j & \mathcal{Q} \end{pmatrix}$. On remplace tous les 0 par la matrice $\begin{pmatrix} 1 & -1 \\ -1 & -1 \end{pmatrix}$, les 1 par $\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ et les -1 par $\begin{pmatrix} -1 & -1 \\ -1 & 1 \end{pmatrix}$. On obtient une matrice de taille $2(q + 1)$.

Nous pouvons donner un exemple pour mieux comprendre.

Exemple 12. Dans F_7, on a $\chi(2) = 1$ car $3^2 \equiv 2 (\mathrm{mod}\ 7)$, mais $\chi(3) = -1$ car aucun élément au carré de F_7 n'est égal à 3. On calcule tous les carrés, ce qui nous donne : $1 = 1^2 = 6^2$, $4 = 2^2 = 5^2$ et $2 = 3^2 = 4^2$. On a :

- $\chi(0) = 0$,
- $\chi(1) = \chi(2) = \chi(4) = 1$,
- $\chi(3) = \chi(5) = \chi(6) = 1$.

On obtient donc

$$\mathcal{Q} = \begin{pmatrix} 0 & -1 & -1 & 1 & -1 & 1 & 1 \\ 1 & 0 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & 0 & -1 & -1 & 1 & -1 \\ -1 & 1 & 1 & 0 & -1 & -1 & 1 \\ 1 & -1 & 1 & 1 & 0 & -1 & -1 \\ -1 & 1 & -1 & 1 & 1 & 0 & -1 \\ -1 & -1 & 1 & -1 & 1 & 1 & 0 \end{pmatrix}.$$

Comme $7 \equiv 3 (\mathrm{mod}\ 4)$, nous sommes dans le cas 1. On a donc :

$$\mathcal{H}_8 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} + \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ -1 & 0 & -1 & -1 & 1 & -1 & 1 & 1 \\ -1 & 1 & 0 & -1 & -1 & 1 & -1 & 1 \\ -1 & 1 & 1 & 0 & -1 & -1 & 1 & -1 \\ -1 & -1 & 1 & 1 & 0 & -1 & -1 & 1 \\ -1 & 1 & -1 & 1 & 1 & 0 & -1 & -1 \\ -1 & -1 & 1 & -1 & 1 & 1 & 0 & -1 \\ -1 & -1 & -1 & 1 & -1 & 1 & 1 & 0 \end{pmatrix}$$

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

2.3.1 Algorithme de construction

Nous pouvons maintenant utiliser cette construction pour définir un algorithme en Python permettant d'obtenir une matrice d'Hadamard. Nous étudions uniquement le cas où la taille du corps est un nombre premier, étant donné que dans le cas d'une puissance d'un premier nous aurions besoin des extensions de corps.

Nous commençons par définir une fonction `quad_char(a,n)`, qui permet de déterminer si l'élément a, donné comme argument dans la fonction, est un carré parfait ou non dans F_n, $n \in \mathsf{N}$ un nombre premier [1]. On commence par construire le corps fini `F_n` dans la fonction sous forme d'une liste. On crée ensuite une liste vide `sq` ("sequence") qui servira à stocker les carrés des éléments. On la remplit ensuite avec les éléments au carré du corps modulo n. Il s'agit de la liste de tous les carrés que l'on peut obtenir à partir des éléments de ce corps. Ensuite, on vérifie les trois conditions pour déterminer le caractère quadratique comme défini dans la Déf 10.

On définit ensuite une fonction `jacob_matrix(n)` qui nous permet d'obtenir la matrice de Jacobsthal d'ordre n. On crée une liste vide `M_J`, puis une première boucle pour construire les n colonnes. On crée une liste vide `row`, qui correspond à un rang vide. Dans une deuxième boucle, on construit les colonnes, en remplissant chaque rang des caractères quadratiques appropriés aux indices du rang moins celui de la colonne, le tout modulo n. A la fin de cette boucle, on ajoute le rang à la liste vide `M_J`, et on recommence $n-$fois jusqu'à avoir la matrice de Jacobsthal.

On définit enfin la fonction `hadamard(n)`, qui permet de construire une matrice d'Hadamard de taille $n+1$ ou $2(n+1)$ selon le cas. On définit une matrice `H` vide et une matrice identité `I` de taille $n+1$ toutes les deux. Si n modulo 4 est égal à 0, on est dans le premier cas de la construction. On ajoute à `H` la matrice identité et une matrice par blocs définie en fonction de la matrice de Jacobsthal de taille n comme expliqué précédemment dans la Section 2.3. Dans le deuxième cas, on doit remplacer chaque coefficient de la matrice par une matrice de taille 2×2 en fonction de la valeur du coefficient. On utilise pour cela une boucle avec `hstack` et `vstack`, et une autre fonction `paley(i)`, qui détermine par quelle matrice de taille 2×2 on remplace chaque coefficient.

```
1  import math
2  import numpy as np
3
4  def quad_char(a,n): #déteriner si un élément est un carré parfait dans le corps
       fini ou non
5      F_n = [i for i in range(n)] #corps fini de taille n
6      sq = [] #liste vide pour stocker les carrés
7      for i in range(len(F_n)): #on vérifie tous les éléments
8          sq.append(int(i**2)%n) #carrés modulos 7
9      if a%n == 0:
10         return 0
11     if a%n in sq: #si a appartient à sq, c'est un carré parfait
12         return 1 #le quadratic character est égal à 1
13     else:
14         return -1 #sinon -1
15
16 #fonction utilisée pour construire la matrice de jacobsthal
17 def jacob_matrix(n): #taille n donnée
18     M_J = [] #liste vide
19     for j in range(n): #boucle pour construire les n rangs
20         row = [] #créer rang vide
21         for i in range(n): #boucle pour construire chaque élément dans le rang (
```

