Compression d'images par ondelettes

Benjamin CATINAUD

Année 2017-2018

Abstract

Nowadays, million of pictures are sent every minute. Consequently, we have to create algorithms to compress this data. To this end, several solutions exist. The most famous is perhaps the Fast Fourier Transform, use by scientists. However, this transform take care of frequencies in a signal but doesn't take its chronology into account. To adress this issue, I suggest to study the Wavelet Transform. This paper will detail basics of wavelet theory, as well as algorithms to transform and rebuild pictures with wavelets and, finally, discuss their efficiency.

Introduction

Aujourd'hui où des millions d'images sont envoyées chaque minute, une méthode pour compresser ces données, situées à l'interface ténue entre humain et machine et témoignant des interactions humaines, est par conséquent requise. Cette méthode devra optimiser rapidité, homogenéité, integrité et cohésion de ces informations.

Pour ce faire, plusieurs solutions peuvent être mises en place. Typiquement, les physiciens utilisent dans la plus grande majorité des cas la transformation FFT (pour Fast Fourier Transform) telle que pour une fonction f supposée intégrable, la transformation de Fourier s'écrit :

$$\mathcal{F}(f) : \xi \mapsto \hat{f}(\xi) = \int_{-\infty}^{+\infty} f(x)\, \mathrm{e}^{-\mathrm{i}\xi x}\, \mathrm{d}x.$$

Cependant, en conséquence du terme $\mathrm{e}^{-\mathrm{i}\xi x}$, la transformée de Fourier permet de bien distinguer les fréquences d'un signal mais pas leur chronologie. Cette particularité peut être problématique dans certains cas lors de traitements du signal comme le cas de signaux non sinusoïdaux, notamment pour les images. Ainsi, nous choisirons de travailler avec la transformation par ondelettes où les ondelettes sont des fonctions de carré intégrable, à support compact et d'intégrale nulle.

La théorie des ondelettes, mise au point dans les années 1980 par plusieurs chercheurs dont notamment, Yves Meyer et Jean Morlet, a connu ainsi un rapide succès que ce soit dans le traitement de signaux sismiques (domaine qui a initié la recherche dans cette théorie) ou dans le traitement des images.

L'expression générale de la transformation par ondelettes pour un signal $x \in L^2(\mathbb{R}, \mathbb{R})$ s'écrit :

$$g(a,b) \;=\; \frac{1}{\sqrt{a}} \int_{-\infty}^{+\infty} x(t) \psi\Big(\frac{t-b}{a}\Big)\, \mathrm{d}t$$

Le paramètre a est ainsi associé à une dilatation, a est donc appelé le paramètre d'échelle. Quant à lui, le paramètre b est associé à une translation, permettant ainsi de tenir compte de la chronologie des différents signaux.

Ce travail va donc s'appuyer sur la théorie des ondelettes afin de répondre à la question de comment compresser des images en utilisant à profit cette théorie. Dans un premier temps, nous introduirons plusieures notions utiles dans la suite ainsi que d'une bibliothéque, créée pour l'occasion, permettant de manipuler les vecteurs et les matrices. Dans un deuxième temps, après avoir donné la définition d'une ondelette et avoir défini l'analyse multirésolution, nous détaillerons l'algorithme de la transformation par ondelettes (abrégé dans la suite l'algorithme FWT) conçu par Stéphane Mallat. Enfin, nous discuterons de l'efficacité de cet algorithme du point de vue de la complexité ainsi que les pertes occasionnées.
Il est important de signaler que les fonctions relatives à l'algorithme FWT ont été traduites depuis du code $Matlab$.

I. Cadre de travail

1) Cadre mathématique

En règle générale, les ondelettes sont définies par des fonctions à valeurs complexes. Ici, nous nous intéresserons uniquement à des fonctions à valeurs réelles.

