De l'algèbre de Hadamard à la Transformation Cognitive

Gérard A. Langlet

CEA/DSM/DRECAM/SCM/LIT (Lab. d'Informatique Théorique)
C.E. Saclay, 91191-Gif sur Yvette CEDEX

Résumé

Après un rappel de la construction de Hadamard et de la transformation orthogonale qu'elle permet de pratiquer, on montre sur des exemples, comment cette algèbre se simplifie - en quatre étapes - et on aboutit à la simplification ultime du traitement exact d'informations massives par la transformation cognitive, équivalent optimal de toutes les transformations orthogonales classiques, dès que l'information est considérée - comme elle doit l'être, en binaire pur.

Mots-clé : Transformations orthogonales, Fourier, Hadamard, fractals, binaire.

Algèbre de Hadamard - Rappels

Soit la matrice H_2 de rang 2 :

```
1  1
1 -1
```

On définit la matrice H_4 de rang 4 comme composée de trois blocs de rang 2 contenant H_2 et d'un bloc de rang 2 (le bloc Sud-Est) contenant $-H_2$:

```
1  1  1  1
1 -1  1 -1
1  1 -1 -1
1 -1 -1  1
```

On peut itérer cette construction autosimilaire en construisant H_8, matrice de Hadamard de rang 8, par remplacement de chaque terme `1` de la matrice H_4 par un bloc H_2 de rang 2, et chaque terme `-1` de H_4 par un bloc $-H_2$ de rang 2 (certains mathématiciens utilisent des produits de *Kronecker*, alors qu'il est beaucoup plus simple de procéder par concaténation de blocs de données).

On peut aussi construire directement H_{16} à partir de H_4 en remplaçant chaque `1` de H_4 par H_4 et chaque `-1` de H_4 par $-H_4$. Sur ordinateur, il est également très rapide de remplir les quadrants Nord-Ouest, Nord-Est et Sud-Ouest, par des matrices H_n, puis le quadrant Sud-Est par une matrice $-H_n$, pour obtenir la matrice H_{2n}, par simple recopie de blocs, avec le minimum d'opérations.

Voici H_{16} :

```
 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
 1 -1  1 -1  1 -1  1 -1  1 -1  1 -1  1 -1  1 -1
 1  1 -1 -1  1  1 -1 -1  1  1 -1 -1  1  1 -1 -1
 1 -1 -1  1  1 -1 -1  1  1 -1 -1  1  1 -1 -1  1
 1  1  1  1 -1 -1 -1 -1  1  1  1  1 -1 -1 -1 -1
 1 -1  1 -1 -1  1 -1  1  1 -1  1 -1 -1  1 -1  1
 1  1 -1 -1 -1 -1  1  1  1  1 -1 -1 -1 -1  1  1
 1 -1 -1  1 -1  1  1 -1  1 -1 -1  1 -1  1  1 -1
 1  1  1  1  1  1  1  1 -1 -1 -1 -1 -1 -1 -1 -1
 1 -1  1 -1  1 -1  1 -1 -1  1 -1  1 -1  1 -1  1
 1  1 -1 -1  1  1 -1 -1 -1 -1  1  1 -1 -1  1  1
 1 -1 -1  1  1 -1 -1  1 -1  1  1 -1 -1  1  1 -1
 1  1  1  1 -1 -1 -1 -1 -1 -1 -1 -1  1  1  1  1
 1 -1  1 -1 -1  1 -1  1 -1  1 -1  1  1 -1  1 -1
 1  1 -1 -1 -1 -1  1  1 -1 -1  1  1  1  1 -1 -1
 1 -1 -1  1 -1  1  1 -1 -1  1  1 -1  1 -1 -1  1
```

Voici les valeurs des déterminants (non nuls) des premières matrices de Hadamard :

H_2	H_4	H_8	H_{16}	H_{32}
-2	16	4096	4294967296	1.20892582E24

Voici la matrice inverse de H_2 :

```
1/2  1/2
1/2 -1/2
```

L'expression sous forme de fraction montre que la matrice inverse de H_2 est H_2 dont chaque terme est divisé par le rang (au sens mathématique), ici 2.

Voici les 8 premières lignes et les 8 premières colonnes de la matrice inverse de H_{32} :

```
 0.03125  0.03125  0.03125  0.03125  0.03125  0.03125  0.03125  0.03125
 0.03125 -0.03125  0.03125 -0.03125  0.03125 -0.03125  0.03125 -0.03125
 0.03125  0.03125 -0.03125 -0.03125  0.03125  0.03125 -0.03125 -0.03125
 0.03125 -0.03125 -0.03125  0.03125  0.03125 -0.03125 -0.03125  0.03125
 0.03125  0.03125  0.03125  0.03125 -0.03125 -0.03125 -0.03125 -0.03125
 0.03125 -0.03125  0.03125 -0.03125 -0.03125  0.03125 -0.03125  0.03125
 0.03125  0.03125 -0.03125 -0.03125 -0.03125 -0.03125  0.03125  0.03125
 0.03125 -0.03125 -0.03125  0.03125 -0.03125  0.03125  0.03125 -0.03125
```

Ainsi, H_{32} dont chaque terme est divisé par le rang, 32, devient la matrice inverse de H_{32}.

Il en sera ainsi pour tout rang (par définition une puissance positive de 2),... jusqu'à l'infini.

Alors, toutes les matrices de Hadamard, si on divise chaque terme par la racine carrée de leur déterminant, deviennent auto-inverses (mais leur codage en entiers simples n'est plus possible).

Ceci signifie que si on multiplie une telle matrice de rang N par une matrice-colonne de N lignes, contenant par exemple les amplitudes d'un signal échantillonné sur N valeurs, on obtiendra une transformée de ce signal, dite ***transformée de Hadamard***.

En effectuant le même produit sur cette transformée, on retrouvera le signal original.

La transformation de Hadamard n'est pas celle de *Fourier*, mais elle permet, comme cette dernière, d'obtenir un spectre de fréquence (anamorphosé) sur lequel il est possible d'agir, par suppression ou correction, avant de pratiquer la transformation inverse.

Par rapport à Fourier, le principe est beaucoup plus simple. En outre, il suffit de conserver (et ces valeurs, toujours puissances de 2, sont codables sur ordinateur très facilement) les constantes (déterminants ou racines carrées de ces derniers) et d'opérer avec des matrices qui, pour toute dimension de signal qui soit puissance de 2, ne contiennent que des termes 1 ou -1.

Ainsi, le calcul des produits matriciels (opérations toujours linéaires) se résume à pratiquer des additions et des soustractions : il n'apparaît plus ni divisions ni multiplications. Sur ordinateur vectoriel, on aurait intérêt à utiliser des entiers (longs) et non plus des flottants; il ne se produit plus de troncature si on ne dépasse pas la limite du plus grand entier codable en valeur absolue, de l'ordre de du milliard; (on peut aussi, en langage-machine, engendrer un codage en entiers de haute précision, nécessaire pour effectuer des calculs très rapides, et toujours exacts, sur un nombre de chiffres significatifs encore plus grand).

A cause de la linéarité et de la simplicité de cette transformation, les Américains (et les Russes) s'y sont beaucoup intéressés surtout à l'époque balbutiante de la conquête de l'espace, en particulier pour la transmission, après compression et éventuellement correction, des images en provenance de la Lune ou de Mars.

A cette époque, les ordinateurs étaient plus chers et moins rapides. Beaucoup d'énergie fut dépensée pour construire des circuits spécialisés, ne comprenant que des additionneurs; de très nombreux congrès, en particulier à Washington DC, furent organisés, avec un pic majeur vers les années 1970.

Aux Etats-Unis, le pionnier (qui retrouva la construction de Jacques Hadamard et la développa), s'appelle G. Walsh. (C'est ainsi que le nom de transformée de Hadamard fut lui-même transformé par les Anglo-Saxons en transformée de Hadamard-Walsh, puis en transformée de Walsh-Hadamard, puis, tout simplement, en transformée de Walsh, et également par les théoriciens russes qui en avaient assez de traîner une appellation trop longue). Il est important, néanmoins, de connaître ce nom, car, lorsqu'on effectue une recherche bibliographique, on peut passer à côté de références fort intéressantes, si l'on interroge avec, pour critère, le seul nom de Hadamard.

Les domaines où la transformation de Hadamard est la plus utilisées ont été (et sont encore) l'interférométrie et la transmission d'images (en recherche spatiale comme en télévision). Bien que cette transformation puisse certainement rendre de nombreux services en spectroscopie, elle est, en général, inconnue dans la plupart des disciplines où elle aurait pu jouer un rôle fondamental. Par contre, on trouve des utilisations fort spectaculaires dans des domaines très variés, comme le biomédical (étude des électroencéphalogrammes ou des cycles menstruels) ou la criminologie (reconnaissance par empreintes digitales, très développée aux Etats-Unis).