1. Le n ici correspond au q utilisé dans la Déf 11

```
    ce qui correspond à l'élément de chaque colonne)
22             row.append(quad_char((j-i)%n,n)) #on ajoute chaque quadratic
    character au rang selon l'indice du rang - de la colonne modulo n
23         M_J.append(row) #on ajoute chaque rang à la matrice
24     return np.array(M_J)
25 
26 def paley2(i): #détermine par quelle matrice 2*2 on remplace chaque coefficient,
     en fonction de si c'est 1,-1, ou 0
27     A = np.zeros((2,2), int)
28     if i == 1:
29         A = np.array([[1,1],[1,-1]])
30     elif i == -1:
31         A = np.array([[-1,-1],[-1,1]])
32     elif i == 0:
33         A = np.array([[1,-1],[-1,-1]])
34     return A
35 
36 def hadamard(n): #définir matrice Hadamard, n est la taille du corps fini choisi
37     H = np.zeros((n+1,n+1),int) #créer matrice vide
38     I = np.identity((n+1),int) #créer matrice identité
39     if n%4 == 3: #Paley 1
40         H = I + np.block([[0, np.ones(n)], [np.full((n,1), -1), jacob_matrix(n)
    ]]) #matrice par blocs
41         return(np.array(H)) #matrice construite, taille n+1
42     if n%4 == 1: #Paley 2
43         H_2 = np.block([[0, np.ones(n)], [np.full((n,1), 1), jacob_matrix(n)]])
44         H_2 = np.vstack(np.hstack(paley2(H_2[j][i]) for i in range(n+1)) for j
    in range(n+1)) #construction matrice par for loop avec vstack et hstack
45         return(np.array(H_2)) #matrice construite, taille 2(n+1)
```

Listing 2 – Construction de Paley.

Nous pouvons donner un exemple de résultat obtenu grâce à cet algorithme :

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

Figure 2 – Matrices $\mathcal{H}_8, \mathcal{H}_{12}$ construites à l'aide de l'algorithme de Paley.

En comparaison à l'algorithme de Sylvester, nous avons donc moins de restrictions quant à la taille des matrices constructibles. Par exemple, nous n'aurions pas pu construire la matrice

$\mathcal{H}_{12}$ avec l'algorithme de Sylvester, 12 n'étant pas une puissance de 2. Mais cela est possible par l'algorithme de Paley, où on se situe dans le cas $12 - 1 = 11 \equiv 3 (\text{mod } 4)$.

On peut également mesurer le temps mis par les deux algorithmes pour construire une matrice de même taille, afin de les comparer et savoir lequel est le plus rapide.

On va utiliser la fonction `time` et le module `time` de Python afin de les comparer pour une matrice de taille n.

Voici un tableau suivi d'un graphique représentant les trois premières valeurs de n qui illustrent cette comparaison :

Algorithme \ n	4	8	32	128	8192
Sylvester	0,00026s	0,0013s	0,0018s	0,0026s	1,35s
Paley	0,0017s	0,0033s	0,038s	1,45s	>2h

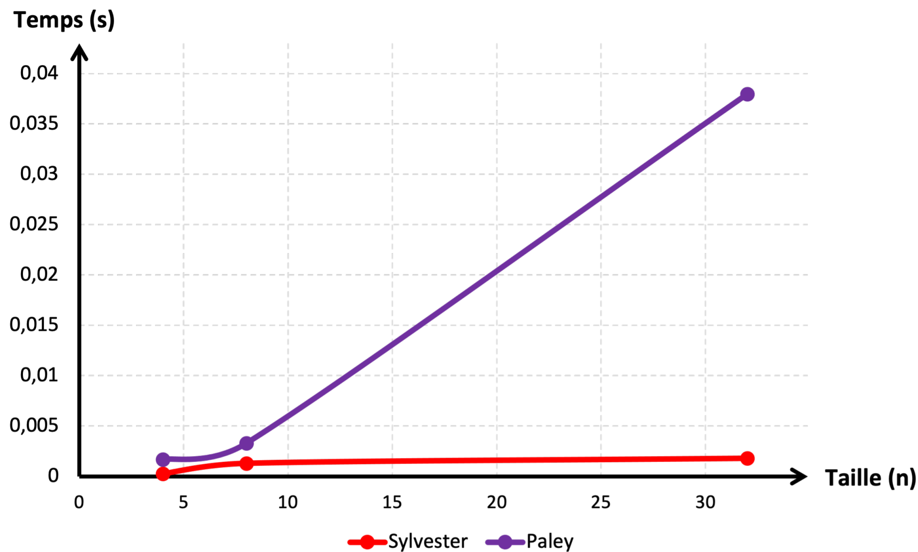


Figure 3 – Temps d'exécution des programmes en fonction de la taille.

On constate alors que l'algorithme de Sylvester est bien plus rapide, étant donné qu'il est plus limité quant à la taille à construire et procède à la construction en beaucoup moins d'étapes. Dans l'algorithme de Paley, une fois que la taille est assez importante, il devient très lent du fait qu'il doit évaluer chaque coefficient du corps fini afin de déterminer si c'est un carré parfait, puis lors de la construction de la matrice de Jacobsthal où il doit évaluer à nouveau chaque coefficient de la matrice.

