Matrices de Hadamard

Lev-Arcady Sellem

2 juillet 2016

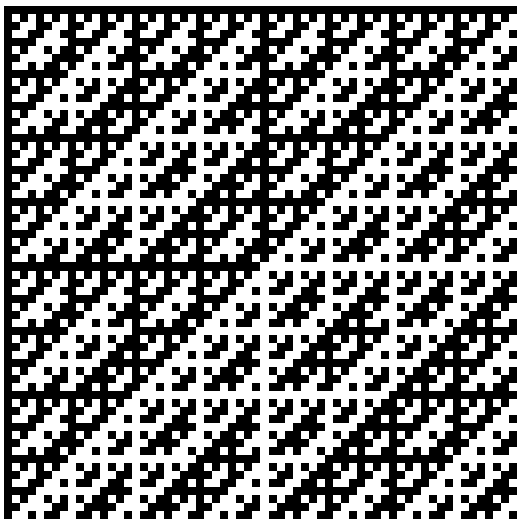


FIGURE 1 – La matrice S64

Table des matières

Résumé

Dans ce TIPE, nous nous intéresserons à une famille particulière de matrices carrées à coefficients dans $\{-1, +1\}$: les matrices dites *de Hadamard*.

Après les avoir définies, nous examinerons les méthodes de recherche et de construction de telles matrices, fondées pour certaines sur une exploitation de leurs propriétés, pour d'autres sur une approche algorithmique, avant de nous pencher sur leurs différentes applications en mathématiques, en informatique et en physique, notamment dans le domaine de l'optique.

I Matrices de Hadamard

I.1 Définition

Soit n un entier naturel non-nul ; on notera $\mathcal{H}_n$ l'ensemble des matrices M de $\mathcal{M}_n(\{-1;+1\})$ vérifiant $M \cdot {}^tM = {}^tM \cdot M = nI_n$, où $\mathcal{M}_n(\{-1;+1\})$ représente l'ensemble des matrices carrées à coefficients dans $\{-1;+1\}$ et I_n représente la matrice identité d'ordre n.

De manière équivalente, $M \in \mathcal{H}_n$ si et seulement si les colonnes (respectivement les lignes) de M forment une base orthogonale de $\mathcal{M}_{n,1}(\{-1;+1\})$ (respectivement $\mathcal{M}_{1,n}(\{-1;+1\})$, ou encore si $\frac{1}{\sqrt{n}}M$ est une matrice orthogonale, les différents produits scalaires en jeu étant les produits scalaires usuels.

Dans ce cas, M est une *matrice de Hadamard d'ordre n.*

I.2 Propriétés

Propriété I.2.1. *Soit M une matrice de Hadamard d'ordre n ; alors, toute matrice obtenue à partir de M par permutation de deux lignes ou de deux colonnes est encore une matrice de Hadamard d'ordre n.*

Propriété I.2.2. *Soit M une matrice de Hadamard d'ordre n ; alors, toute matrice obtenue à partir de M par multiplication d'une ligne ou d'une colonne par -1 est encore une matrice de Hadamard d'ordre n.*

Ces deux propriétés sont immédiates en considérant qu'une matrice est de Hadamard si et seulement si la famille de ses lignes (respectivement de ses colonnes) est orthogonale pour le produit scalaire canonique de $\mathcal{M}_n(\{-1,+1\})$.

I.3 Ordres accessibles

La première question qui se pose est celle de l'existence d'un entier naturel n non-nul tel que $\mathcal{H}_n$ ne soit pas vide ; on se persuade aisément que $\begin{bmatrix}1\end{bmatrix}, \begin{bmatrix}-1\end{bmatrix} \in \mathcal{H}_1$.

On peut ensuite se demander quels sont les $n \neq 0$ tels que $\mathcal{H}_n$ n'est pas réduit à l'ensemble vide ; on dispose alors du théorème suivant :

Théorème I.3.1. $\forall n \in \mathbb{N}^*, \mathcal{H}_n \neq \emptyset \Rightarrow n \leq 2$ ou $n \equiv 0 \pmod 4$

Démonstration. Montrons d'abord que le cas $n = 2$ est possible :

$$\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix} \in \mathcal{H}_2 \Rightarrow \mathcal{H}_2 \neq \emptyset$$

Soit M une matrice de Hadamard de rang $n, n > 2$; en multipliant au besoin les colonnes de M par -1, on peut supposer que la première ligne de M ne contient que des 1.

Le produit scalaire des deux premières lignes étant nul, la deuxième ligne contient autant de 1 que de -1, donc n *est pair*.

Quitte à réorganiser les colonnes de M, on peut supposer que les $n/2$ premiers coefficients de la deuxième ligne de M sont des 1, et les autres des -1 ; considérons maintenant la troisième ligne de M. On note a le nombre de 1 sur cette ligne entre la position 1 et la position $n/2$, et b le nombre de 1 entre la position $n/2+1$ et la position n. D'une part, le produit scalaire de la troisième ligne avec la première est nul, donc $a+b = n/2$. D'autre part, le produit scalaire de la troisième ligne avec la deuxième est nul, donc $a-b = 0$. En conséquence, $a = b = n/4$, donc n *est divisible par 4*. □

Ce procédé ne peut être poursuivi : en passant à la quatrième ligne, chercher à reproduire la démonstration précédente mène à un système de trois équations pour quatre inconnues.

I.4 La conjecture de Hadamard

La relation précédente fournit une condition nécessaire, mais pas suffisante, pour que $\mathcal{H}_n$ soit non-vide ; la *conjecture de Hadamard* consiste à supposer la réciproque vraie, c'est-à-dire :

Conjecture (1893). $\forall n \in \mathbb{N}, \mathcal{H}_n \neq \emptyset \Leftrightarrow n \leq 2$ ou $n \equiv 0 \pmod 4$

À ce jour, aucun contre-exemple n'est venu infirmer cette conjecture. Comme nous le verrons à la section III page 21, elle est vraie pour une infinité de multiples de 4, et en particulier pour toutes les puissances de 2, cas examiné en III.3 page 27 ; expérimentalement, le premier multiple de 4 pour lequel aucune matrice de Hadamard n'a encore été trouvée est 668.

I.5 Conventions d'écriture

Il existe plusieurs manières de représenter une même matrice de Hadamard ; la première est celle utilisée précédemment, à savoir un tableau de $+1$ et de -1. Une autre convention rencontrée souvent est de remplacer les caractères $(+1,-1)$ par $(1,-)$ ou encore par $(+,-)$. Cependant, ces trois représentations ont le désavantage de devenir illisibles assez rapidement au regard de la taille de la matrice de Hadamard sélectionnée ; pour pallier à ce problème, on choisit usuellement de représenter les matrices de Hadamard sous forme d'images bicolores (généralement noires et blanches).

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & - & - & - & - \\ 1 & 1 & - & - & 1 & 1 & - & - \\ 1 & 1 & - & - & - & - & 1 & 1 \\ 1 & - & 1 & - & 1 & - & 1 & - \\ 1 & - & 1 & - & - & 1 & - & 1 \\ 1 & - & - & 1 & 1 & - & - & 1 \\ 1 & - & - & 1 & - & 1 & 1 & - \end{bmatrix}$$

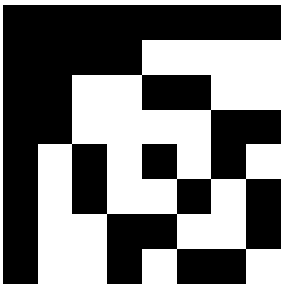


FIGURE 2 – Deux représentations d'une même matrice

II Dénombrement des matrices de Hadamard

Dans cette section, nous nous efforcerons de mettre au point des algorithmes qui, étant donné un entier naturel n non-nul, devrons retourner, s'il en existe, au moins une matrice de Hadamard d'ordre n, voire le nombre de matrices de Hadamard d'ordre n ou encore la liste de ces matrices.

Les codes seront produits dans le langage de programmation Python ; ils ont tous été testés fonctionnels sous Python 3.4.1.