En étudiant de plus près les propriétés des signaux transformés, on s'aperçoit, par simple comparaison des résultats obtenus et des figures présentées, que la transformation de Hadamard se rapproche plus de la transformation par ondelettes (encore peu répandue mais, elle aussi, d'origine française - *Morlet*), que de celle de *Fourier*. Les ondelettes sont utilisées de façon de plus en plus intense, en acoustique, essentiellement pour la reconnaissance vocale.

Simplifications

Même si le calcul d'une transformée de Hadamard est rapide, à cause de la disparition des multiplications et des divisions, il ne faut pas oublier, hélas, que la présentation mathématique succincte faite ici, par l'exposé de la construction initiale des matrices de Hadamard, puis par produit matriciel, correspond à un algorithme dont le temps d'exécution, si on ne simplifiait pas plus avant, serait proportionnel au carré de la dimension du signal.

Il importe donc d'effectuer une compression algorithmique avant toute tentative de programmation pour de gros jeux de données.

D'abord, le fait que les matrices de Hadamard ne contiennent que des termes 1 ou −1, en outre disposés régulièrement sous forme de blocs plus ou moins répétés, avec ou sans changement de signe, va permettre de NE PAS conserver en mémoire l'intégralité de la matrice.

Par exemple, une matrice de Hadamard H_{1024} de rang 1024 aurait 1048576 termes, ce qui, même si on code en entiers courts (deux octets par nombre) remplirait immédiatement 2 Méga-octets de mémoire d'ordinateur.

D'abord, on peut coder −1 sous forme d'un 0, donc utiliser du binaire pur. En effet, tout terme valant 0 est différent de 1 et correspond donc à −1, ou, simplement à un changement de signe du terme numérique du signal sur lequel il s'applique. Cette seule remarque fait gagner un facteur 16 sur le codage de la matrice : on passe de 2 Méga-octets à 125 kilo-octets pour H_{1024}.

On peut alors opérer vectoriellement (ou en parallèle, sur une machine adéquate). En effet, chaque terme d'un produit matriciel est une somme de termes (on n'a plus à effectuer QUE des additions), dont le signe a été changé (par basculement du seul bit de signe) *si et seulement si* le masque de Hadamard correspondant à une ligne de la matrice H adéquate, contient un 0 pour le terme en question. Il va de soi que le compilateur du langage de programmation utilisé doit produire un code binaire optimisé, sinon, il vaut mieux programmer la transformation en assembleur ou en langage-machine.

Ceci constitue la simplification **Numéro 1**.

Car on peut aller plus loin :

Les matrices de Hadamard ont, bien évidemment, des propriétés d'auto-similarité qu'il convient de mettre à profit; ce sont, même si cela ne se voit pas (et, à l'époque de Hadamard, le terme n'était pas encore à la mode) des matrices parfaitement ***fractales***.

On démontre alors, par récurrence (les informaticiens utiliseront éventuellement la "récursivité" pour en faire la preuve par un programme), que si la longueur (dimension) N du signal, déjà puissance de 2, est une puissance paire de 2, donc un carré, on peut, d'abord, disposer les valeurs des amplitudes de ce signal sous la forme d'une matrice carrée dont le rang est, bien sûr, **n** la racine carrée de N, puis effectuer un double produit matriciel (à gauche et à droite) par la matrice de Hadamard de rang **n**, et obtenir la transformée, en déroulant la matrice obtenue après ce double produit, en un signal linéaire de longueur N.

Alors, le temps d'exécution nécessaire n'est plus proportionnel au carré de N mais seulement à N.Log(N), le gain devenant encore plus intéressant si on tient compte, ce qui reste possible, de la simplification Numéro 1 précédemment décrite. La consommation de mémoire chute en conséquence :

Pour traiter un signal de 1024 valeurs, il suffit (et on peut maintenant se le permettre), de conserver en mémoire une matrice H_{32}, comprenant 1024 termes; si cette matrice est codée en bits, on parvient alors à un encombrement - presque ridicule - de 1024 bits soit 128 octets (pas plus que le code ASCII sur 7 bits).

On peut se permettre d'aborder enfin les transformations bidimensionnelles : une image en pixels (bits sur un écran noir et blanc de résolution 1024x1024) correspond à une matrice carrée de 1048576 bits (et non pas octets, car un écran, même si les logiciels qui le gèrent considèrent des nuances de gris - des intensités - avec une échelle numérique, par exemple entre 0 pour une plage carrée toute noire et 255 pour une plage carrée toute blanche, ou inversement - ne sait gérer que des pixels).

De nombreuses formules connues (par exemple pour Fourier) vont pouvoir s'appliquer en bidimensionnel (et aussi en multidimensionnel) à Hadamard, mais, cette fois en bits directement.

En effet, une plage d'intensité numérique donnée dans une échelle quelconque, par exemple 16 nuances ou 64 ou 256 ou plus, correspond, sur l'écran, à un petit carré de pixels dans lequel le rapport de bits à 1, par rapport aux bits à 0, code la nuance en question par un motif régulier plus ou moins rempli (par exemple un bit sur deux à 1 dans un motif en quinconce pour le gris moyen); de meilleurs résultats (cela dépend du type d'image) sont aussi obtenus, en traitement d'image, par une simple distribution au hasard des bits à 1 dans le motif, car l' il humain détecte les trop grandes régularités, sauf s'il regarde l'écran d'assez loin.

(Pour le traitement de la couleur, on agit, en parallèle, indépendamment, sur trois matrices de pixels, donc de bits, une pour chaque couleur fondamentale du système **RVB**, donc sur 3 plans d'une hyper-matrice).

Voici un exemple, restreint à 64 amplitudes, pour lequel il suffit d'utiliser deux fois, une matrice de Hadamard de rang 8. Cet exemple reste valable pour des rangs bien plus grands, grâce à l'auto-similarité intrinsèque de la transformation.

Les 64 valeurs suivantes du signal S sont des entiers tirés au hasard :

```
8 48 29 34 14 3 43 43 59 24 33 53 2 3 33 42 0 24 4 26 43 37 59
54 33 5 41 26 44 58 48 16 3 47 21 40 48 63 23 15 62 46 48 41 4
40 56 17 27 49 30 15 17 22 10 31 57 58 3 57 32 33 20 63
```

Voici le résultat du double produit matriciel $H_8 \cdot S_m \cdot H_8$:

```
2087 -179  -61  -35   15  -91  159   97
-227 -165   13 -153 -255 -209  -83 -305
   3   33 -133 -115  279  -43   27   69
  65 -105  125   15 -127 -105 -191  -53
-109  173 -297   45 -205  -35   19   17
 125 -105 -115   43  -47 -117  261   -5
 -97 -143 -105  261  331   45  311 -179
 -71  219  -35 -253  137  107   -7   -9
```

Ce résultat est affiché sous la forme d'une matrice de rang 8. La matrice suivante est S_m, elle aussi de rang 8 :

```
 8 48 29 34 14  3 43 43
59 24 33 53  2  3 33 42
 0 24  4 26 43 37 59 54
33  5 41 26 44 58 48 16
 3 47 21 40 48 63 23 15
62 46 48 41  4 40 56 17
27 49 30 15 17 22 10 31
57 58  3 57 32 33 20 63
```

Voici le résultat S_t du simple produit matriciel H_{64} . S donc la transformée de Hadamard de S, si on divise tous les termes par 8. Ce résultat est le même que celui du double produit matriciel déroulé en mettant les lignes bout à bout.

```
2087 -179 -61 -35 15 -91 159 97 -227 -165 13 -153 -255 -209 -
83 -305 3 33 -133 -115 279 -43 27 69 65 -105 125 15 -127 -105
-191 -53 -109 173 -297 45 -205 -35 19 17 125 -105 -115 43 -47
-117 261 -5 -97 -143 -105 261 331 45 311 -179 -71 219 -35 -253
137 107 -7 -9
```

Voici le résultat S_{tt} du produit matriciel H_{64} . St (ou, linéarisé, le résultat du double produit matriciel de H_8 par la matrice obtenue en enroulant S_t ci-dessus) :

```
512 3072 1856 2176 896 192 2752 2752 3776 1536 2112 3392 128
192 2112 2688 0 1536 256 1664 2752 2368 3776 3456 2112 320
2624 1664 2816 3712 3072 1024 192 3008 1344 2560 3072 4032
1472 960 3968 2944 3072 2624 256 2560 3584 1088 1728 3136 1920
960 1088 1408 640 1984 3648 3712 192 3648 2048 2112 1280 4032 .
```

On pourra constater que chaque terme est simplement le produit de chaque terme du signal S d'origine, par 64, la dimension du signal.

L'important demeure **a)** la rapidité de l'algorithme, **b)** son exactitude numérique absolue, tant que l'on peut coder les valeurs en nombres entiers.