2.4 Visualisation des matrices

On peut visualiser plus facilement la structure des matrices dans Python en remplaçant les -1 par des 0. On utilise pour cela la fonction `visualisation(M)`, qui prend pour argument une

matrice M et qui remplace tous les coefficients -1 par 0 à l'aide d'une boucle évaluant tous les coefficients.

```python
def visualisation(M): #simplifie la visualisation des matrices en remplaçant les -1 par des 0
    for (col, row), value in np.ndenumerate(M):
        if M[col,row] == -1 :
            M[col,row] = 0
    return np.array(M)
```

Listing 3 – Visualisation des matrices.

Par exemple, on obtient le résultat suivant pour la matrice $\mathcal{H}_8$:

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

Figure 4 – Matrices $\mathcal{H}_8$,$\mathcal{H}_{12}$ visualisée avec des 0 remplaçant les -1.

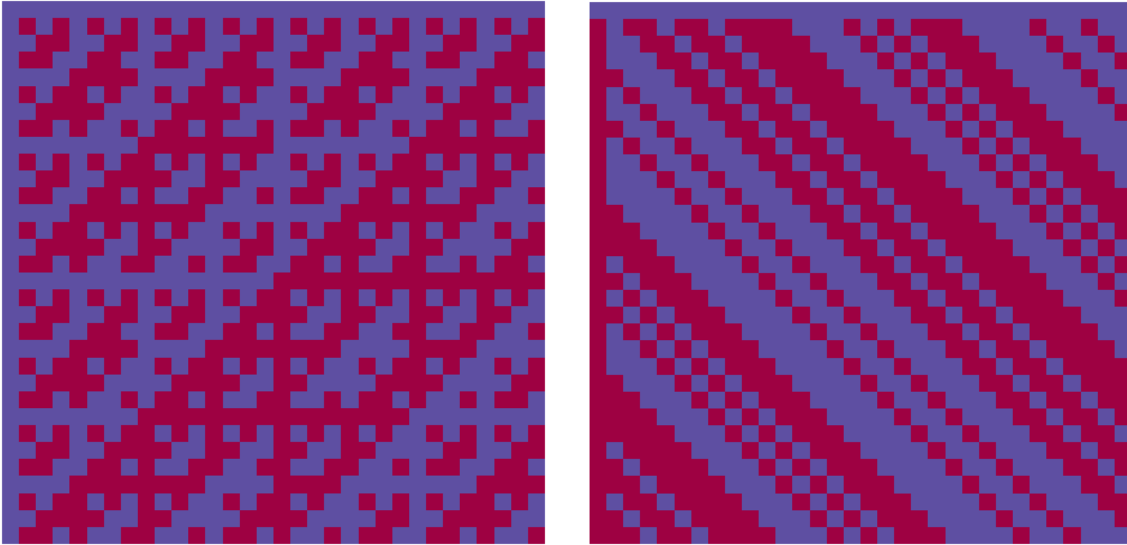
On peut également utiliser Matplotlib pour les représenter plus efficacement avec un code couleur. On utilise pour cela le code suivant, qui prend comme paramètre mat, une matrice d'Hadamard où les 1 sont représentés par la couleur lila et les -1 représentés par la couleur rose.

```python
mat = hadamard(n) #on choisit la matrice à visualiser
plt.figure(figsize=(10,10)) #on crée la taille de l'image (en pouces par défaut)
plt.imshow(mat, cmap='Spectral') #on chosiit quelle image on veut montrer et la couleur
plt.axis(False) #on désactive l'axe
plt.show() #on montre le résultat obtenu dans le terminal
```

Listing 4 – Visualisation des matrices avec Matplotlib.

On peut ainsi observer plus facilement la structure des matrices d'Hadamard selon leur construction, et comment les motifs varient selon le cas de Paley.

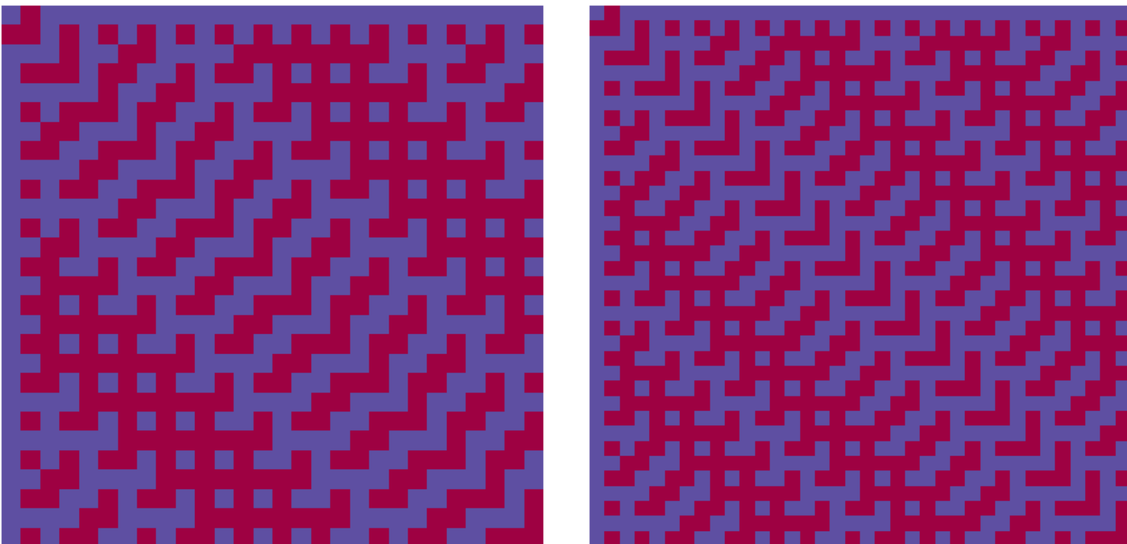
On prend par exemple une matrice de taille 32 construite avec les deux algorithmes différents, et on voit que leurs structures et motifs ne se ressemblent pas.