II.1 Algorithme naïf

La première méthode est tout simplement d'explorer entièrement l'ensemble $\mathcal{M}_n(\{-1,+1\})$, puisque ce dernier est fini : il contient 2^{n^2} matrices. Il suffira ensuite de vérifier pour chaque matrice construite si elle est bien de Hadamard.

À cette fin, deux fonctions seront nécessaires :

- Une fonction *hadamard(M)* qui devra retourner Vrai si et seulement si M est une matrice de Hadamard ;
- Une fonction *denombrer(n)*, exploitant la précédente, qui devra retourner la liste des matrices de Hadamard d'ordre n.

II.1.i Algorithme

Au lieu d'implémenter la fonction *hadamard* d'un bloc, on commence par la découper en trois sous-fonctions. On y gagne en lisibilité, et, ainsi, chacune pourra resservir plus tard dans d'autres algorithmes.

On commence par écrire une première fonction qui, étant donnés une matrice carrée M et deux entiers i et j, renvoie le produit scalaires des colonnes i et j de M :

```
def produit_scalaire_colonnes(M,i,j):
    ps = 0
    for k in range(len(M)):
        ps += M[k][i]*M[k][j]
    return ps
```

On écrit ensuite une fonction qui, étant donnée une matrice M, renvoie la matrice correspondant au produit de M par sa transposée. Cette fonction exploite le fait que dans ce produit, le coefficient (i,j) n'est autre que le produit scalaire des colonnes i et j :

```
def produit_transpose(M):
    return [[produit_scalaire_colonnes(M,i,j) for j in range(len(M))]
                for i in range(len(M))]
```

Enfin, on écrit une fonction qui, étant donnés une matrice M et un entier k, renvoie Vrai si et seulement si $M = k * I$, où I est la matrice identité :

```
def homothetie_recursive(M,k):
    def aux(i,j):
        if i>=len(M):
            return True
        if j>=len(M):
            return aux(i+1,0)
        return aux(i,j+1) and (M[i][j]==0 or (i==j and M[i][j]==k))
    return aux(0,0)
```

Dans le corps de cette fonction, les deux premiers tests ne servent qu'à réaliser le parcours de la matrice, tandis que le dernier s'assure que seuls les coefficients diagonaux sont non-nuls, de valeur k.

On peut maintenant assembler les pièces du puzzle pour réaliser la fonction *hadamard* :

```
def hadamard_recursive(M):
    return homothetie_recursive(produit_transpose(M), len(M))
```

On en déduit alors la fonction *denombrer* :

```
def denombrer_matrices_de_Hadamard_naif(n):
    matrices = []
    V = [[0 for i in range(n)] for i in range(n)]

    def aux(i,j,M):
        if i>= n :
            if hadamard_recursive(M):
                matrices.append([l.copy() for l in M])
        elif j>= n:
            aux(i+1,0,M)
        else:
            M[i][j] = 1
            aux(i,j+1,M)
            M[i][j] = -1
            aux(i,j+1,M)

    debut = time.clock()
    aux(0,0,V)
    fin = time.clock()
    return (matrices,fin-debut)
```

Cette fonction va travailler en place sur une matrice V, initialement vide, dont elle va tenter de remplir les cases de gauche à droite et de haut en bas au moyen d'une fonction auxiliaire récursive. Encore une fois, les deux premiers tests de la partie récursive se chargent du parcours de la matrice et de la copie profonde des matrices de Hadamard découvertes, tandis que le dernier sert à lancer, sur chaque coefficient, l'exploration sur les deux valeurs possibles, $+1$ et -1. La fonction principale chronomètre en outre sa propre exécution.

II.1.ii Complexité

Tous les codes précédents étant récursifs, ils sont limités, d'une part, par les capacités de calcul de l'ordinateur hôte, mais également, d'autre part, par la taille de la pile récursive de l'interpréteur Python, qui abandonne après un millier d'appels récursifs.

Pour l'analyse temporelle des algorithmes, on se rapportera à la taille n des matrices carrées traitées ; on supposera que l'accès à un élément donné d'une matrice est réalisé à coût constant.

- Le calcul du produit scalaire de deux colonnes fait appel à n tours de boucle, chacun s'effectuant à coût constant ; son coût est donc en $\Theta(n)$.
- Le calcul du produit d'une matrice par sa transposée fait appel au produit scalaire à travers deux listes en compréhension imbriquées ; son coût est donc en $\Theta(n^3)$.
- La comparaison d'une matrice donnée à une homothétie effectue un nombre constant de tests sur chaque coefficient de la matrice ; son coût est donc en $\Theta(n^2)$. De plus, la pile récursive atteint une taille de $n^2 + n + 1$, donc l'appel n'est réalisable que pour $n < 31$.
- Le test d'appartenance d'une matrice aux matrices de Hadamard fait appel à un calcul de la taille de sa première ligne, de coût linéaire, et aux fonctions de produit par la transposée et de comparaison à une homothétie ; son coût est donc en $\Theta(n^3)$, le produit par la transposée l'emportant.
- Enfin, la fonction de dénombrement des matrices de Hadamard appelle la fonction précédente sur l'ensemble des matrices de $\mathcal{M}_n(\{-1, +1\})$; son coût est donc en $\Theta(n^3 * 2^{n^2})$. De plus, la pile récursive atteint une taille de $2n(n + 1)$, donc l'appel n'est réalisable que pour $n < 22$.

Les problèmes de limitation de la pile récursive peuvent être contournés en implémentant manuellement cette pile pour la masquer à l'interpréteur ; cependant, cela n'a que peu d'intérêt à ce stade, au vu des complexités temporelles qui sont bien plus limitantes.

II.1.iii Temps mesurés

- Pour $n = 1$, le programme trouve les 2 matrices de Hadamard en une trentaine de micro-secondes en moyenne.
- Pour $n = 2$, le programme trouve les 8 matrices de Hadamard en 500μs en moyenne.
- Pour $n = 4$, le programme trouve les 768 matrices de Hadamard en 3,7s en moyenne.
- Pour $n = 8$, la durée théorique de calcul tournerait autour de 10^{15}s, soit près de 27 millions d'années !

II.2 Algorithme moins naïf

Toute l'idée de ce nouvel algorithme repose sur la remarque déjà soulevée lors de l'écriture de la fonction de multiplication d'une matrice par sa transposée : dans ce produit, le coefficient (i, j) n'est autre que le produit scalaire des colonnes i et j.

Ceci ouvre la voie à une méthode dite de *backtracking*, dont le principe sera le suivant : plutôt que calculer le produit d'une matrice complète par sa transposée, nous allons construire une matrice de -1 et de $+1$ colonne par colonne, en revenant sur nos pas à chaque fois que les colonnes ainsi formées ne forment plus une famille orthogonale. À la fin, nous aurons donc calculé exactement autant de produits scalaires que dans la méthode précédente pour chacune des matrices de Hadamard trouvée, mais moins pour les autres.

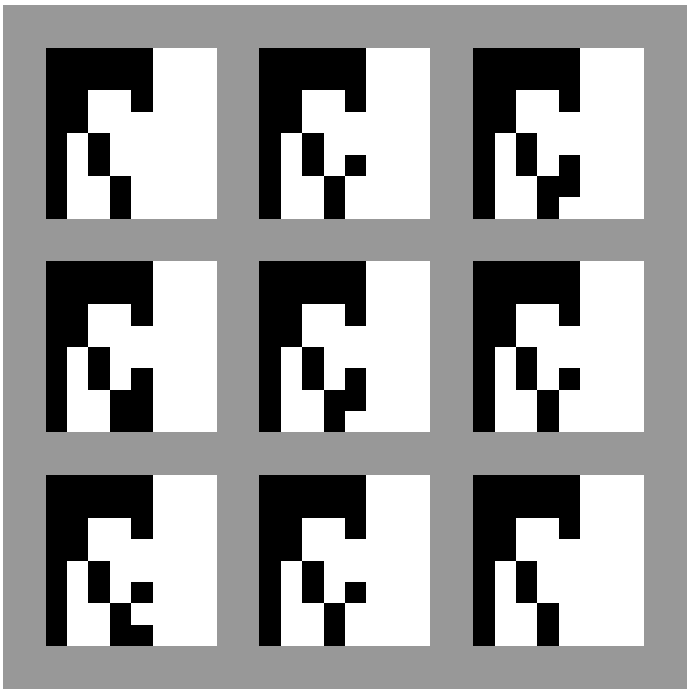


FIGURE 3 – Le *backtracking* en action

II.2.i Algorithme

Le code sera cette fois-ci découpé en deux fonctions distinctes.