(Note. Cet exemple est obtenu, en *notation d'Iverson*, sans qu'il soit besoin d'écrire un programme quelconque, car *APL* est un langage vectoriel et matriciel, par définition).

Ceci constitue la simplification **Numéro 2**.

Jusqu'ici, on n'a fait que rapporter des résultats connus, que l'on peut rassembler en effectuant une recherche bibliographique sur le sujet (toutefois dans des rapports parfois difficiles à trouver en France).

A partir de maintenant, les résultats exposés proviennent des recherches du LIT (Laboratoire d'Informatique Théorique) sur la compression algorithmique et sur l'unification des transformations orthogonales (sujet sur lequel travaille par exemple aussi le MIT, mais plutôt à partir de Fourier; voir **[Tolimieri]** comme référence la plus récente connue).

R. Tolimieri, M. An & Lu, Mathematics of Multidimensional Fourier Transform Algorithms (1993).

Vers le passage au binaire pur.
Des racines carrées récursives fort utiles

De nombreuses propriétés de l'algèbre et de la transformation de Hadamard se conservent si l'on ne brise pas la symétrie des transformations matricielles.

Au contraire, il va apparaître de nouvelles propriétés si on modifie la règle de symétrie initiale (−1 symétrique de 1 par rapport à 0, par *négation arithmétique*).

On va simplement changer d'algèbre et passer de l'algèbre entière modulo M (M étant l'entier-limite au-delà duquel on ne peut plus coder de données dans l'ordinateur sans dégrader le traitement de l'information par des troncatures - ce qui oblige à travailler effectivement dans une certaine fenêtre de valeurs, équivalente par translation, effectivement, à considérer une algèbre entière modulo M) à l'**algèbre modulo 2**, dont les seuls nombres sont 0 et 1.

On a déjà vu que les matrices de Hadamard se codaient aussi avec 0 et 1 ce qui aboutissait à une économie considérable de la mémoire (par simple transformation linéaire conservant 1 mais transformant −1 en 0).

Il est possible d'aller beaucoup plus loin, et ce, dans deux directions différentes.

D'abord une remarque concerne le double produit matriciel, déjà très réducteur (S_m est un signal enroulé en matrice) :

Le double produit matriciel $H_{\mathbf{n}} \cdot Sm \cdot H_{\mathbf{n}}$ avec $N=\mathbf{n}^2$ si N est la dimension de S, peut se simplifier, à condition que **n** soit lui-même un carré. Ce n'est pas le cas dans l'exemple numérique choisi plus haut; alors, prenons-en un petit morceau, les 16 premières valeurs, car 16 est le carré de 4, carré de 2 :

```
8 48 29 34 14 3 43 43 59 24 33 53 2 3 33 42
```

Voici la transformée de Hadamard de ce signal S_{16}, calculée par $H_{16} \cdot S_{16}$:

```
471 -29 -149 39 105 -31 129 1 -27 -39 -3 -87 -73 -81 -5 -93
```

puis calculée en matrice par le double produit matriciel $H_4 \cdot S_{16}m \cdot H_4$:

```
 471  -29 -149   39
 105  -31  129    1
 -27  -39   -3  -87
 -73  -81   -5  -93
```

Voici la transformée de la transformée (16 fois S_{16}) ou $H_{16} . H_{16} . S$:

```
128 768 464 544 224 48 688 688 944 384 528 848 32 48 528 672
```

Voici le simple produit $S_{16}m . H_4$:

```
 119 -45  -7 -35
 103  11 -69  11
 169  15  -3  55
  80 -10 -70   8
```

Et voici le double produit avec restructuration de S_{16} cette fois en hyper-matrice à quatre dimensions, chacune valant seulement 2 (l'expression ci-dessous est écrite directement en notation linéaire exécutable, normalisée internationale *ISO8485*, **ISO**, CH-Genève, 1989). Note. L'origine des indices vaut 0.

$$3\ 0\ 1\ 2\circledcirc H_2 +.\times (1\ 2\ 3\ 0\circledcirc S_{2222}) +.\times H_2$$

```
 119 -45
  -7 -35

 103  11
 -69  11

 169  15
  -3  55

  80 -10
 -70   8
```

S_{2222} est le signal enroulé en hyper-matrice à quatre dimensions 2 comme le résultat affiché ci-contre.

Le déroulement d'un signal enroulé en hyper-matrice, comme en matrice, consiste à lire les termes dans l'ordre de l'affichage (ou, hors FORTRAN, dans l'ordre de rangement dans la mémoire séquentielle de l'ordinateur, c'est tout simple). Les 4 dimensions (hyper-plans, plans, lignes, colonnes) de la matrice S_{2222} sont numérotées respectivement 0, 1, 2 et 3. L'écriture telle que $1\ 2\ 3\ 0^3$ signifie simplement que les dimensions de l'objet à 4 dimensions écrit à droite dans l'expression doivent subir une permutation telle que les nouvelles dimensions seront celles qui portaient anciennement les numéros 1, 2, 3 et 0, donc les plans, lignes, colonnes et hyper-plans de S_{2222} .

De même, $3\ 0\ 1\ 2\circledcirc$ indique que l'on doit prendre pour nouvelles dimensions respectivement les colonnes, hyperplans, plans et lignes dans les anciennes dimensions.

Sachant que le résultat, réenroulé en matrice de rang 4, et multiplié matriciellement par la matrice H_4 placée à gauche, produit la transformée de S_{16} cherchée après déroulement du résultat, il va suffire d'appliquer encore deux fois, une fois à gauche et l'autre fois à droite, la simple matrice H_2 pour obtenir aussi la transformée cherchée, en déroulant encore une fois une hyper-matrice à quatre dimensions dont chacune est de rang 2, avec permutation des dimensions d'une manière adéquate :

Bien sûr, si on programme en FORTRAN, on aboutit vite à un casse-tête, d'autant plus que, depuis les origines de ce langage, c'est toujours, dans le rangement des données en mémoire, le premier indice qui varie le plus vite. En *APL* ou en langage-machine, il n'y a pas grand chose à faire pour dérouler ou enrouler une matrice ou une hyper-hypermatrice, si on a su uvrer pour que données et résultats se trouvent dans le bon ordre en posant sainement les opérations. Le casse-tête se renforce en FORTRAN si, au lieu de traiter un signal de dimension 16 comme une hyper-matrice de dimension (2,2,2,2), on essaie de généraliser, par exemple à un signal de dimension 65536 enroulé en dimension (2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2), car il existe peu de compilateurs acceptant une telle dimension, et peu de programmeurs capables de maîtriser les indices de 16 boucles imbriquées ! On n'oserait proposer un tel remue-méninges à un fortraniste chevronné, pour traiter un signal d'un million de nombres, que si l'on possède des actions dans une fabrique d'aspirine ou de maxiton.

Pourtant, si on parvient à se sortir de ce puzzle, l'algorithme qui en découle s'apparente à celui de Winograd pour la transformation de Fourier rapide, car son temps d'exécution va devenir proportionnel à la dimension du signal, rigoureusement (en effet, le nombre théorique de boucles imbriquées est alors égal au logarithme en base 2 de la longueur du signal) et on n'a plus besoin que de la matrice H_2. Par avance, on sait que l'algorithme sera optimal, car, maintenant, l'utilisation de cette seule matrice H_2 peut se remplacer par la programmation de simples additions en changeant un seul signe une fois sur quatre.

Voici alors un modèle *APL* complet, une expression exécutable directement sur micro-ordinateur sans que l'on ait besoin d'écrire un fastidieux programme, fonctionnant en hyper-blocs et diminuant le nombre opérations arithmétiques, dans laquelle on fait appel à un quadruple produit matriciel, pour obtenir, linéarisée, la transformée de Hadamard d'un signal de 16 bits :

```
,1 2 3 0 ⍉H2+.×(2 3 0 1⍉H2+.×(1 2 3 0⍉2 2 2 2⍴S16)+.×H2)+.×H2

 71 -29 -149 39 105 -31 129 1 -27 -39 -3 -87 -73 -81 -5 -93
```

Notes.

Les signes "moins" de la notation *APL* ont été remplacés ici par des signes "moins" ASCII dans les résultats.

Le produit matriciel, noté par ailleurs dans le texte comme un point gras, s'exprime en *APL* par la composition de l'addition avec le produit, comme en mathématiques on utilise $f \circ g$, le simple point faisant alors office de fonctionnelle de composition au lieu du rond, dans ce que l'on appelle un produit interne. C'est tout de même bien plus simple d'utiliser + . × que de s'obstiner à écrire des échafaudages gréco-mathématiques tels que : $\Sigma\, a_{ij}.b_{jk}$ d'autant plus que la notation mathématique d'Iverson s'exécute directement sur tout ordinateur digne de ce nom, sans qu'il soit besoin de retranscrire le code des expressions en langage(s) dit(s) "évolué(s)", mais, en réalité fort dépassé(s), et de s'occuper, en outre, de dimensionnements, formatages divers et multiples indices de boucles; ce langage, intelligent, prévu 20 ans à l'avance pour des machines vectorielles encore dans l' uf, sait gérer cela dynamiquement, tout seul comme un grand; hélas, les programmeurs, masochistes, se complaisent dans leurs notations B, A, BA simplement initiales des mots "babil", "balbutiement" et "Babel". Et tous les "progrès" qu'a pu faire la norme du langage FORTRAN en post-fixant 90 au lieu de 77 ce qui a pris une quinzaine d'années, ont consisté à récupérer, intelligemment, une toute petite partie ce que les *connoisseurs* d'*APL* pratiquaient déjà, pour certains, vers 1967...