(a) Matrice de taille 32 avec Sylvester. (b) Matrice de taille 32 avec Paley dans le cas 1.

Figure 5 – Matrices de taille 32 selon les différents algorithmes.

On peut également voir que le motif dans le cas 2 de Paley est différent, avec par exemple les matrices hadamard(13) et hadamard(17), qui seront des matrices de taille 28 et 36 respectivement. Cela vient du fait que $13 \equiv 17 \equiv 1(\mathrm{mod}\ 4)$, ce qui correspond au cas 2, alors que $31 \equiv 3(\mathrm{mod}\ 4)$, qui correspondait ainsi au cas 1.



(a) Matrice de taille 28. (b) Matrice de taille 36.

Figure 6 – Matrices de Paley dans le cas 2.

3 Matrices de Walsh

Les matrices de Walsh, $\mathcal{W}_n,\ n \in \mathbb{N}$, sont un cas particulier des matrices d'Hadamard. Elles ont été construites par Joseph Leonard Walsh en 1923.

Elles sont de trois formes différentes. Elles peuvent être dans un ordre : naturel, séquentiel, et dyadique. Nous pouvons trouver plus de détails dans [9].

(i) Ordre naturel

L'ordre naturel est aussi connu sous le nom de l'ordre d'Hadamard. La matrice est obtenue par la méthode de Sylvester en utilisant le produit de Kronecker. On a alors $\mathcal{W}_n = \mathcal{H}_n$.

(ii) Ordre séquentiel

L'ordre séquentiel est aussi connu sous le nom de l'ordre de Walsh. La matrice est obtenue en permutant d'une certaine façon les lignes de la matrice d'Hadamard.

Il faut tout d'abord compter pour chaque ligne le nombre de changement de signe.

Ensuite, il faut ordonner ces changements de signe par ordre croissant. On obtient finalement la matrice dans son ordre séquentiel.

(iii) Ordre dyadique

L'ordre séquentiel est aussi connu sous le nom de l'ordre de Paley. La matrice est obtenue également en permutant d'une certaine façon les lignes de la matrice d'Hadamard.

Il faut tout d'abord compter pour chaque ligne le nombre de changements de signe.

Ensuite, il faut convertir chaque nombre de changements de signe en bit associé en passant par le code Gray, puis en l'inversant et finalement en revenant au nombre associé (voir le tableau ci-dessous).

Le code de Gray est un codage binaire permettant de ne modifier qu'un seul bit à la fois quand un nombre est augmenté d'une unité. Il a pour intérêt principal d'éviter les états transitoires, ce qui peut perturber certaines opérations, notamment dans la transmission de données.

Nous pouvons en apprendre plus sur son origine et son fonctionnement dans [10].

Voici le tableau de conversion afin d'obtenir l'ordre dyadique :

* Pour $n \leq 3$:

n	0	1	2	3
Gray	00	01	11	10
Inversé	00	10	11	01
n'	0	3	2	1

* Pour $3 < n < 8$:

n	0	1	2	3	4	5	6	7
Gray	000	001	011	010	110	111	101	100
Inversé	000	100	110	010	011	111	101	001
n'	0	7	4	3	2	5	6	1

* Pour $n \geq 8$: on continue de la même façon.

Exemple 13. On donne un exemple pour une matrice de Walsh de taille 8. On a les résultats suivants :

— Ordre naturel :

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{pmatrix}.$$

On obtient cette matrice à l'aide du produit de Kronecker.

— Ordre séquentiel :

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \end{pmatrix}.$$

En comptant le nombre de changements de signe de chaque rang, on obtient $0, 1, 2, 3, 4, 5, 6, 7$ changements, ce qui est bien un ordre croissant.

— Ordre dyadique :

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{pmatrix}.$$

En comptant le nombre de changements de signe de chaque rang avec le code de Gray, on obtient $0, 7, 3, 4, 1, 6, 2, 5$ changements, ce qui est bien un ordre dyadique.

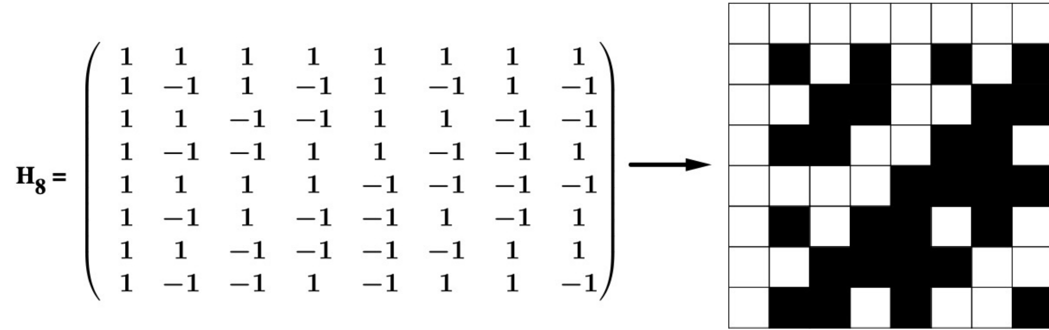


Figure 7 – Transformation : Ordre Naturel.