La première, qui fait appel à la fonction produit scalaire déjà écrite, prend en argument une matrice M et un indice i, et renvoie Vrai si et seulement si la i-ème colonne de M est orthogonale aux $i-1$ précédentes.

```
def colonne_orthogonale_aux_precedentes(M,i):
    res,k = True,0
    while res == True and k < i:
        res = (produit_scalaire_colonnes(M,k,i)==0)
        k+=1
    return res
```

La seconde reprend le principe de la construction coefficient-à-coefficient de matrices de Hadamard de l'algorithme naïf, mais en appelant une vérification à chaque fin de colonne.

```
def denombrer_matrices_de_Hadamard_backtracking(n):
    matrices = []
    V = [[0 for i in range(n)] for i in range(n)]

    def aux(i,j,M):
        if j>= n :
            matrices.append([l.copy() for l in M])
        elif i>= n:
            if colonne_orthogonale_aux_precedentes(M,j):
                aux(0,j+1,M)
        else:
            M[i][j] = 1
            aux(i+1,j,M)
            M[i][j] = -1
            aux(i+1,j,M)

    debut = time.clock()
    aux(0,0,V)
    fin = time.clock()
    return (matrices,fin-debut)
```

On retrouve la même structure de parcours récursif que précédemment, mais l'emplacement des tests a changé : on en effectue à chaque fin de colonne, mais ils ne sont plus nécessaires en fin de matrice.

Enfin, on peut s'intéresser au dénombrement seul des matrices, sans mémorisation :

```
def denombrer_matrices_de_Hadamard_backtracking_compteur(n):
    V = [[0 for i in range(n)] for i in range(n)]
    l = [0]

    def aux(i,j,M):
        if j>= n :
            l[0]+=1
            print(l[0])
        elif i>= n:
            if colonne_orthogonale_aux_precedentes(M,j):
                aux(0,j+1,M)
        else:
            M[i][j] = 1
            aux(i+1,j,M)
            M[i][j] = -1
            aux(i+1,j,M)

    debut = time.clock()
    aux(0,0,V)
    fin = time.clock()
```

Dans tous les cas, les différentes fonctions continuent à chronométrer leur propre exécution.

II.2.ii Complexité

L'étude de la complexité des algorithmes est cette fois-ci un peu plus délicate, dans la mesure où il est difficile d'estimer le gain par rapport aux algorithmes naïfs, en l'absence de connaissances sur la répartition des matrices de Hadamard dans $\mathcal{M}_n(\{-1,+1\})$; la comparaison s'effectuera principalement au niveau du temps d'exécution. On peut toutefois remarquer les points suivants :

- La fonction de test d'orthogonalité de la colonne i calcule son produit scalaire avec les $i-1$ précédentes : son coût est donc en $\Theta(n*(i-1))$.
- Lors de la construction d'une matrice de Hadamard, parvenir jusqu'à la colonne i nécessite d'avoir effectué ce test sur les i premières colonnes : le coût de cette opération est donc en $\Theta(n*i^2)$. On ne peut guère en dire beaucoup plus sans savoir combien de matrices parviendront jusqu'à la colonne i sans être abandonnées. En revanche, la pile récursive est cette fois-ci de taille $n*(n+1)$, donc l'appel à cette fonction est possible pour $n < 32$.
- Enfin, pour une matrice abandonnée par cette algorithme après examen de la colonne i, l'algorithme précédent aurait continué à remplir les $n*(n-i)$ coefficients restants, pour un coût en $\Theta(n^3 * 2^{n*(n-i)})$.

II.2.iii Temps mesurés

- Pour $n = 1$, le programme trouve les 2 matrices de Hadamard en une 10μs, qu'il les stocke ou non.
- Pour $n = 2$, le programme trouve les 8 matrices de Hadamard en 80μs, qu'il les stocke ou non.
- Pour $n = 4$, le programme trouve les 768 matrices de Hadamard en 48ms s'il les stocke, en 45ms sinon.
- Pour $n = 8$, on ne peut plus trancher par une estimation du temps de calcul théoriquement nécessaire. En revanche, on peut signaler qu'au bout de deux jours de calcul, plus de 250 millions de matrices de Hadamard d'ordre 8 ont été découvertes par l'algorithme, sans terminer.

En conclusion, malgré une accélération notable obtenue par cette méthode, le nombre de matrices à analyser croît toujours bien trop vite pour la puissance accessible à un ordinateur standard ; on constate en outre que ce n'est pas la mémorisation des résultats qui pénalisait le temps d'exécution, la différence étant à peine perceptible.

II.3 À la recherche d'une bonne relation d'équivalence

Puisqu'il est manifestement déraisonnable de dénombrer toutes les matrices de Hadamard d'un ordre donné, on peut essayer de les chercher sous une forme particulière, pour déduire toutes les matrices de Hadamard de taille donnée à partir d'une part restreinte d'entre elles.

II.3.i Opérations élémentaires

Comme cela a déjà été souligné en I.2 (page 5), pour n fixé, l'ensemble $\mathcal{H}_n$ est stable pour un certain nombre d'opérations, que nous allons nous efforcer de traduire matriciellement.

Pour $n \in \mathbb{N}^*$, on notera $\mathfrak{S}_n$ l'ensemble des permutations de $[\![1, n]\!]$, et $\mathcal{E}_n$ l'ensemble des fonctions de $[\![1, n]\!]$ dans $\{-1, +1\}$.

Enfin, pour $(\sigma, \epsilon) \in \mathfrak{S}_n \times \mathcal{E}_n$, on pose $P_{\sigma,\epsilon} = (\epsilon(i)\delta_{i,\sigma(j)})_{1\leq i,j\leq n}$; les matrices $P_{\sigma,\epsilon}$ pour $(\sigma, \epsilon) \in \mathfrak{S}_n \times \mathcal{E}_n$ formeront l'ensemble des *matrices d'opération élémentaire*, au sens où multiplier à gauche (respectivement à droite) une matrice de Hadamard par une matrice d'opération élémentaire revient à permuter ses lignes (respectivement ses colonnes) en les multipliant éventuellement par -1, ce qui donne une nouvelle matrice de Hadamard.

On notera que toutes ces matrices d'opération élémentaire sont inversibles, et de déterminant ± 1.

II.3.ii Matrices normalisées

Étant donnée une matrice de Hadamard de rang n, quitte à multiplier certaines colonnes par -1, on peut toujours se ramener à une matrice de Hadamard dont la première ligne ne contient que des 1. De même, on peut ramener la première colonne à 1. Une telle matrice sera dite *normalisée*. Le dénombrement de telles matrices normalisées se fait de manière similaire aux algorithmes précédents :

```
def denombrer_normalisees(n):
    l = [0]
    V = [[0 for i in range(n)] for i in range(n)]
    for k in range(n):
        V[k][0] = 1
        V[0][k] = 1

    def aux(i,j,M):
        if j>= n :
            l[0]+=1
        elif i>= n:
            if colonne_orthogonale_aux_precedentes(M,j):
                aux(1,j+1,M)
        else:
            M[i][j] = 1
            aux(i+1,j,M)
            M[i][j] = -1
            aux(i+1,j,M)

    debut = time.clock()
    aux(1,1,V)
    fin = time.clock()
```

En dimension 1 et 2, il n'existe qu'une seule matrice de Hadamard normalisée. En dimension 4, sur les 768 matrices de Hadamard trouvées, seules 6 sont normalisées. Enfin, en dimension 8, où le dénombrement complet n'aboutit pas, on trouve 151200 matrices de Hadamard normalisées.