La jonglerie intellectuelle ne s'arrête pas ici pour autant, mais convient-il de dépenser du phosphore pour décomposer les produits matriciels de matrices d'ordre supérieur en produits de matrices H_2; encore faudrait-il que le jeu en vaille la chandelle, car on engendre facilement une matrice H_{16}, laquelle, codée en bits, tient dans 32 octets de mémoire, la taille d'un seul entier long !

En effet, si, dans l'expression précédente, on remplace *H2* par *H4* (donc H_2 par H_4), et si on récrit le redimensionnement 2 2 2 2 comme 4 4 4 4, l'expression nouvelle donnera la possibilité de traiter non plus un signal de 16 amplitudes (16 étant égal à 2^4), mais un signal de 256 amplitudes (car ce nombre est 4^4).

Pareillement, avec la matrice H_8 (donc 64 valeurs 1 ou −1 ce qui tient dans une toute petite mémoire), on traite, en théorie aussi bien qu'en pratique, un signal de 8^4 soit 4096 amplitudes; quant à la petite matrice H_{16} qui contient 256 termes, elle permet alors de traiter 65536 amplitudes, et ainsi de suite... On se rapproche de la célèbre formule de Planck, en T^4...relative à l'énergie rayonnée par un corps noir chauffé (utiliser *APL*, n'est- ce point tirer à boulets rouges pour faire fondre la glace du lac d'indifférence mathématique sous le soleil d'Austerlitz qui rayonne aussi à la puissance quatre ?)

Ceci constitue la simplification **numéro 3** (encore très classique... et très loin, encore, du but). Chauffons maintenant... à blanc.

Passage au fractal pur

Oublions tout ce qui vient d'être dit, démontré ou calculé, SAUF la définition initiale, de Hadamard, concernant H_2, et la construction de H_4 en fonction de H_2, mais passons en binaire pur :

On a vu que la matrice H_2 est devenue : $\begin{matrix}1&1\\1&0\end{matrix}$ par transformation linéaire conservant 1, mais changeant −1 en 0, ce qui permettait de représenter les matrices H en binaire avec une économie substantielle de mémoire.

Maintenant, changeons radicalement d'*algèbre* :

0 ne va plus être considéré comme donné par substitution de 0 à −1 mais devient (comme −1 était la négation arithmétique de 1 et réciproquement) la négation **LOGIQUE** de 1 et réciproquement.

Cette subtilité de définition introduit une simplification plus considérable que tout ce qui avait été vu jusqu'ici (mais les autres simplifications, provisoirement oubliées, vont rester valables avec ce changement de définition, qui crée une nouvelle famille de transformations, utilisables sur des signaux codés en binaire).

Appelons G_2 cette nouvelle matrice et fabriquons G_4 à partir de trois blocs G_2 et d'un bloc Z_2 c'est-à-dire d'une matrice de rang 2 identiquement nulle.

```
G2 :     1 1          G4 :     1 1 1 1
         1 0                   1 0 1 0
                               1 1 0 0
                               1 0 0 0
```

La construction récursive lente, en doublant la taille à chaque fois seulement, ou explosive (en remplaçant directement chaque 1 de G_4 par G_4 et chaque 0 par Z_4, ce qui produit G_{16} puis G_{256} à partir de G_{16}, puis G_{65536} ... etc...) fonctionne toujours, mais remplace les matrices de Hadamard par un triangle de *Sierpinski* immergé dans une matrice carrée, le ***FRACTAL*** qui hante tous les livres sur le chaos (et dont les auteurs passent sous silence les fantastiques propriétés matricielles, directement liées à la transformation de type Fourier, entre autres).

Alors que les matrices H de Hadamard ne peuvent avoir comme rang qu'une puissance de 2, les matrices G ("génitons", mot plus euphonique que "sierpinskons") vont, en outre, posséder des propriétés intéressantes, pour n'importe quel rang.

Par exemple, en gommant la dernière ligne et la première colonne de G_4, on obtient la matrice-géniton G_3 :

```
1 1 1
0 1 0
1 0 0
```

puis, en gommant autant de dernières lignes qu'on veut, et autant de premières colonnes d'un gigantesque géniton, on obtient G_N pour toute valeur de N égale ou supérieure à 2 (le géniton est généralisé).

Ces matrices sont alors fibonacciennes et hyper-fibonacciennes : leur puissances successives sont des suites ou des hypersuites matricielles fibonacciennes, conduisant à la symétrie pentagonale, mais aussi à toutes les symétries classiques d'ordre 2 (images dans un miroir), d'ordre 3 (rotation des axes ou des faces d'un trièdre autour de sa diagonale), d'ordre 6 (symétrie hexagonale), etc... dès que l'on approfondit l'étude des symétries fractales conjuguées, en algèbre entière modulo 2, isomorphe de l'algèbre logique.

Dans cette algèbre, que l'on n'étudie pas et donc que l'on n'utilise guère, tout devient additif modulo 2, de telle sorte que la complexité s'évanouit comme par enchantement : les motifs que l'on ne savait pas interpréter (que l'on jugeait chaotiques (ou simplement, en anglais "random") se résolvent en superposition de motifs plus simples, comme des signaux de fréquences multiples se superposent en une modulation complexe.

La différence, subtile, avec l'algèbre numérique discrète classique ou avec les fonctions continues (par exemple les sinusoïdes de Fourier) fait que les lois de symétrie apparaissent clairement, de sorte que la décomposition (ou déconvolution) devient beaucoup plus facile que l'on ne pensait :

Un signal tel que 0 1 0 1 0 1 0 1 0 1 ..., répété à l'infini, peut apparaître comme l'échantillonnage d'une sinusoïde (ou d'une sinusoïde au carré, qui est encore une sinusoïde). Inversement, la sinusoïde apparaît comme l'enveloppe de cette modulation quantique, si on essaie de lisser les valeurs discrètes. L'information d'un signal, correctement transcrite en bits, c'est-à-dire linéairement et non plus d'une manière logarithmique, est échantillonnable, décomposable, recomposable, compressible, en bref taillable et corvéable à merci.

Le produit matriciel d'un géniton quelconque, à gauche ou à droite, par un signal de même longueur que le rang de la matrice, mais exprimé en bits (alors, on n'est plus en modulation d'amplitude, mais en modulation de fréquence ou de phase, ce qui en accroît la fiabilité), permet immédiatement d'obtenir les seuls équivalents possibles de TOUTES les **transformations orthogonales** connues, discrètes ou continues, en algèbre classique. Cela est maintenant démontré mathématiquement (sans appel, mais avec *APL*).

Et les formules simplificatrices utilisées plus haut, avec la transformation de Hadamard, vont s'avérer effectivement inutiles (elles auront servi simplement de preuves et de points de passage intermédiaires), car on réduit maintenant sommes et changements de signe, à une seule et unique application logique, à savoir la somme modulo 2, identique à la différence modulo 2 en algèbre entière modulo 2 (par définition), mais surtout isomorphe du Ou Exclusif de l'algèbre logique, qui devient la SEULE opération utile; la négation logique (ou changement du bit de signe pour un nombre), n'est, elle-même, QUE l'application du Ou Exclusif avec 1, ce que savent nombre de programmeurs utilisant l'Assembleur ou l'*APL*.

Les produits matriciels se simplifient. En effet, multiplier par 1 revient à ne rien faire (ce qui est assez facile). Et multiplier par 0 revient à ignorer le terme, en algèbre modulo 2 comme en algèbre classique. Elever 1 à une puissance quelconque revient encore à ne rien faire. Pareillement, élever 0 à une puissance quelconque (de préférence positive) ne fait pas grand chose non plus. En fait, il est impossible, pour effectuer une transformation réversible quelconque, de ne pas trouver une solution linéaire : la non-linéarité se réduit à un mythe antique, dû à une méconnaissance des propriétés de l'algèbre p-adique avec $p=2$.