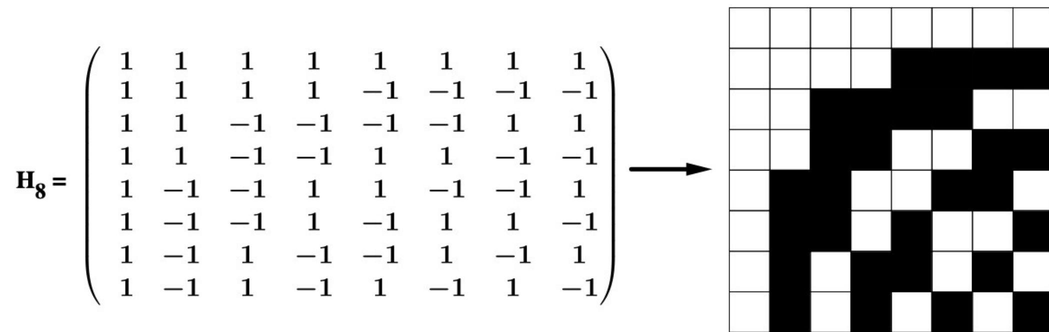


Figure 8 – Transformation : Ordre Séquentiel.

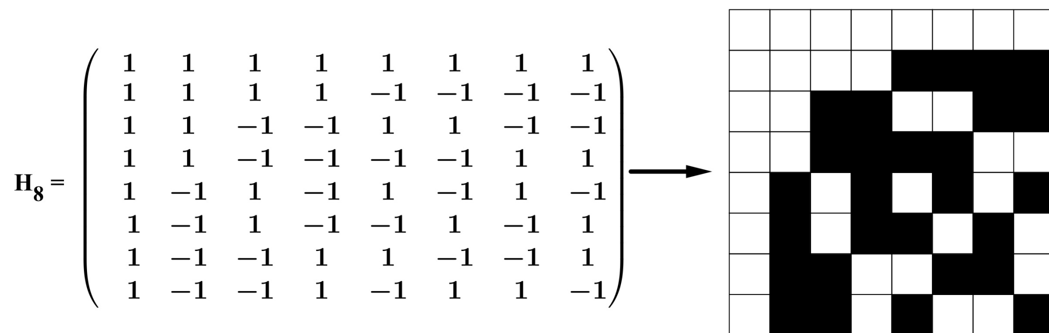


Figure 9 – Transformation : Ordre Dyadique.

4 Correction d'erreurs

Les codes de correction d'erreurs sont destinés à corriger les altérations possibles lors de la transmission d'informations telles que des messages, des images ou des sons, souvent sur des canaux de communications peu fiables.

Nous allons ici essayer d'utiliser les matrices d'Hadamard afin de corriger des messages, en utilisant les travaux précédemment réalisés dans [9].

4.1 Théorie

La dimension 2 du code d'Hadamard nécessite davantage de recherche et est surtout utilisée lors de la correction d'images ; nous avons donc décidé de nous concentrer sur le code d'Hadamard en dimension 1, qui nous permet de corriger des messages, d'après [9], Sections $2.1-2.2$.

Le code d'Hadamard à n-bit est un code non-linéaire. Il est généré par les lignes de la matrice d'Hadamard $\mathcal{H}_n$ de taille $n \times n$, avec $n \in \mathbb{N}$. La matrice que nous utilisons doit être construite avec la méthode de Sylvester, étant donné qu'elle doit être symétrique pour que le code fonctionne. Ce n'est pas forcément le cas des matrices construites à l'aide de la méthode de Paley, comme nous pouvons le voir par exemple dans l'image 5b. Cette matrice est alors capable de crypter $k = \log_2(n)$ messages.

Généralement, le code d'Hadamard à n-bit peut corriger un total de $\frac{n}{2} - 1$ types d'erreurs. D'après [9], le nombre d'erreurs doit se trouver soit dans l'intervalle $I_1 = \left[1, \frac{n}{4} - 1\right]$ ou soit dans l'intervalle $I_2 = \left[\frac{3n}{4} + 1, n\right]$, sinon on ne peut pas obtenir le décodage exact.

Nous pourrions étudier ces intervalles en testant le programme utilisé dans la Section 4.2 avec des pourcentages d'erreurs différents à chaque fois. Nous pourrions alors comparer les intervalles obtenus lors de cette expérience et les intervalles théoriques, afin de voir s'ils correspondent.

Le processus de décodage est basé sur le spectre d'Hadamard. Nous pouvons en trouver une représentation pour les matrices d'Hadamard de taille 8 dans [9], page 2 :

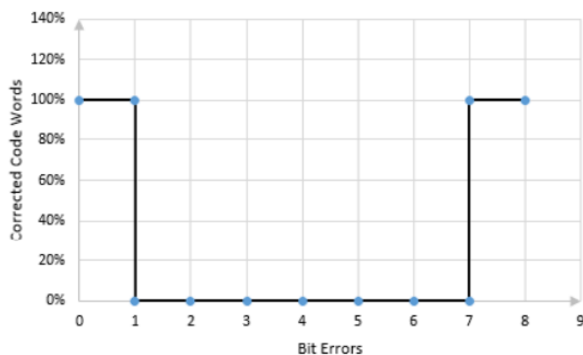


Figure 10 – Capacité de correction d'un code avec une matrice $\mathcal{H}_8$.