On remarque que, s'il s'agit de la définition usuelle des matrices de Hadamard normalisées, on pourrait pousser les choses un peu plus loin : par exemple, par nullité du produit scalaire des deux premières lignes et colonnes, on peut se ramener à une matrice dont la deuxième ligne et la deuxième colonne commencent par une moitié de 1.

De même, on peut imposer que la troisième ligne et la troisième colonne alternent entre 1 et -1 par quart ; c'est par contre la forme la plus restrictive dans le cas général, c'est-à-dire lorsque n est divisible par 4 mais pas par 8. On passe ainsi à une matrice en dimension 1, 2 et 4, huit en dimension 8 et 15552 en dimension 12 ; on accède ainsi au premier ordre accessible qui n'est pas une puissance de 2.

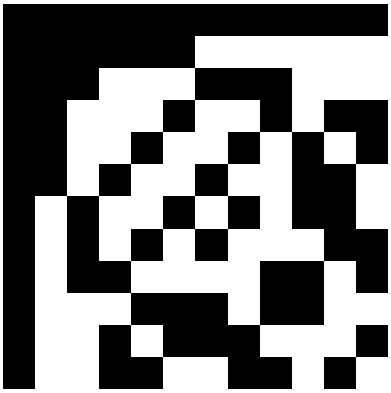


FIGURE 4 – Une matrice de Hadamard normalisée d'ordre 12

II.3.iii Cardinal des classes d'équivalence

Une première relation La relation d'équivalence la plus simple consiste à regrouper ensemble les matrices qui ne diffèrent que par des opérations élémentaires sur les lignes :

$$A \overset{l}{\sim} B \Leftrightarrow \exists(\sigma, \epsilon) \in \mathfrak{S}_n \times \mathcal{E}_n | A = P_{\sigma,\epsilon} B$$

Les matrices d'opérations élémentaires étant inversibles, le cardinal des classes d'équivalence associées à cette relation vaut $\mid \mathfrak{S}_n \mid * \mid \mathcal{E}_n \mid = n! * 2^n$. Il n'y a donc qu'une classe d'équivalence en dimension 1 et 2, et deux en dimension 4. En dimension 8, chaque classe est de taille 10321920 ; comme on avait trouvé plus de 250 millions de matrices avant d'abandonner, il y a au moins 24 classes d'équivalences.

Une relation plus poussée On peut ensuite se demander ce qu'il advient si l'on s'autorise, en plus des opérations élémentaires sur les lignes, des opérations élémentaires sur les colonnes :

$$A \overset{lc}{\sim} B \Leftrightarrow \exists(\sigma_1, \epsilon_1), (\sigma_2, \epsilon_2) \in \mathfrak{S}_n \times \mathcal{E}_n | A = P_{\sigma_1,\epsilon_1} B P_{\sigma_2,\epsilon_2}$$

Le problème est qu'il ne suffit plus de dénombrer les combinaisons possibles : en effet, pour $(\sigma_1, \epsilon_1), (\sigma_2, \epsilon_2), (\sigma_1', \epsilon_1'), (\sigma_2', \epsilon_2') \in \mathfrak{S}_n \times \mathcal{E}_n$:
$P_{\sigma_1,\epsilon_1} B P_{\sigma_2,\epsilon_2} = P_{\sigma_1',\epsilon_1'} B P_{\sigma_2',\epsilon_2'} \nRightarrow P_{\sigma_1,\epsilon_1} = P_{\sigma_1',\epsilon_1'}$ et $P_{\sigma_2,\epsilon_2} = P_{\sigma_2',\epsilon_2'}$

En particulier, dès $n = 4$, il y a 147456 couples de matrices d'opérations élémentaires, pour seulement 768 matrices de Hadamard.

Pour aller plus loin Il existe une dernière relation d'équivalence, usuellement appelée *relation de ressemblance* :

$$A \overset{r}{\sim} B \Leftrightarrow A \overset{lc}{\sim} B \text{ ou } {}^tA \overset{lc}{\sim} B.$$

Ceci revient en fait à s'autoriser à regrouper les classes d'équivalence précédentes par paires. La relation de ressemblance est intéressante lorsque l'on cherche à étudier les matrices de Hadamard par l'intermédiaire de leur permanent, dont la valeur absolue est constante à l'intérieur d'une classe de ressemblance ; voir [4] à ce sujet.

Dans tous les cas, il reste à trouver, pour chaque relation d'équivalence, un moyen de créer un ensemble de représentants ; ou *a minima* de déterminer si deux matrices données sont équivalentes, en un temps raisonnables. Il n'y a *a priori* pas de réponse simple à cette problématique.

III Construction de matrices de Hadamard

III.1 Recherche d'une matrice d'ordre donné

Au vu des temps colossaux mis en jeu lors du dénombrement des matrices de Hadamard, on peut abandonner cette perspective pour, à la place, se tourner vers la recherche d'une matrice de Hadamard d'ordre donné.

III.1.i Algorithme

Ici, rien de bien nouveau : il suffit de reprendre l'algorithme le plus performant trouvé pour l'instant [1], en nous contentant de le modifier pour qu'il s'arrête à la première matrice de Hadamard trouvée. Il ne la retournera par ailleurs pas, se contentant d'indiquer si elle existe (dans une optique de vérification de la conjecture, par exemple).

```
def existe_matrice_de_Hadamard(n):
    V = [[0 for i in range(n)] for i in range(n)]

    def aux(i,j,M):
        if j>= n :
            return True
        elif i>= n:
            if colonne_orthogonale_aux_precedentes(M,j):
                return aux(0,j+1,M)
        else:
            M[i][j] = 1
            M1 = aux(i+1,j,M)
            if M1 != None:
                return M1
            else:
                M[i][j] = -1
                return aux(i+1,j,M)
    debut = time.clock()
    R = aux(0,0,V)
    fin = time.clock()
```

On retrouve bien ici la même structure que précédemment : un parcours récursif de la matrice qui s'appelle sur chaque coefficient. L'unique différence est qu'avant de passer d'une possibilité à la suivante, on vérifie qu'aucune

1. voir page 13

matrice de Hadamard n'a été trouvée. Encore une fois, le tout est chronométré.

III.1.ii Temps mesurés

- Pour $n = 1$, le programme termine en 5μs.
- Pour $n = 2$, le programme termine en 20μs.
- Pour $n = 4$, le programme termine en 90μs.
- Pour $n = 8$, le programme termine en 450μs.
- Pour $n = 12$, le programme termine en 130ms.
- Pour $n = 16$, le programme termine en 2,2s.
- Pour $n = 20$, le programme n'a pas terminé au bout de cinq heures de calcul. On accède bien ainsi à des dimensions supérieures à ce qui était possible en dénombrement ; toutefois, cela reste encore bien limité, et est loin de fournir une méthode efficace de vérification de la conjecture pour des ordres importants.

III.2 Matrices circulantes

Même la recherche d'une seule matrice de Hadamard se révélant trop complexe, il faut se tourner vers des méthodes de génération rapide de matrices de Hadamard ; une piste intéressante est celle des *matrices circulantes*.

III.2.i Définition

Soit $A \in \mathcal{M}_n(\mathbb{C})$; A est dit *circulante* si ses lignes sont toutes obtenues par permutation circulaire de la première, dans un sens ou dans l'autre. Plus précisément, en notant $\sigma \in \mathfrak{S}_n$ le cycle $(1 \dots n)$:

A est *circulante à gauche* si $\forall i, j \in [\![1, n]\!], a_{i,j} = a_{1,\sigma^{i-1}(j)}$

A est *circulante à droite* si $\forall i, j \in [\![1, n]\!], a_{i,j} = a_{1,\sigma^{1-i}(j)}$

Par exemple, $\begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 0 \\ 2 & 0 & 1 \end{bmatrix}$ est circulante à gauche, tandis que $\begin{bmatrix} 0 & 1 & 2 \\ 2 & 0 & 1 \\ 1 & 2 & 0 \end{bmatrix}$ est circulante à droite. L'intérêt des matrices circulantes est qu'elles sont entièrement déterminées par la donnée de n coefficients, et non n^2.