Ces propriétés sont extrêmement connectées à celles des pliages (enseignés à tous les écoliers japonais). Une cocotte en papier résulte d'une série de transformations linéaires à partir d'une feuille de papier. C'est la séquence de ces transformations élémentaires qui aboutit à construire la cocotte aussi bien que des objets plus compliqués (qui apparaissent non-linéaires quand on ne sait pas décomposer). Les algorithmes de tri (Batcher), et la transformation rapide de Fourier (méthode de Winograd), le montrent déjà avec les nombres. Et on ne peut pas s'en rendre compte quand on a décidé, a priori, de reproduire un signal avec une superposition de fonctions au départ non linéaires, comme, justement, des sinusoïdes...

Un mouvement sinusoïdal (non linéaire) peut résulter de la projection d'un mouvement circulaire uniforme, lequel, lorsqu'on considère comme des "variables" le *module* (constant car il s'agit du rayon) et l'*angle*, devient **linéaire** en déplacement angulaire; alors, pourquoi raisonner en sinus, sinon pour se compliquer l'existence: sur ordinateur, cette vision trigonométrique, même reformulée à l'aide d'exponentielles complexes, convient fort mal. En outre, sur le plan strictement mathématique, on va parfois aboutir à des impasses gênantes (systèmes différentiels non intégrables, NP-complétude par abus de combinatoire). S'il existe des problèmes mathématiquement insolubles (pour lesquels la Nature a pourtant, elle, une solution unique), c'est souvent parce que le choix des paramètres descripteurs du système est inadéquat, parce que les axiomes initiaux sont à revoir, parce que l'algèbre utilisée n'est pas la bonne, parce que les "simplifications" imposées n'ont aucun sens physique (cordon **sans** masse pour le pendule, mouvement d'un véhicule avec rotation de ses roues **sans** frottement, gaz **infiniment** compressible, déplacements supposés **aléatoires**, information **moyennable** et **lissable**, développements limités dont chacun des termes ignorés est certes petit mais dont on n'arrivera JAMAIS à prouver que la somme est nulle - on l'*admettra* seulement - et tutti quanti dinosauri ou, en bon français, couleuvres de manuels scolaires inapplicables à la plupart des problèmes sérieux).

Retrouvons maintenant une partie de la *mémoire*...

Le symbole point "." (gras) représente toujours le produit matriciel, mais celui-ci se comprend, à partir de maintenant, **modulo 2** (le résultat est le reste 0 ou 1 de la division entière par 2 du résultat du produit matriciel classique; mais on peut effectuer le produit matriciel modulo 2 directement en binaire par composition du **Ou Exclusif** et du **ET**, directement en bits, sans gaspiller une seule position de mémoire).

Si B est un signal binaire de longueur (dimension) puissance de 4, (ici 16) :

0 0 0 1 0 1 0 0 1 1 0 1 0 1 1 1

le produit G_{16} . B est : 0 0 0 1 0 1 0 1 0 0 1 0 1 0 0 0

alors que le produit G_4 . B*m* . G_4 , si B*m* est B enroulé en matrice (ici de rang 4), vaut, une fois déroulé :

0 0 0 1 0 1 0 1 0 0 1 0 1 0 0 0 résultat que l'on appellera R.

Mais attention !

Si on répète sur R, le produit G_{16} . R , ou, sur R*m* (R enroulé en matrice), le produit G_4 . R*m* . G_4 et que l'on déroule ce nouveau résultat, on trouvera :

1 0 0 1 0 1 0 0 1 0 1 0 1 0 0 0 résultat que l'on appellera RR.

Et ce résultat n'est PAS le signal binaire B d'origine. Que s'est-il donc passé ?

Pour le savoir, recommençons. Répétons, sur RR, le produit G_{16} . RR ou, sur RR*m* (RR enroulé en matrice), le produit G_4 . RR*m* . G_4 et déroulons ce nouveau résultat :

0 0 0 1 0 1 0 0 1 1 0 1 0 1 1 1

Il s'agit, cette fois, du signal B d'origine.

Le mystère est simple; on n'a plus affaire à une transformation orthogonale mais à une transformation trine ou TRIGONALE : ce qui est nouveau.

On a découvert une nouvelle paire de transformations (qui, comme celle de Hadamard) donne des résultats exacts, MAIS, cette fois-ci, quelle que soit la taille de B, car le déterminant modulo 2 de tous les génitons G vaut 1; jamais plus il ne peut se produire une seule erreur sur un seul bit de donnée initiale.

Mais on peut aussi retrouver une paire de transformations involutives, et non pas une seule; il suffit de retourner, soit verticalement, soit horizontalement, les génitons, opérateurs matriciels :

Le produit : Gv_{16} . B a pour résultat : 0 0 0 1 0 1 0 1 0 0 1 0 1 0 0 1 que l'on notera C.

Gv_{16} est G_{16} retourné *verticalement*.

Si G_2 est $\begin{matrix} 1 & 1 \\ 1 & 0 \end{matrix}$, son retournement (symétrique *vertical*) Gv_2 est $\begin{matrix} 1 & 1 \\ 0 & 1 \end{matrix}$.

Le produit : Gv_{16} . C a donne : 0 0 0 1 0 1 0 0 1 1 0 1 0 1 1 1 qui est B.

Et on remarque que C est l'envers de RR, autre résultat nouveau.

Regardons ce que produit l'autre retournement de G_{16} :

Le produit : Gh_{16} . B a pour résultat : 0 0 0 1 0 1 0 0 1 0 1 0 1 0 0 0 que l'on notera H.

Gh_{16} est G_{16} retourné *horizontalement*.

Si G_2 est 1 1, son symétrique *horizontal* Gh_2 est 1 0
1 0 1 1

Le produit : Gh_{16} . H donne : 0 0 0 1 0 1 0 0 1 1 0 1 0 1 1 1 qui est B.

Et on remarque alors que H est l'envers de R.

Ces propriétés sont générales, quelle que soient la taille et le contenu de B.

Bien entendu, on retrouvera des résultats analogues avec les doubles produits matriciels modulo 2, en opérant avec les deux retournements possibles de G_4, et on peut s'amuser à trouver les relations qui existent lorsque l'on utilisera G_2 ou les deux retournements possibles de G_2. Alors, ces relations se simplifient ici, à cause de la présence de nombreux zéros, alors qu'avec les matrices de Hadamard, les −1 ne se simplifient pas... (Quand la matrice de Hadamard est grande, le rapport du nombre de −1 au nombre de 1 reste de l'ordre de l'unité 1; voir à ce propos les illustrations qui suivent).

La compression ultime

Jetons tout ce qui précède pour procéder tout à fait autrement, et obtenir, dans le minimum de temps et avec le minimum de place en mémoire, les mêmes résultats qu'avec les derniers produits matriciels considérés, mais **sans** les produits matriciels, pour les transformations orthogonales couplées absolues que l'on appelle : transformation ***cognitive*** pour C et transformation ***hélicoïdale*** pour H.

On considère B, le même signal que précédemment et ses 16 bits :

```
0 0 0 1 0 1 0 0 1 1 0 1 0 1 1 1
```

On extrait un bit sur deux, en position impaire (ici les bits 1, 3, 5, 7, 9, 11, 13, 15). :

```
0   0   0   0   1   0   0   1
```

On extrait les bits de position paire; on les met dessous (en les cadrant à gauche) :

```
0   1   1   0   1   1   1   1
```

On effectue la somme ou la différence modulo 2 de ces deux lignes (Ou Exclusif fait l'affaire) :

```
0   1   1   0   0   1   1   0
```

On répète ce résultat en remettant en-dessous la ligne des bits pairs, ce qui, dans l'exemple traité, forme un tableau de 2 lignes et 8 colonnes :

```
0   1   1   0   0   1   1   0
0   1   1   0   1   1   1   1
```

On extrait une colonne sur deux (les colonnes impaires), comme précédemment:

```
0 1 0 1
0 1 1 1
```

On extrait les colonnes paires que l'on recadre en-dessous et à gauche :

```
1 0 1 0
1 0 1 1
```

On effectue la somme ou différence modulo 2 terme à terme de ces deux derniers *tableaux*

```
1 1 1 1
1 1 0 0
```

On répète ce résultat en accolant en-dessous le tableau précédent des colonnes paires, ce qui, dans l'exemple traité, forme maintenant un tableau de 4 lignes et 4 colonnes :

```
1 1 1 1
1 1 0 0
1 0 1 0
1 0 1 1
```

On retient une colonne sur deux (encore les colonnes impaires) :

```
1 1
1 0
1 1
1 1
```

On prend les colonnes paires que l'on met en dessous :

```
1 1
1 0
0 0
0 1
```

On effectue la somme ou différence modulo 2 terme à terme des deux tableaux :

```
0 0
0 0
1 1
1 0
```

On répète ce résultat en mettant en-dessous le tableau précédent des colonnes, ce qui, dans l'exemple traité, forme alors un tableau de 8 lignes et 2 colonnes :

```
0 0
0 0
1 1
1 0
1 1
1 0
0 0
0 1
```

On prend la colonne impaire, puis la colonne paire, et on en effectue encore la somme ou différence terme à terme modulo 2. Comme il ne reste plus que deux colonnes, la sélection n'est pas trop difficile, et, pour économiser le papier, on écrit tout de suite, en ligne horizontale, cette dernière différence, à laquelle on attache la colonne paire, à droite :

0 0 0 1 0 1 0 1 0 0 1 0 1 0 0 1 ce qui est bien un résultat identique à la transformée C.