D'une manière générale, on peut généraliser et on peut donc voir que le code est capable de corriger $\frac{n}{2} - 1$ des cas d'erreurs.

Nous commençons par un exemple afin de mieux pouvoir expliquer la théorie.

Exemple 14. On suppose qu'on envoie un message composé de plusieurs lettres, chacune étant représentée par un vecteur de longueur 8, composé de 1 et -1. Par exemple, on suppose qu'on a voulu transmettre la lettre $A = \begin{bmatrix} 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \end{bmatrix}$, et qu'une erreur s'y est glissée, donnant le vecteur $A' = \begin{bmatrix} 1 & -1 & 1 & 1 & 1 & -1 & 1 & -1 \end{bmatrix}$. On se trouve donc dans le cas d'une matrice d'Hadamard $\mathcal{H}_8$ de taille 8×8, construite avec la méthode de Sylvester.

On procède en trois étapes :

1. On commence par multiplier la lettre à corriger par la matrice d'Hadamard $\mathcal{H}_8$:

$$A' \cdot \mathcal{H}_8 = \begin{bmatrix} 2 & 6 & -2 & 2 & 2 & -2 & -2 & 2 \end{bmatrix}.$$

2. On identifie le coefficient dont la valeur absolue est supérieure aux autres. Ici, c'est le coefficient 6 qui est à la deuxième position.
3. Le coefficient 6 étant en deuxième position, le vecteur correct correspond alors à la deuxième ligne de la matrice d'Hadamard.

On retrouve donc bien $A = \begin{bmatrix} 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \end{bmatrix}$

Nous pouvons répéter ce procédé de façon plus générale, en utilisant la matrice $\mathcal{H}_n$ pour corriger des vecteurs A_n de taille $n \in \mathbb{N}$.

4.2 Algorithme de correction

Dans notre algorithme, nous allons transmettre des messages utilisant les caractères du codage ASCII (*American Standard Code for Information Interchange*). Ce codage étant composé de 128 caractères, chaque lettre du message sera associée à une ligne de la matrice d'Hadamrd $\mathcal{H}_{128}$. On la construit avec la méthode de Sylvester, ce qui nous donne le résultat suivant :

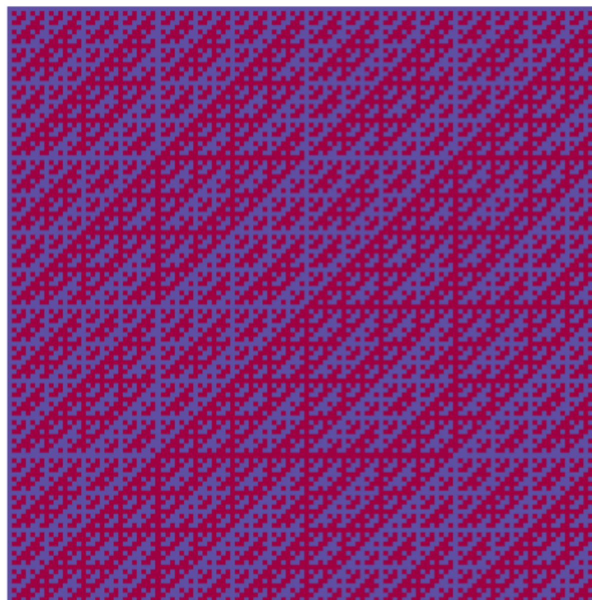


Figure 11 – Matrice d'Hadamard de taille 128.

On associe donc chaque caractère ASCII à une ligne de cette matrice d'Hadamard comme suivant :

Décimal	Caractère	Ligne de $\mathcal{H}_{128}$
0	[Caractère nul]	Première ligne de la matrice $\mathcal{H}_{128}$
1	[Début de titre]	Deuxième ligne de la matrice $\mathcal{H}_{128}$
2	[Début de texte]	Troisième ligne de la matrice $\mathcal{H}_{128}$
...	...	...
125	}	Ligne 126 de la matrice $\mathcal{H}_{128}$
126		Ligne 127 de la matrice $\mathcal{H}_{128}$
127	[Supprimer]	Ligne 128 de la matrice $\mathcal{H}_{128}$

Table 1 – Correspondance caractère ASCII - matrice d'Hadamard.

On remarque que certaines lignes correspondent à des instructions d'un terminal plutôt que des symboles, mais cela ne change rien pour nous. On définit plusieurs fonctions pour cet algorithme.

On commence par définir la fonction `convertLetter(l)`. Elle prend un caractère comme paramètre et donne la ligne de la matrice $\mathcal{H}_{128}$ qui correspond en utilisant l'indice de son caractère ASCII grâce à la fonction intégrée de Python `ord`. Par exemple, si on demande "`print(convertLetter("e"))`", on obtient le résulat suivant, qui correspond à la 101e ligne de $\mathcal{H}_{128}$ car `ord(e)`=101 :

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

Figure 12 – Ligne de $\mathcal{H}_{128}$ correspondant à "e".