Exemple plus important, on dispose d'une matrice de Hadamard circulante à droite d'ordre 4 :

$$\begin{bmatrix} -1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 \end{bmatrix}$$

III.2.ii Conjecture

La recherche de matrices de Hadamard circulante est très simple à mettre en oeuvre.

On commence par se donner une fonction *circulante* qui, à une ligne donnée, associe la matrice circulante à gauche correspondante :

```
def circulante(l):
    n = len(l)
    c = lambda i : (i + 1) % n
    def aux(i,u,res):
        if i>=n: return res
        v = [u[c(k)] for k in range(n)]
        res.append(v)
        return aux(i+1,v,res)
    return aux(1,l,[l])
```

Il suffit ensuite de parcourir, pour n donné, toutes les lignes de -1 et 1 de taille n, en mémorisant celles qui engendrent une matrice de Hadamard circulante à gauche :

```
def hadamard_circulante(n):
    lignes = []
    L0 = [0 for i in range(n)]
    def aux(i,L):
        if i>=n:
            if hadamard_recursive(circulante(L)):
                lignes.append(L.copy())
        else:
            L[i]=1
            aux(i+1,L)
            L[i]=-1
            aux(i+1,L)
    aux(0,L0)
    return lignes
```

Soulignons que le dénombrement des matrices de Hadamard circulantes d'ordre n présente une complexité temporelle en $\Theta(n^3 * 2^n)$, quand l'algorithme de dénombrement des matrices de Hadamard en II.1 avait une complexité temporelle en $\Theta(n^3 * 2^{n^2})$.

Malheureusement, en explorant quelques ordres accessibles, la matrice de Hadamard circulante d'ordre 4 exhibée plus haut fait figure d'exception : on trouve ainsi deux matrices de Hadamard circulantes d'ordre 1, aucune d'ordre 2, huit d'ordre 4, puis aucune d'ordre 8, 12, 16, 20 ou 24. Cela nous mène à la conjecture suivante :

Conjecture. *Pour $n > 4$, il n'existe aucune matrice de Hadamard circulante à gauche d'ordre n.*

Remarquons qu'il suffit de chercher des matrices de Hadamard circulantes à gauche : en effet, étant donnée une matrice de Hadamard circulante à droite, on forme une matrice de Hadamard circulante à gauche en lisant ses lignes de bas en haut.

Informatiquement, aucun contre-exemple n'a été trouvé jusqu'à des ordres voisins de 100000.

III.2.iii S-matrices

S'il apparaît irréalisable de chercher des matrices de Hadamard circulante, en leur faisant subir une transformation simple et aisément réversible, on peut par contre obtenir une famille de matrices contenant des matrices circulantes ; de plus, on peut exhiber un procédé de construction de telles matrices circulantes d'ordre arbitrairement grand. On part pour cela d'une matrice de Hadamard normalisée, dont on ôte la première ligne et la première colonne ; après quoi, on remplace les 1 par des 0 et les -1 par des 1.

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & - & - \\ 1 & - & 1 & - \\ 1 & - & - & 1 \end{bmatrix}$$

$$\downarrow$$

$$\begin{bmatrix} 1 & - & - \\ - & 1 & - \\ - & - & 1 \end{bmatrix}$$

$$\downarrow$$

$$\begin{bmatrix} 0 & - & - \\ - & 0 & - \\ - & - & 0 \end{bmatrix}$$

$$\downarrow$$

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

III.2.iv Polynômes binaires primitifs

Un polynôme P, à coefficients dans $\mathbb{Z}/2\mathbb{Z}$, de degré n, est dit primitif si et seulement si, d'une part, il est irréductible et, d'autre, le plus petit m tel que P divise $X^m - 1$ est $m = 2^n - 1$. Il existe des algorithmes relativement peu coûteux de génération de tels polynômes, permettant d'en construire des tables pour des degrés allant jusqu'à quelques centaines [2].

Exemple : $X^{168} + X^{17} + X^{15} + X^2 + 1$.

Considérons maintenant une suite récurrente linéaire d'ordre p sur $\mathbb{Z}/2\mathbb{Z}$; on montre qu'elle est nécessairement périodique, de période au plus $2^p - 1$, et que cette périodicité maximale est atteinte si et seulement si son polynôme caractéristique, de degré p, est primitif ; enfin, le cas échéant, les $2^p - 1$ valeurs formées par n'importe quelle période forment la première ligne d'une S-matrice circulante d'ordre $2^p - 1$.

2. voir [3]

III.3 La construction de Sylvester

Cette construction, due à Sylvester, exploite les propriétés des matrices de Hadamard pour en construire d'ordre arbitrairement grand (mais pas quelconque) à partir de la connaissance d'au moins une d'ordre supérieur ou égal à 2.

Concernant le *produit de Kronecker* de deux matrices, on reprend la définition et les notations développés en V.2, notamment en ce qui concerne les fonctions ϕ_n et ψ_n.

La construction de Sylvester repose sur le théorème suivant :

Théorème III.3.1. *Soit $A \in \mathcal{H}_n$, soit $B \in \mathcal{H}_m$; alors $A \otimes B \in \mathcal{H}_{n\times m}$*

Démonstration. Étant donnée une matrice M quelconque, on notera $L_i(M)$ la i-ème ligne de M.

Pour $A \in \mathcal{H}_n$ et $B \in \mathcal{H}_m$, considérons les lignes i et j de $A \otimes B$.

$L_i(A\otimes B) = L_{\phi_m(i)}(A)\otimes L_{\psi_m(i)}(B)$, et $L_j(A\otimes B) = L_{\phi_m(j)}(A)\otimes L_{\psi_m(j)}(B)$.Alors :

$$< L_i(A\otimes B), L_j(A\otimes B) > = \sum_{k=1}^{n} a_{\phi_m(i),k} * a_{\phi_m(j),k} < L_{\psi_m(i)}(B), L_{\psi_m(j)}(B) >$$

$$< L_i(A\otimes B), L_j(A\otimes B) > =< L_{\phi_m(i)}(A), L_{\phi_m(j)}(A) > * < L_{\psi_m(i)}(B), L_{\psi_m(j)}(B) >$$

$$< L_i(A\otimes B), L_j(A\otimes B) > = \delta_{\phi_m(i),\phi_m(j)} n * \delta_{\psi_m(i),\psi_m(j)} m = \delta_{i,j} n * m$$

□

En particulier, on posera $S_n = H_2^n$, où $H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$, et où l'exposant représente l'itérée pour le produit de Kronecker.

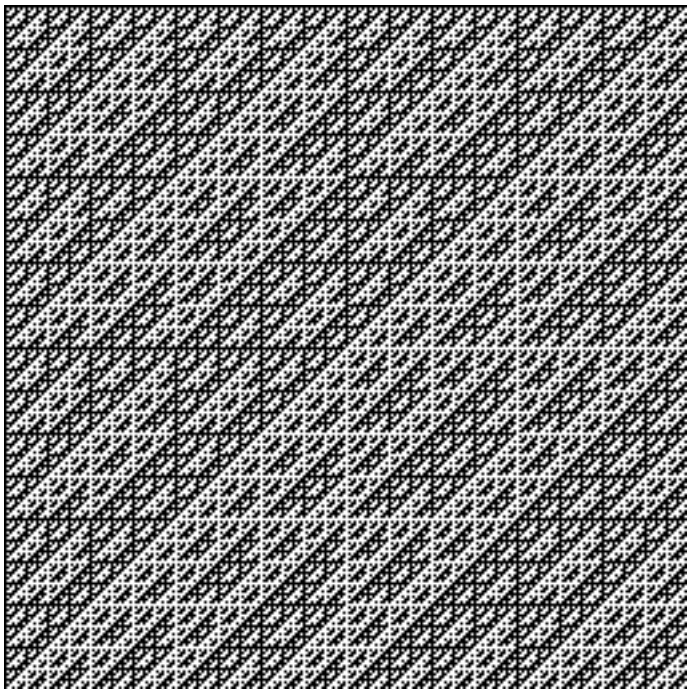


FIGURE 5 – La matrice S256

IV Fonctions de Walsh

L'un des intérêts principaux des matrices de Hadamard réside dans leur application en traitement du signal, que nous présentons ici.