Si on reprend le même algorithme sur C comme donnée à transformer, on retrouvera B initial.

Pour obtenir H, il suffit, au lieu de reprendre la colonne paire, de reprendre la colonne impaire, et de la mettre AVANT la différence des deux colonnes, et non pas après.

Le nombre d'itérations est 4, le *logarithme en base 2* de la dimension (ici 16). Le nombre de différences terme à terme, par itération, est égal à la moitié de cette dimension du signal.

Jusqu'à présent, on n'a pas trouvé de procédure plus simple.

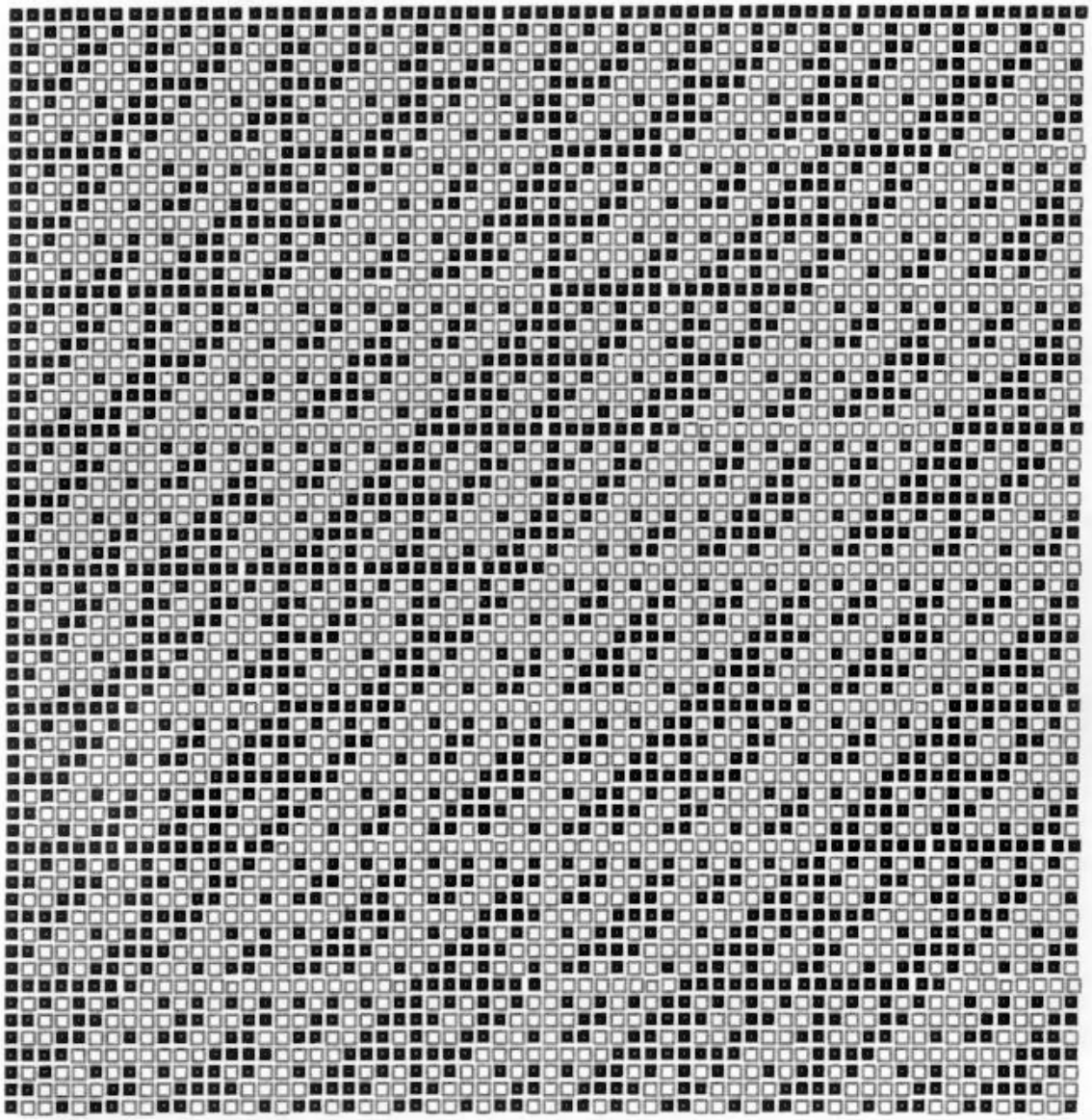
Pour traiter un signal d'un milliard de bits, il suffit de 30 itérations. Les masques ou suites d'indices à considérer sont triviaux, - et toujours les mêmes. Dans une machine non vectorielle et non parallèle, à registres de 32 bits, cela représente moins d'un *gigaflop*, et, si les mots ont 64 bits, seulement la moitié de cette valeur, un quasi-rien pour les micro-ordinateurs actuels de type 486 ou "Power-PC"...

Un milliard de bits, en ASCII étendu sur 8 bits, cela code 128 millions de caractères, donc 16 millions de mots d'un texte moyen à 8 caractères (espaces compris) par mot de langue française ou anglaise, donc 1 million de lignes à 16 mots par ligne, donc 20.000 pages à 50 lignes par page, donc 100 volumes de 200 pages, un bon quintal d'encyclopédies existantes, dans toutes les langues de l'Europe à la fois... de quoi lire pendant les grandes vacances. Mais, avec ce régime simplificateur, que lira-t-on l'an prochain ?

N.B. En notation d'Iverson, le programme complet tient en une ligne. Le présent texte comporte un peu plus de 33000 caractères, la racine carrée du milliard...

Représentations graphiques comparées

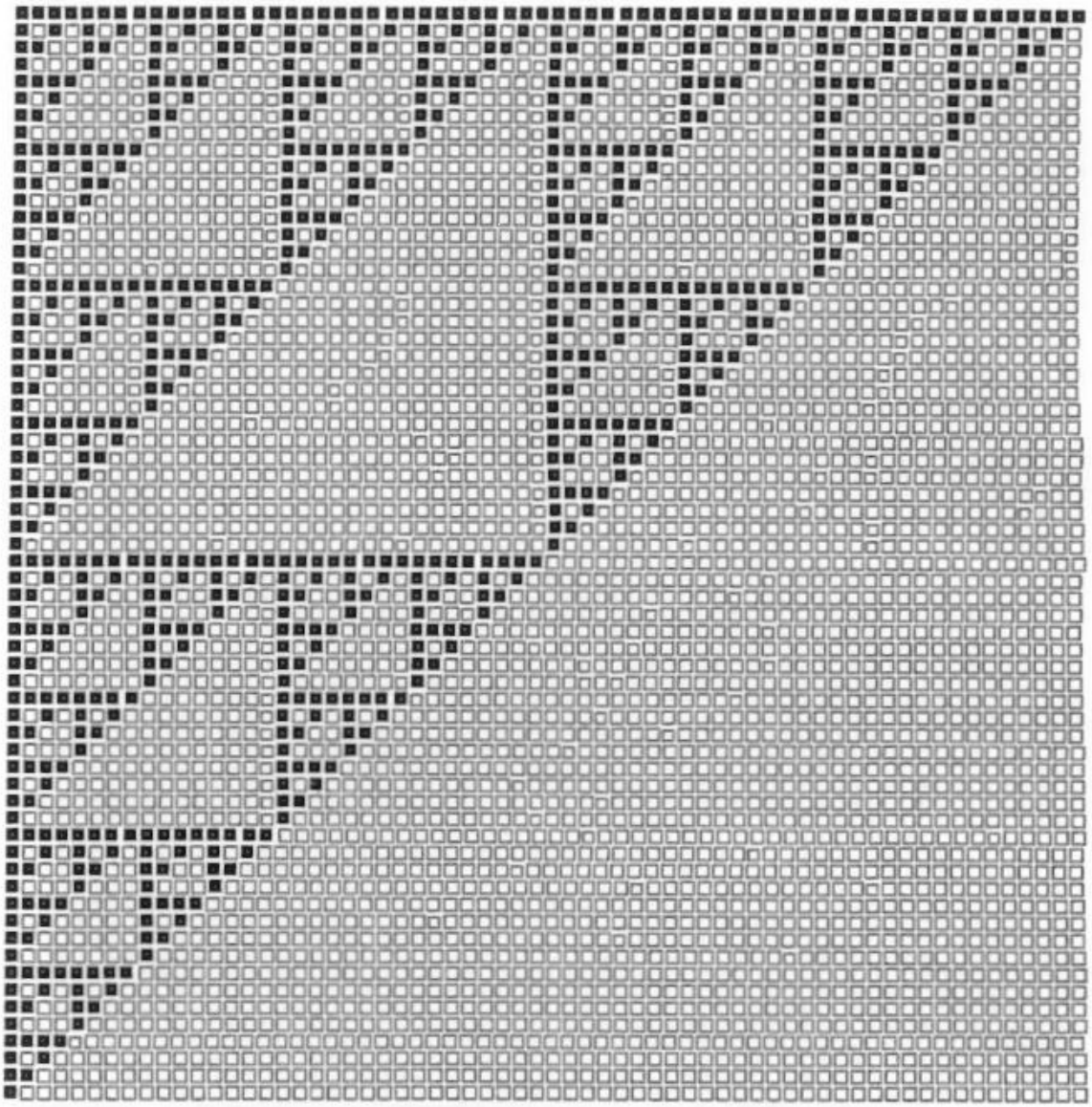
Matrice H_{64} de Hadamard (les carrés remplis codent 1, les vides −1) :



Dans les matrices H, le rapport du nombre de 1 au nombre total d'éléments reste à peu près constant (1/2 pour N assez grand); alors, on ne peut pas espérer, même avec un codage binaire, diminuer le nombre d'opérations élémentaires (changements de signe et additions).