On définit ensuite la fonction `correctLine(c,n)` qui prend comme argument une liste `c` correspondant à une lettre et sa taille `n` (ici on utilisera 128). On réalise dans la variable `d` la multiplication entre `c` et la matrice $\mathcal{H}_{128}$, ce qui nous donne une liste. On cherche ensuite la valeur absolue maximale dans cette liste, que l'on nomme `dMax` et la position à laquelle elle se trouve en utilisant la fonction index. Ensuite, selon la valeur de `dMax`, on va voir si on peut corriger la lettre transmise en cas d'erreurs. Si `dMax` est égal à 64, alors il s'agit de la distance d'Hamming, une valeur indiquant le nombre maximal de changements entre deux vecteurs, qui est atteinte et on ne peut pas la corriger car cette valeur apparaîtra plusieurs fois dans la liste `d`. On affiche alors un message pour dire que la situation est trop ambiguë pour corriger. Sinon, l'indice auquel la plus grande valeur absolue se trouve correspond à la lettre correcte.

Pour illustrer ce fonctionnement plus clairement, prenons un exemple sur un message erroné. Pour simuler une mauvaise transmission, on définit une fonction `brouille(l,n)` qui va brouiller une liste *l* en `n` endroits. Pour cela, on remplace aléatoirement n éléments de la liste par leur opposé. Par exemple, si on choisit de brouiller "e" en changeant aléatoirement 20 éléments, on obtient la liste suivante qui comprend des erreurs :

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

Figure 13 – Lettre "e" erronée.

En appliquant la fonction de correction à cette lettre brouillée et en affichant `d` et `dMaxIndex`, on obtient le résultat suivant :

```
[-6, -2, 2, -2, -6, -2, 2, -2, 2, 6, 10, 6, -6, -2, 2, -2, -2, -14, -2, -6, -10, 10, 6, 2, 14, 2, -2, -6, -2, 18, -2, -6, 2, -10, 2,
-2, -14, 6, 2, -2, 10, -2, 10, 6, -14, 6, 2, -2, -10, -6, -2, -6, -2, 2, 6, 2, 6, 10, -2, -6, 6, 10, -2, -6, -2, -6, 6, 10, -10, -14,
 -2, 2, 6, 2, -2, 2, 6, 2, -2, 2, -6, 6, -6, -2, 10, -10, -6, -2, -6, 6, -6, -2, 18, -2, 2, 6, -10, 2, 6, 10, -2, 106, -2, 2, -2, 10,
 -2, 2, 14, -6, -2, 2, 2, -2, -6, -2, 2, -2, -6, -2, 2, -2, -6, -2, 10, 6, 2, 6]
101
e
```

Figure 14 – Correction de la lettre "e" brouillée.

On remarque que le nombre effectif d'erreurs est inférieur au nombre d'erreurs introduites. En effet, il est possible qu'un même coefficient soit aléatoirement modifié un nombre pair de fois, ce qui fait qu'il reste correct. Il faudrait ici rajouter une comparaison entre la lettre correcte et erronée pour obtenir le nombre d'erreurs effectivement introduites. On peut également remarquer que la probabilité que la lettre brouillée corresponde à une lettre existante est négligeable étant donné la taille des vecteurs. Elle est ici de $\frac{1}{2^{128}}$, étant donné que nous avons deux possibilités pour chacun des 128 coefficients du vecteur.

On voit que la valeur la plus élevée dans cette liste est 106, et qu'elle se trouve en 101e position. Cet indice correspond donc à la lettre "e", ce que nous souhaitions.

```
1  import math
2  import numpy as np
3  import random
4  
5  import sylvester as syl
6  
7  def convertLetter(l): #convertir un message de ascii en ligne de la matrice
       H_128
8      H_128 = syl.hadamard(128) #matrice 128
9      return np.array(H_128[ord(l)])
10 
11 def correctLine(c,n): #on rentre une "lettre" c et sa taille n
12     H_n = syl.hadamard(n) #créer la matrice d'Hadamard de taille n
```

```python
    d = list(c.dot(H_n)) #produit entre la matrice et la lettre sous forme de liste
    dMax = max(d, key=abs) #élément en valeur absolue
    dMaxIndex = d.index(dMax) #trouver l'index de la valeur absolue max
    hamming = n/2 #permet de déterminer la capacité de correction maximum
    if abs(dMax) == hamming: #impossible à corriger dans ce cas : la ligne est composée de 0 et de +- n/2
        return "Trop ambigu pour corriger"
    else:
        return chr(int(dMaxIndex)) #version améliorée : se base sur l'indice, peu importe le signe de dMax

def brouille(l,n): #brouille une liste en n endroits
    for i in range(n):
        a = random.randint(0,len(l)-1)
        l[a] = -l[a]
    return l

def correctMessage(m): #utilise le correct line sur un message entier
    for char in m:
        print(correctLine(brouille(convertLetter(char),15), 128))
```

Listing 5 – Correction d'erreurs grâce aux matrices d'Hadamard.

Nous avons également une fonction `correctMessage(m)`, qui prend un message entier et le corrige lettre par lettre.

Nous pouvons voir que si l'on introduit plus d'erreurs pour simuler une transmission plus brouillée, on a parfois des lettres qui ne peuvent pas être corrigées.