IV.1 Définition

Les *fonctions de Walsh* sont une famille de fonctions constantes par morceaux, définies sur $[0,1]$ et à valeurs dans $\{-1,+1\}$.

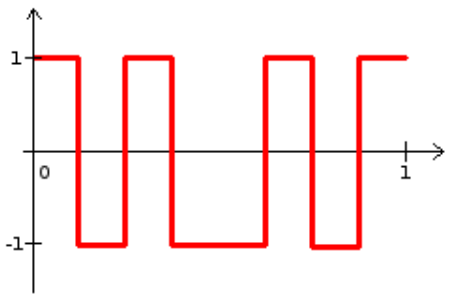
Soient $j \in \mathbb{N}^*$, et k l'entier correspondant à $j-1$ en binaire réfléchi[3] ; k se décompose sous la forme : $k = \sum_{i=0}^{m} k_i 2^i$, où les k_i valent tous 0 ou 1.
Soit maintenant $x \in [0,1[$; x se décompose lui sous la forme $x = \sum_{i=0}^{+\infty} x_i 2^{-i-1}$, où les x_i valent tous 0 ou 1.
La j-ième fonction de Walsh vaut alors $(-1)^{k_0 x_0 + \ldots + k_m x_m}$ en x.

IV.2 Lien avec les matrices de Hadamard

Considérons une matrice de Hadamard M, d'ordre n donné. Le cas le plus courant est celui où n est une puissance de 2 : la construction de Sylvester[4] nous fournit un moyen rapide de générer de telles matrices. Posons $n = 2^r$.
Les lignes de M forment une famille de 2^r vecteurs orthogonaux de $\mathcal{M}_{1,n}(\{-1,+1\})$; en divisant le segment $[0,1]$ en 2^r segments de longueurs 2^{-r}, on associe à chaque ligne une fonction de Walsh, dont la valeur sur le i-ème segment, pour $i \in [\![1, 2^r]\!]$ est celle du coefficient correspond dans M.
Ainsi, la ligne :

$$\begin{bmatrix} +1 & -1 & +1 & -1 & -1 & +1 & -1 & +1 \end{bmatrix}$$

est associée à la fonction :



3. voir [5]
4. voir III.3 page 27

IV.3 Applications

L'ensemble des fonctions de Walsh forme une base hilbertienne de l'espace des fonctions de carré intégrable sur $[0, 1]$: on peut donc construire à partir des fonctions de Walsh une transformée, dite *transformée de Walsh-Hadamard*, similaire à la transformée de Fourier, mais où une fonction donnée n'est plus caractérisée par son produit scalaire avec des sinusoïdes mais avec des fonctions de Walsh.

Dans le cas où la fonction en question est un signal discret à traiter, cela revient donc à effectuer un produit matriciel avec une matrice de Hadamard, opération très simple à inverser.

Physiquement, cette transformée est simple à interpréter, par exemple en optique : dans le cas d'un signal constitué de plusieurs composantes (spectrales ou spatiales), on peut choisir d'envoyer le rayonnement à étudier sur un dispositif semi-réfléchissant, qui laissera passer certaines composantes directement vers le capteur, et fera transiter les autres par un miroir intermédiaire. Une fois le signal recombiné au niveau du capteur, cela revient à sommer les composantes sur la voie transmise et à retrancher celles sur la voie réfléchie (en raison du déphasage induit par le procédé). Le cas des S-matrices correspond lui à un dispositif qui, au lieu de transmettre ou réfléchir sélectivement, transmet ou absorbe.

Cette idée de mesurer des combinaisons linéaires des composantes du signal à étudier est développée dans une discipline appelée *weighing design*, où les matrices de Hadamard et les S-matrices se révèlent justement optimales sous certaines conditions : les matrices de Hadamard maximisent le rapport signal sur bruit quand les seuls coefficients utilisés sont -1 et +1, et on conjecture que les S-matrices le maximisent pour 0 et 1[5].

5. voir [1]

Grandeurs : $(\gamma_1 \dots \gamma_n) \in \mathbb{R}^n$.
Mesures : $(\phi_1 \dots \phi_n) \in \mathbb{R}^n$.
Erreurs : $(\delta_1 \dots \delta_n) \in \mathbb{R}^n$, indépendantes, espérance nulle, écart-type σ.
Weighing design : $W \in GL_n(\mathbb{R})$.

$$\Phi = W\Gamma + \Delta \Rightarrow \widetilde{\Gamma} = \Gamma + W^{-1}\Delta$$
$$\mathbb{E}(\widetilde{\Gamma} - \Gamma) = W^{-1}\mathbb{E}(\Delta) = 0$$
$$\mathbb{E}((\widetilde{\gamma_i} - \gamma_i)^2) = \mathbb{E}([W^{-1}\Delta]_i)$$

Méthode naïve : $W = I_n \Rightarrow \mathbb{E}((\widetilde{\gamma_i} - \gamma_i)^2) = \sigma^2$

Théorème IV.3.1. $W \in \mathcal{M}_n(\{-1, +1\}) \Rightarrow \mathbb{E}((\widetilde{\gamma_i} - \gamma_i)^2) \geq \sigma^2/n$

Théorème IV.3.2. $W \in \mathcal{H}_n \Rightarrow \mathbb{E}((\widetilde{\gamma_i} - \gamma_i)^2) = \sigma^2/n$

V Annexes

V.1 Intérêt historique des matrices de Hadamard

L'un des problèmes sur lesquels travaillait Hadamard était de borner au mieux le déterminant d'une matrice, connaissant une borne sur la valeur de ses coefficients ; en particulier, lorsque ceux-ci sont compris entre -1 et 1, cas auquel on peut toujours se ramener. Le résultat principal est le suivant :

Théorème V.1.1 (Inégalité de Hadamard)**.** *Soient n un entier naturel, A une matrice carrée d'ordre n à coefficients dans $[-1,1]$ et $C_1(A)\dots C_n(A)$ les colonnes de A ; on munit l'espace des colonnes de son produit scalaire canonique. Alors :*

$$\mid det(A) \mid \leq \prod_{k=1}^{n} \parallel C_k(A) \parallel$$

Démonstration. Si A n'est pas inversible, le résultat est immédiat, car son déterminant s'annule. Dans le cas contraire, notons $U_1\dots U_n$ la famille orthogonale de colonnes obtenue à partir de $C_1(A)\dots C_n(A)$ par le procédé de Gram-Schmidt, et B la matrice formée par cette famille. Le procédé de Gram-Schmidt se traduit matriciellement par une multiplication à droite par une matrice triangulaire supérieure P dont la diagonale n'est composée que de 1 : $B = AP$, d'où $det(A) = det(B)$.

Le déterminant de B est lui très simple à calculer. En effet, en remarquant que ${}^tBB = (< C_i(B), C_j(B) >)_{1\leq i,j\leq n} = (< U_i, U_j >)_{1\leq i,j\leq n}$, l'orthogonalité des colonnes de B se traduit par : $det({}^tBB) = det^2(B) = \prod_{k=1}^{n} \parallel U_k \parallel^2$.