On voit sur la figure relative à H_{64}, que les matrices H pourraient se décomposer par des sommes modulo 2 de matrices G de tailles différentes (comme des poupées-gigogne).

Géniton G_{64} :



On voit, sur une telle représentation, que le nombre de bits non nuls chute drastiquement dans les matrices G - en conséquence de la dimension fractale : le rapport de remplissage vaut 3/4 pour G_2 mais $(3/4)^2$ pour G_4 et $((3/4)^2)^2$ pour G_{16}, etc... et tend vers 0 quand N tend vers l'infini.

Cette remarque a excité une recherche (par compression algorithmique) des algorithmes minimaux non seulement pour effectuer des transformations orthogonales ou trines, mais aussi, pour obtenir un algorithme d'inversion matricielle généralisée, capable d'inverser des matrices même à déterminant nul, trouvant automatiquement les sous-bases vectorielles, donc les vecteurs linéairement indépendants.

A noter que les matrices G se construisent avec le seul Ou Exclusif (voir, par exemple, la seule règle non physiquement "ad hoc" des automates cellulaires, portant le numéro 90 dans la classification de Wolfram).

Toutes les matrices G se contiennent elles-mêmes... Elles ont pour équivalent, dans le continu, des courbes en cloche - les ***gaussiennes*** - à toutes les échelles quantifiées possibles : en effet, les génitons sont les équivalents, en algèbre entière modulo 2, de triangles de *Pascal* en algèbre entière non modulo, lesquels contiennent, dans leurs colonnes, les coefficients du binôme dont les enveloppes "continues" ont bien la forme de la courbe de *Gauss* :

On peut obtenir la dernière figure, colonne après colonne, à partir de la droite et en allant vers la gauche, comme les coefficients du développement (modulo 2) en puissances successives, ici de 0 à 63 mais on pourrait itérer à l'infini, du binôme X+Y (ou du binôme X−Y). Pour la puissance 0, on obtient un seul 1 comme coefficient; pour la puissance 1, on obtient 1 1; pour la puissance 2, on obtient : 1 0 1 (c'est-à-dire 1 2 1 modulo 2); pour la puissance 3, on obtient 1 1 1 1 (c'est-à-dire 1 3 3 1 modulo 2); pour la puissance 4, on obtient 1 0 0 0 1 (c'est-à-dire 1 4 6 4 1 modulo 2), et ainsi de suite.

L'invariance de la forme de la *gaussienne* par **transformation de Fourier** a joué un rôle dans les raisonnements qui ont conduit à développer la présente théorie; il suffisait en effet de se servir du principe d'invariance et de se poser la question : "*Quel est, en algèbre discrète entière modulo 2 (isomorphe de l'algèbre logique), l'équivalent de la* ***gaussienne****, fonction continue ?*"

Sachant que le raisonnement intermédiaire passe par la discrétisation de gaussiennes de taille de plus en plus grande, on tombe, nécessairement, sur le triangle de Pascal dont la parité (reste de la division entière par 2) est aussi une matrice G, un géniton. Sachant que l'inverse de Fourier d'une gaussienne normalisée peut être elle-même, mais aussi, car on ne verrait pas la différence, (la courbe en cloche étant symétrique) son retournement vertical, on comprend mieux, maintenant, que les inverses matricielles de tous les génitons, présentés comme sur la figure ci-dessus, soient aussi des retournements d'eux-mêmes : leur symétrique par rapport à leur centre, pour tous les rangs même non puissances de 2. Il était connu (voir par exemple la réf. **[Iverson]**), sans qu'on ait établi le rapport avec Hadamard, Gauss, Fourier et, les fractals, que les retournements verticaux des matrices G, pour un rang puissance de 2, étaient auto-inverses modulo 2. Cette propriété dormait comme une curiosité mathématique; il ne reste plus qu'à s'en servir... en traitement du signal, en physique et en biologie.

Genèse et propriétés du dernier algorithme

Puisque le nombre d'opérations relatives reste constant lorsque N augmente, dans le cas des matrices G, on peut parvenir à la **quatrième simplification**, drastique, en développant analytiquement les produits matriciels à l'aide de polynômes de Boole et en simplifiant; on utilisa un ordinateur pour produire un modèle *formel* (en notation d'Iverson), ce qui permit de mettre au point l'algorithme minimal exposé plus haut (appelé l'*algorithme du prestidigitateur ou du magicien*); cet algorithme a pour propriété non seulement d'utiliser une seule application *logique*, non productrice d'**entropie** (car, par application de l'algorithme sur le résultat, on retrouve le signal d'origine intact, quelle que soient la complexité et la dimension de ce dernier), mais aussi d'opérer à **volume constant** (de mémoire nécessaire). La réunion de ces deux conditions (que l'on trouve aussi réalisée dans les transformations de seconde espèce en physique du solide) constitue une *preuve - thermodynamique* - indépendante, de son *optimalité*.

Il convient de répéter que la progression des simplifications algorithmiques présentées (avec exposé de l'algèbre de Hadamard) n'a ici qu'un but essentiellement *pédagogique*, celui de se raccorder à quelque chose de connu et d'aisément vérifiable, par petites étapes - comme des leçons successives.

Toutefois, ces transformations n'ont pas été trouvées ainsi, mais par CEA (Compression Expérimentale Algorithmique, comme exposé dans des articles antérieurs, puis par étude des propriétés de ≠\ la *propagation asymétrique de la parité*, noyau commun trouvé à l'issue de cette CEA, pour toute évolution de l'information quel que soit l'algorithme initial, habituellement considéré numériquement; on a tout décomposé en binaire pur jusqu'à **simplification ultime**. Le rôle joué par la notation d'Iverson a été déterminant; on se rend compte que cette recherche (qui a duré 15 ans) aurait été non pas impossible mais fort difficile, si l'on n'avait pas disposé d'*APL* qui permet de reconsidérer le calcul matriciel par *symétrie*; en outre, il existe des implantations puissantes de cette notation sur ordinateur - certaines sont gratuites; elles ont permis de tester des modèles a priori fort compliqués à dominer s'il avait fallu les étudier en FORTRAN, PASCAL, ADA, LISP, PROLOG ou même en C++, par exemple.

K. E. Iverson, A Programming Language, J. Wiley & Sons, New York (1962), pp. 251-5, spécialement p. 252. (1962).

Note et remerciements.

La première démonstration mathématique - par les polynômes de Boole - que les matrices G, retournées verticalement, étaient les opérateurs involutifs matriciels de la transformation que j'avais appelée *cognitive*, (opérateurs que je cherchais alors à identifier en 1990 dans le cas général) est due à **Claude Cortet**; qu'il en soit ici chaleureusement remercié, car cette preuve a induit une recherche intense sur le sujet, et a constitué le pivot de nouvelles hypothèses concernant, entre autres, la modélisation des mécanismes relatifs au traitement de l'information par nos sens et par notre cerveau, ainsi que la conservation (optimale) de notre patrimoine génétique dans la molécule d'ADN. (Le terme *hélicoïdal,* pour l'une des deux transformations, provient de l'étude des structures de données en hélices).