Par exemple, si on veut transmettre le message "`Mariner 9`" et qu'on y ajoute 50 erreurs par lettre, puis qu'on y applique la fonction `correctMessage`, on obtient le résultat suivant :

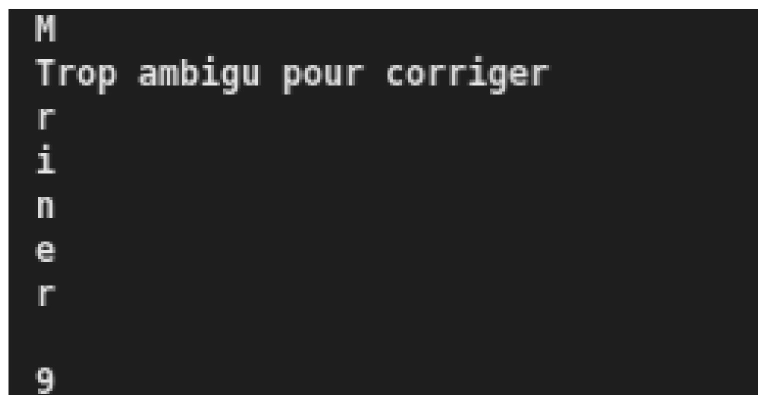


Figure 15 – Correction du message "Mariner 9".

On remarque que certaines lettres sont trop ambiguës pour être corrigées ("a" ici), signifiant que le nombre d'erreurs effectif ne se situe pas dans un des intervalles où le code de correction est possible.

Conclusion

Ainsi, nous avons pu atteindre notre but en utilisant les matrices d'Hadamard : écrire un code de correction d'erreurs qui fonctionne pour des messages.

Pour nous aider, nous avons eu une discussion avec Luca Notarnicola, un doctorant de l'Université du Luxembourg spécialisé en cryptographie. Il nous a assuré que nos démarches dans les algorithmes étaient correctes et nous a donné des pistes pour développer certaines parties.

Nous avons rencontré quelques difficultés au cours du projet, surtout concernant les matrices de Walsh. En effet, nous avons eu du mal à trouver des renseignements et ils étaient parfois contradictoires.

Cependant, nous aurions aimé aller plus loin et pouvoir faire un code de correction efficace pour des images, étant donné que ce sont les missions spatiales qui nous avaient inspirées. Nous n'avons cependant pas pu atteindre notre but final en raison du temps limité et des recherches supplémentaires nécessaires. Pour corriger des images, nous aurions dû trouver un moyen de convertir chaque pixel d'une image en un vecteur associé à une ligne d'une matrice d'Hadamard de taille adaptée. Nous aurions "brouillé" chaque vecteur afin d'introduire des erreurs que nous aurions ensuite corrigées. A partir de là, il aurait fallu trouver un moyen de retranscrire ces vecteurs en pixels et de reformer l'image.

Listings

1 Construction de Sylvester via SciPy. 8
2 Construction de Paley. 12
3 Visualisation des matrices. 15
4 Visualisation des matrices avec Matplotlib. 15
5 Correction d'erreurs grâce aux matrices d'Hadamard. 23

Table des figures

1 Matrices $\mathcal{H}_8, \mathcal{H}_{16}, \mathcal{H}_{32}$ construites à l'aide de l'algorithme de Sylvester. 9
2 Matrices $\mathcal{H}_8, \mathcal{H}_{12}$ construites à l'aide de l'algorithme de Paley. 13
3 Temps d'exécution des programmes en fonction de la taille. 14
4 Matrices $\mathcal{H}_8$,$\mathcal{H}_{12}$ visualisée avec des 0 remplaçant les -1. 15
5 Matrices de taille 32 selon les différents algorithmes. 16
6 Matrices de Paley dans le cas 2. 16
7 Transformation : Ordre Naturel. 19
8 Transformation : Ordre Séquentiel. 19
9 Transformation : Ordre Dyadique. 19
10 Capacité de correction d'un code avec une matrice $\mathcal{H}_8$. 20
11 Matrice d'Hadamard de taille 128. 21
12 Ligne de $\mathcal{H}_{128}$ correspondant à "e". 22
13 Lettre "e" erronée. 23
14 Correction de la lettre "e" brouillée. 23
15 Correction du message "Mariner 9". 24

Références

[1] Hadamard code, May 2020. [Online https://en.wikipedia.org/w/index.php?title=Hadamard_code&oldid=959245432 ; accessed 7. Apr. 2021].

[2] Eric W. Weisstein. Hadamard's Maximum Determinant Problem. *Wolfram Research, Inc.*, Mar 2003.

[3] Images des mathématiques, Apr 2021. [Online, http://images.math.cnrs.fr/La-conjecture-de-Hadamard-I.html ; accessed 7. Apr. 2021].

[4] Dragomir Z. Djokovic. Hadamard matrices of order 764 exist. *arXiv*, Mar 2007.

[5] Jay H. Beder. Conjectures about hadamard matrices. *Journal of Statistical Planning and Inference*, 72(1) :7–14, 1998.

[6] Matrice de hadamard. disponible sur https://fr.wikipedia.org/wiki/Matrice_de_Hadamard, dernière consultation.

[7] Construction de sylvester. [Online https://github.com/scipy/scipy/blob/v1.6.1/scipy/linalg/special_matrices.py#L300-L354, accessed 2. Apr. 2021].

[8] Hadamard matrices, Apr 2021. [Online ; https://github.com/ergzz/hadamard_matrices accessed 8. Apr. 2021].

[9] Andrzej Dziech and Jakob Wassermann. Application of enhanced hadamard error correcting code in video-watermarking and his comparison to reed-solomon code.

[10] Wikipédia. Code de gray — wikipédia, l'encyclopédie libre, 2021. [En ligne ; Page disponible le 9-mars-2021].