Enfin, lors de l'orthogonalisation de la famille des colonnes de A, la colonne U_i est obtenue par projection orthogonale de la colonne $C_i(A)$, donc est de norme inférieure. En conclusion :

$$\mid det(A) \mid = \mid det(B) \mid = \prod_{k=1}^{n} \parallel U_k \parallel \leq \prod_{k=1}^{n} \parallel C_k(A) \parallel$$

□

En conséquence, si les coefficients de A sont dans $[-1,1]$, chaque colonne est de norme au plus n, donc $\mid det(A) \mid \leq n^{n/2}$. Si A est une matrice de Hadamard :

$${}^tAA = nI_n \Rightarrow det({}^tAA) = det^2(A) = n^n \Rightarrow \mid det(A) \mid = n^{n/2}$$

Les matrices de Hadamard prouvent donc que l'inégalité du même nom est optimale dès que $\mathcal{H}_n \neq \emptyset$, soit, si la conjecture est vérifiée, pour $n = 1$, $n = 2$ et $n \in 4\mathbb{N}$. Elle peut en revanche être améliorée pour d'autres valeurs de n : voir [7] à ce sujet.

V.2 Produit de Kronecker

On se donne deux matrices $A = (a_{i,j})_{\substack{1\leq i\leq m\\1\leq j\leq n}}$ et $B = (b_{i,j})_{\substack{1\leq i\leq s\\1\leq j\leq t}}$; le *produit de Kronecker* de A et B est défini par bloc par :

$$A \otimes B = (a_{i,j}B)_{\substack{1\leq i\leq m\\1\leq j\leq n}} \in \mathcal{M}_{ms,nt}(\mathbb{C})$$

Par exemple :

$$\begin{bmatrix}1 & 2\end{bmatrix} \otimes \begin{bmatrix}0 & 1\\-1 & 0\end{bmatrix} = \begin{bmatrix}0 & 1 & 0 & 2\\-1 & 0 & -2 & 0\end{bmatrix}$$

V.2.i Notations

Pour toute matrice M, on notera $M_{i,j}$ son coefficient d'indice (i,j).

La première difficulté est de pouvoir expliciter facilement le coefficient d'indice (i,j) d'un produit de Kronecker, en fonction des coefficients de ses deux termes : intuitivement, il faut "compter combien de blocs B sont passés", et savoir "où l'on en est dans le bloc en cours".

Pour cela, pour $n \in \mathbb{N}^*$, on définit récursivement deux fonctions ϕ_n et ψ_n par :

$$\begin{cases} \forall i \in \mathbb{N}, \phi_n(i) = (i-1)//n + 1\\ \forall i \in \mathbb{N}, \psi_n(i) = (i-1)\%n + 1\end{cases}$$

où la division euclidienne de i par n est notée : $i = (i//n)n + (i\%n)$.

Pour $A \in \mathcal{M}_{m,n}(\mathbb{C})$ et $B \in \mathcal{M}_{s,t}(\mathbb{C})$, le coefficient d'indice (i,j) de $A \otimes B$ vaut alors $a_{\phi_s(i),\phi_t(j)}b_{\psi_s(i),\psi_t(j)}$.

V.2.ii Propriétés

En plus de la propriété exploitée en III.3 (page 27), on dispose de plusieurs résultats sur le produit de Kronecker, permettant de le manipuler plus facilement.

Propriété V.2.ii.1 (Bilinéarité)**.** *Soient A, B et C trois matrices, soit k un scalaire ; alors, si les dimensions sont compatibles :*

$$\begin{aligned} A \otimes (B + C) &= A \otimes B + A \otimes C\\ (A + B) \otimes C &= A \otimes C + B \otimes C\\ (kA) \otimes B &= kA \otimes B\\ A \otimes (kB) &= kA \otimes B \end{aligned}$$

Propriété V.2.ii.2 (Associativité). *Soient A, B et C trois matrices ; alors :*

$$(A \otimes B) \otimes C = A \otimes (B \otimes C)$$

Démonstration. Considérons $A \in \mathcal{M}_{n,m}(\mathbb{C})$, $B \in \mathcal{M}_{r,s}(\mathbb{C})$ et $C \in \mathcal{M}_{p,q}(\mathbb{C})$. La bilinéarité du produit de Kronecker permet de se contenter d'examiner le cas où A, B et C sont des matrices élémentaires, *i.e.* où A est nulle sauf en (a_0, a'_0), B est nulle sauf en (b_0, b'_0) et C est nulle sauf en (c_0, c'_0).

Alors $A \otimes B$ est nulle, sauf en $\Big((a_0-1)r+b_0, (a'_0-1)s+b'_0\Big)$, donc $(A \otimes B) \otimes C$ est nulle sauf en $\Big(\big((a_0-1)r+b_0-1\big)p+c_0, \big((a'_0-1)s+b'_0-1\big)q+c'_0\Big)$.

De même, $B \otimes C$ est nulle, sauf en $\Big((b_0-1)p+c_0, (b'_0-1)q+c'_0\Big)$, donc $A \otimes (B \otimes C)$ est nulle sauf en $\Big((a_0-1)rp+(b_0-1)p+c_0, (a'_0-1)sq+(b'_0-1)q+c'_0\Big)$.

□

Propriété V.2.ii.3 (Produit mixte). *Soient A, B, C et D quatre matrices ; alors, si les dimensions sont compatibles :*

$$(A \otimes B)(C \otimes D) = (AC) \otimes (BD)$$

Démonstration. Considérons $A \in \mathcal{M}_{n,m}(\mathbb{C})$, $B \in \mathcal{M}_{r,s}(\mathbb{C})$, $C \in \mathcal{M}_{m,p}(\mathbb{C})$ et $D \in \mathcal{M}_{s,t}(\mathbb{C})$; $AC \in \mathcal{M}_{n,p}(\mathbb{C})$, $BD \in \mathcal{M}_{r,t}(\mathbb{C})$, $A \otimes B \in \mathcal{M}_{nr,ms}(\mathbb{C})$, $C \otimes D \in \mathcal{M}_{ms,pt}(\mathbb{C})$.

$$\Big((A \otimes B)(C \otimes D)\Big)_{i,j} = \sum_{k=1}^{ms} (A \otimes B)_{i,k} (C \otimes D)_{k,j}$$

$$\Big((A \otimes B)(C \otimes D)\Big)_{i,j} = \sum_{k=1}^{ms} A_{\phi_r(i),\phi_s(k)} B_{\psi_r(i),\psi_s(k)} C_{\phi_s(k),\phi_t(j)} D_{\psi_s(k),\psi_t(j)}$$

$$\Big((A \otimes B)(C \otimes D)\Big)_{i,j} = \sum_{k=1}^{ms} A_{\phi_r(i),\phi_s(k)} C_{\phi_s(k),\phi_t(j)} B_{\psi_r(i),\psi_s(k)} D_{\psi_s(k),\psi_t(j)}$$

Pour faire apparaître les produits AC et BD, on découpe la somme principale en m sommes, afin de simplifier le calcul des valeurs de ϕ et ψ : on utilise

deux variables u et v parcourant $[\![1,m]\!]$ et $[\![1,s]\!]$, en posant $k = s*(u-1)+v$.

$$\Big((A\otimes B)(C\otimes D)\Big)_{i,j} = \sum_{u=1}^{m}\sum_{v=1}^{s} A_{\phi_r(i),u}C_{u,\phi_t(j)}B_{\psi_r(i),v}D_{v,\psi_t(j)}$$

$$\Big((A\otimes B)(C\otimes D)\Big)_{i,j} = \left(\sum_{u=1}^{m} A_{\phi_r(i),u}C_{u,\phi_t(j)}\right)\left(\sum_{v=1}^{s} B_{\psi_r(i),v}D_{v,\psi_t(j)}\right)$$

$$\Big((A\otimes B)(C\otimes D)\Big)_{i,j} = (AC)_{\phi_r(i),\phi_t(j)}(BD)_{\psi_r(i),\psi_t(j)}$$

$$\Big((A\otimes B)(C\otimes D)\Big)_{i,j} = \Big((AC)\otimes(BD)\Big)_{i,j}$$

□

Propriété V.2.1 (Trace). *Pour A et B deux matrices quelconques :*

$$tr(A\otimes B) = tr(A)*tr(B)$$

Propriété V.2.2 (Rang). *Pour A et B deux matrices quelconques :*

$$rg(A\otimes B) = rg(A)*rg(B)$$

Propriété V.2.3 (Déterminant). *Pour A et B deux matrices carrées, d'ordres respectivement n et m :*

$$det(A\otimes B) = det^m(A)*det^n(B)$$

Propriété V.2.4 (Valeurs propres). *Soient A et B deux matrices de spectres $(\lambda_i)_{1\leq i\leq n}$ et $(\mu_j)_{1\leq j\leq m}$: le spectre de $(A\otimes B)$ est $(\lambda_i\mu_j)_{\substack{1\leq i\leq n\\1\leq j\leq m}}$, les valeurs propres étant comptées avec leur multiplicité.*