Annexe. Petit conte mathématico-naval

Reprenons le signal que nous avons appelé B précédemment :

```
0 0 0 1 0 1 0 0 1 1 0 1 0 1 1 1
```

et agrémentons-le de ses différences finies, écrites comme une nouvelle ligne de 15 bits sous chacune des 15 paires de bits adjacents : cette différence finie binaire vaut 1 si les deux bits de la paire sont différents, et 0 s'ils ne sont pas différents : ce mécanisme sait faire la différence (dans tous les sens de cette expression en français); la *différence qui crée une différence* est la base du calcul différentiel en logique pure :

```
0 0 0 1 0 1 0 0 1 1 0 1 0 1 1 1
 0 0 1 1 1 1 0 1 0 1 1 1 1 0 0
```

Cette nouvelle ligne est encore un signal binaire. Si le signal B contenait une suite de micro-gènes soit dominants (1) soit récessifs (0), on aurait effectué une différenciation génétique (et, certainement pas *au hasard*).

Mais le propre de la différenciation est de ne jamais s'arrêter, sauf faute de combattants (le mot "différenciel" se différencia lui-même en "différentiel" mathématique); il s'agit d'un phénomène dynamique; si cette ligne de 15 bits est la différentielle première d'un patrimoine micro-génétique ou signal initial B, on peut prendre à nouveau les 14 différences finies des 15 paires adjacentes, pour obtenir le petit-fils de B, ou, mathématiquement, l'équivalent de sa différentielle seconde :

```
0 0 0 1 0 1 0 0 1 1 0 1 0 1 1 1
 0 0 1 1 1 1 0 1 0 1 1 1 1 0 0
  0 1 0 0 0 1 1 1 1 0 0 0 1 0
```

On a déjà l'équivalent de ce qu'en mécanique on nomme un Hamiltonien (mais sir Rowan Hamilton ne connaissait ni les codages binaires ni *APL* ni la génétique; il ne travaillait, par force, que sur des forces; force est de le reconnaître).

Pourquoi s'arrêter en si bon chemin et postuler que l'on n'a pas besoin de dérivées - différentielles ou différences finies - autres que les deux premières, pour comprendre ou modéliser une modulation quelconque de signal (ou une variation de fonction quelconque, laquelle apparaît continue si on ne voit pas, à l'aide de la lentille de *Fresnel* adéquate pour focaliser le faisceau du phare, le détail de sa modulation, *bit* après *bit*, *photon* après *photon*) ? Deux différentielles, c'est trop !

Une expérience, filmée en video par **Jean Delaunay**, de l'AFAPL - qu'il en soit ici remercié - a montré qu'un enfant de 6 ans, ignorant tout des mathématiques, était capable sur un signal assez long, matérialisé par des cubes LEGO de couleurs différentes, en l'occurrence rouges et jaunes, de construire la séquence des différences finies successives, absolument équivalente à un système différentiel d'ordre N-1 si N est la dimension (nombre de bits) du signal initial : il suffit de dire à l'enfant : "*si c'est pareil, tu mets un cube jaune, et si ça n'est pas pareil, tu mets un cube rouge*".

L'enfant, satisfait des compliments qu'on lui prodigua à juste titre lorsqu'il finit de placer, sans se tromper, le dernier cube jaune de la différentielle d'ordre 15 en bas de son joli triangle, se retourna alors spontanément vers sa s urette de 4 ans et demi, pour lui expliquer ce nouveau jeu et la guider avec succès dans l'apprentissage... (La bande en SECAM fut convertie en NSTC et présentée à la journée *APL* "**Tool of Thought VIII**" de SIGAPL, New York, en janvier 1993). Faisons comme lui, pour produire le modèle mathématique appelé, à cause de sa forme, et de son aspect décoratif quand il est en couleurs, le *fanion* d'un signal :

```
0 0 0 1 0 1 0 0 1 1 0 1 0 1 1 1
 0 0 1 1 1 1 0 1 0 1 1 1 1 0 0
  0 1 0 0 0 1 1 1 1 0 0 0 1 0
   1 1 0 0 1 0 0 0 1 0 0 1 1
    0 1 0 1 1 0 0 1 1 0 1 0
     1 1 1 0 1 0 1 0 1 1 1
      0 0 1 1 1 1 1 1 1 0 0
       0 1 0 0 0 0 0 0 1 0
        1 1 0 0 0 0 0 1 1
         0 1 0 0 0 0 1 0
          1 1 0 0 0 1 1
           0 1 0 1 0
            1 1 1 1
             0 0 0
              0 0
               0
```

On peut lire dans les *alignements* d'un fanion des informations précieuses, par exemple **H** sur le bord gauche (tribord), de haut en bas, et **C**, la transformée cognitive de B, sur le bord droit (bâbord) de bas en haut; (si ces termes de marine paraissent ici inversés, c'est parce que l'étrave du fanion gît en bas... produisant un soliton binaire, son sillage, dont on verra, *in fine*, l'expression mathématique).

Notre petit mousse, mû par une brise d'instinct cognitif caressant ses jeunes neurones, et guidé par ses récepteurs rétiniens (disposés, dans l' il, comme par hasard, avec cette même symétrie ternaire), a construit, sans le savoir, non seulement UN système d'équations aux différences finies binaires, mais TROIS : En effet, chaque 1 ou 0 interne, est, partout, non seulement la différence finie des deux parités situées *au-dessus*, mais aussi la différence finie des deux parités situées *à babord* et *à tribord*, parallèlement aux bords obliques du fanion.

Résoudre un système d'équations différentielles consiste à **intégrer** le système pour trouver une fonction d'évolution inconnue, alors qu'on connaît seulement une série de différentielles premières (un Jacobien en mathématiques) ou aussi - ce qui vaut mieux avec des nombres (mais est superflu ici comme on va le montrer) - une série de différentielles secondes (un Hessien en mathématiques). Que l'on parle d'Hamiltoniens, de Laplaciens ou de gaussiennes, ces constructions de l'esprit humain n'aboutissent pas, dans la majorité des cas, à des systèmes **intégrables**, loin s'en faut ! (Voir, par exemple ce qui se passe avec l'équation de Schrödinger en mécanique quantique, ou avec la théorie des réseaux neuronaux.)

Alors, brisons-là, et cassons le fanion :

S'il ne reste que la seconde ligne 0 0 1 1 1 1 0 1 0 1 1 1 1 0 0 (la première suite de différences finies binaires), comment peut-on reconstituer le signal B c'est-à-dire 0 0 0 1 0 1 0 0 1 1 0 1 0 1 1 1 ?

Par intégration, donc par une *somme cumulée*, mais modulo 2, comme il se doit.

Devenu loup de mer après passage par l'Ecole Navale, notre petit mousse, le gaillard d'avant, aura peut-être eu vent du large et d'*APL*...; il pourra vérifier ceci sur son portable, embarqué pour fuir la morosité des jours de calme plat dans le Triangle des Bermudes, et ne pas perdre la boussole :

```
      ≠\0 0 1 1 1 1 0 1 0 1 1 1 1 0 0
0 0 1 0 1 0 0 1 1 0 1 0 1 1 1
```

signal auquel il suffit de concaténer, à gauche donc à bâbord, la constante d'intégration aux Grandes Ecoles, soit 0 soit 1 selon le sens du vent, pour obtenir B cherché (ici, on attache 0). Si l'on attachait 1, on obtiendrait :

1 0 0 1 0 1 0 0 1 1 0 1 0 1 1 1 dont la différence finie débuterait par 1, ce qui n'est pas le cas dans le fanion.

Il convient de remarquer que la séquence 1 1 1 0 1 0 1 1 0 0 1 0 1 0 0 0 c'est-à-dire 1 attaché à la négation logique du résultat trouvé par intégration, en *APL* également : 1, 1≠≠\ 0 0 1 1 1 1 0 1 0 1 1 1 1 0 0 est l'autre solution possible pour l'intégration lorsqu'il manque UN bit d'information **à gauche**.

Tel un *hologramme*, un fanion brisé se reconstitue, de proche en proche, dans tous les sens du vent, noroît comme suroît, zéphyr comme aquilon, par intégration binaire, à condition qu'il reste, par ci par là, un minimum d'information (dans l'exemple choisi, 16 bits indépendants), après une tempête *chaotique* destructrice. Voilà d'ailleurs un jeu amusant pour l'été. Cette propriété de mémoire holo-graphique est absolument remarquable : elle explique, les yeux fermés, la tache aveugle, point de sortie de l'information vers le nerf optique, dans la rétine, et la célèbre illusion qui en résulte : le cerveau reconstitue effectivement l'information manquante, sans se fatiguer, à partir du contexte: si ce dernier est tout blanc, il choisit 0, même s'il y a un point noir sur la feuille de papier, ou, réciproquement à partir du contexte noir (il choisit 1) même s'il y a un point blanc.

L'heure est venue de faire le point et de coucher, sur parchemin d'oubli, la vieille lune amère des sargasses et sarcasmes mathématiques, car le soleil point à l'aube de l'ère de ≠\ le verre luisant comme un spath à mille facettes, qui se profile et se faufile partout, binant le binaire, carottant dans ses fanes, ions positifs ou négatifs, fouissant et jouissant à l'envi dans ces triangles d'or et de rêve nouveaux venus, à poings fermés et à point nommé.

≠\