Propriété V.2.5 (Inversibilité). *Si A et B sont inversibles, alors $A\otimes B$ l'est aussi ; dans ce cas, $(A\otimes B)^{-1} = A^{-1}\otimes B^{-1}$.*

Propriété V.2.6 (Diagonalisabilité). *Si A et B sont diagonalisables, alors $A\otimes B$ l'est aussi ; dans ce cas, si $P^{-1}AP = D$ et $Q^{-1}BQ = D'$, alors $(P^{-1}\otimes Q^{-1})(A\otimes B)(P\otimes Q) = D\otimes D'$.*

V.3 Codes Python

```
# -*- coding: utf-8 -*-
import time

def produit_scalaire_colonnes(M,i,j):
    ps = 0
    for k in range(len(M)):
        ps += M[k][i]*M[k][j]
    return ps

def produit_transpose(M):
    return [[produit_scalaire_colonnes(M,i,j) for j in range(len(M))]
                for i in range(len(M))]


def homothetie_recursive(M,k):
    def aux(i,j):
        if i>=len(M):
            return True
        if j>=len(M):
            return aux(i+1,0)
        return aux(i,j+1) and (M[i][j]==0 or (i==j and M[i][j]==k))
    return aux(0,0)


def hadamard_recursive(M):
    return homothetie_recursive(produit_transpose(M), len(M))

def colonne_orthogonale_aux_precedentes(M,i):
    res,k = True,0
    while res == True and k < i:
        res = (produit_scalaire_colonnes(M,k,i)==0)
        k+=1
    return res

def denombrer_matrices_de_Hadamard_naif(n):
    matrices = []
    V = [[0 for i in range(n)] for i in range(n)]

    def aux(i,j,M):
```

```
        if i>= n :
            if hadamard_recursive(M):
                matrices.append([l.copy() for l in M])
        elif j>= n:
            aux(i+1,0,M)
        else:
            M[i][j] = 1
            aux(i,j+1,M)
            M[i][j] = -1
            aux(i,j+1,M)

    debut = time.clock()
    aux(0,0,V)
    fin = time.clock()
    return (matrices,fin-debut)

def denombrer_matrices_de_Hadamard_backtracking(n):
    matrices = []
    V = [[0 for i in range(n)] for i in range(n)]

    def aux(i,j,M):
        if j>= n :
            matrices.append([l.copy() for l in M])
        elif i>= n:
            if colonne_orthogonale_aux_precedentes(M,j):
                aux(0,j+1,M)
        else:
            M[i][j] = 1
            aux(i+1,j,M)
            M[i][j] = -1
            aux(i+1,j,M)

    debut = time.clock()
    aux(0,0,V)
    fin = time.clock()
    return (matrices,fin-debut)

def denombrer_matrices_de_Hadamard_backtracking_compteur(n):
    V = [[0 for i in range(n)] for i in range(n)]
    l = [0]
```

```
    def aux(i,j,M):
        if j>= n :
            l[0]+=1
            print(l[0])
        elif i>= n:
            if colonne_orthogonale_aux_precedentes(M,j):
                aux(0,j+1,M)
        else:
            M[i][j] = 1
            aux(i+1,j,M)
            M[i][j] = -1
            aux(i+1,j,M)

    debut = time.clock()
    aux(0,0,V)
    fin = time.clock()
    return (l[0],fin-debut)

def denombrer_normalisees(n):
    l = [0]
    V = [[0 for i in range(n)] for i in range(n)]
    for k in range(n):
        V[k][0] = 1
        V[0][k] = 1

    def aux(i,j,M):
        if j>= n :
            l[0]+=1
        elif i>= n:
            if colonne_orthogonale_aux_precedentes(M,j):
                aux(1,j+1,M)
        else:
            M[i][j] = 1
            aux(i+1,j,M)
            M[i][j] = -1
            aux(i+1,j,M)

    debut = time.clock()
    aux(1,1,V)
    fin = time.clock()
    return (l[0],fin-debut)
```

```
def existe_matrice_de_Hadamard(n):
    V = [[0 for i in range(n)] for i in range(n)]

    def aux(i,j,M):
        if j>= n :
            return True
        elif i>= n:
            if colonne_orthogonale_aux_precedentes(M,j):
                return aux(0,j+1,M)
        else:
            M[i][j] = 1
            M1 = aux(i+1,j,M)
            if M1 != None:
                return M1
            else:
                M[i][j] = -1
                return aux(i+1,j,M)
    debut = time.clock()
    R = aux(0,0,V)
    fin = time.clock()
    return (R,fin-debut)


def circulante(l):
    n = len(l)
    c = lambda i : (i + 1) % n
    def aux(i,u,res):
        if i>=n: return res
        v = [u[c(k)] for k in range(n)]
        res.append(v)
        return aux(i+1,v,res)
    return aux(1,l,[l])


def hadamard_circulante(n):
    lignes = []
    L0 = [0 for i in range(n)]
    def aux(i,L):
        if i>=n:
            if hadamard_recursive(circulante(L)):
```

```
                lignes.append(L.copy())
        else:
            L[i]=1
            aux(i+1,L)
            L[i]=-1
            aux(i+1,L)
    aux(0,L0)
    return lignes
```

Références

[1] Martin HARWIT et Neil J.A. SLOANE. *Hadamard Transform Optics*. Anglais. Academic Press, 1979. 227 p. ISBN : 9780124335516.

[2] Philippe LANGEVIN. *La formule Alice : un détour par le pays des merveilles suggéré par J. Hadamard*. Français. 1999. URL : http://langevin.univ-tln.fr/notes/Paley/ (visité le 02/03/2016).

[3] Wayne STAHNKE. « Primitive Binary Polynomials ». Anglais. In : *Mathematics of Computation* 27 (124 1973). URL : http://www.ams.org/journals/mcom/1973-27-124/S0025-5718-1973-0327722-7/S0025-5718-1973-0327722-7.pdf (visité le 06/04/2016).

[4] Ian M. WANLESS. « Permanents of matrices of signed ones ». Anglais. In : *Linear and Multilinear Algebra*, 53 (6 2005), p. 427–433. URL : http://dx.doi.org/10.1080/03081080500093990 (visité le 02/03/2016).

[5] WIKIPEDIA. *Gray code*. Anglais. 2016. URL : https://en.wikipedia.org/wiki/Gray_code (visité le 09/04/2016).

[6] WIKIPEDIA. *Hadamard matrix*. Anglais. 2016. URL : https://en.wikipedia.org/wiki/Hadamard_matrix (visité le 09/04/2016).

[7] WIKIPEDIA. *Hadamard's maximal determinant problem*. Anglais. 2016. URL : https://en.wikipedia.org/wiki/Hadamard's_maximal_determinant_problem (visité le 09/04/2016).

[8] WIKIPEDIA. *Kronecker product*. Anglais. 2016. URL : https://en.wikipedia.org/wiki/Kronecker_product (visité le 09/04/2016).

[9] WIKIPEDIA. *Payley construction*. Anglais. 2016. URL : https://en.wikipedia.org/wiki/Paley_construction (visité le 09/04/2016).

[10] WIKIPEDIA. *Walsh function*. Anglais. 2016. URL : https://en.wikipedia.org/wiki/Walsh_function (visité le 09/04/2016).