Définition 1 (i-ième forme coordonnée)

Soit E un $\mathbb{K}$-espace vectoriel de dimension finie n.
Soit $B = (e_i)_{i\in[1;n]}$ une base de E
On appelle i-ième forme coordonnée la fonction $e_i^* \in E^*$:

$$\begin{aligned} e_i^* \ : \ E &\mapsto \mathbb{K} \\ \sum_{j=1}^{n} x_j e_j &\longmapsto x_i \end{aligned}$$

Définition 2 (Support d'une fonction)

Soit $f : I \to \mathbb{R}$
On appelle support de f l'ensemble fermé $\mathrm{Supp}(f) = \overline{\{x \in I, f(x) \neq 0\}}$

Propriété 1 (Produit scalaire canonique sur $M_{n,m}(\mathbb{R})$)

La fonction $<,>$: $M_{n,m}(\mathbb{R})\, x M_{n,m}(\mathbb{R}) \longrightarrow \mathbb{R}$ est un produit scalaire sur $M_{n,m}(\mathbb{R})$

$$(M, N) \longmapsto \sum_{i=1}^{n}\sum_{j=1}^{n} M_{i,j}N_{i,j}$$

Propriété 2 (Norme euclidienne)

Soit $(E, <,>)$ un espace préhilbertien réel.
La fonction $\| \ . \ \|$: $E \longrightarrow \mathbb{R}$, $x \longmapsto \sqrt{< x, x >}$ est une norme sur E.
Cette norme est appelée norme euclidienne.

Définition 3 (Fonction de carré intégrable)

Soit I un intervalle de $\mathbb{R}$. Soit $f : I \to \mathbb{R}$.
On dit que f est de carré intégrable (au sens de Lebesgue) lorsque
$\int_I |f|^2 \ \in \mathbb{R}$.

Propriété 3 (Espace $L^2(I, \mathbb{R})$)

Soit I un intervalle de $\mathbb{R}$.
L'espace des fonctions de carré intégrable forme un espace vectoriel. On le note $L^2(I, \mathbb{R})$.

Propriété 4 (Produit scalaire canonique sur $L^2(I, \mathbb{R})$)

Soit I un intervalle de $\mathbb{R}$.
La fonction $<,>$: $L^2(I,\mathbb{R})$ x $L^2(I,\mathbb{R}) \to \mathbb{R}$, $(f, g) \mapsto \int_I fg$ est un produit scalaire.

Définition 4 (Famille orthogonale, famille orthonormale)

Soit $(E, <,>)$ un espace préhilbertien réel.
Soit $(f_i)_{i\in I}$ une famille de vecteurs de E .
On dit que $(f_i)_{i\in I}$ est orthogonale lorsque
$\forall i, j \in I, i \neq j \ \Rightarrow < f_i, f_j > = \ 0.$
Lorsque de plus, $\forall i, j \in I, < f_i, f_j > = \ \delta_{ij}$, on dit que la famille $(f_i)_{i\in I}$ est orthonormale.

Remarque 1

Dans la suite nous indexerons les familles par $I = \mathbb{Z}^2$.
On dira alors que la famille $(f_{i,j})_{(i,j)\in\mathbb{Z}^2}$ est bi-orthonormale lorsque
$\forall (i, j), (k, l) \in \mathbb{Z}^2, < f_{i,j}, f_{k,l} > = \ \delta_{ik}\delta_{jl}$

2) Modules $Vect$ et $Matrix$

Les fonctions *OCaml* qui seront écrites ont été traduites depuis le langage $Matlab$, langage pouvant faire de nombreuses opérations haut niveau sur les vecteurs et sur les matrices. Il est donc nécessaire d'écrire des modules permettant de faire les mêmes opérations. Ces modules sont conçus pour être les plus généraux possibles (on cherchera à avoir la signature $'a$). cf Annexe B pour le code.

Module $Vect$

Ce module permet de représenter des vecteurs d'un espace vectoriel par l'intermédiaire du type $'a\ array$. Il est constitué de fonctions de création, de fonctions correspondant aux opérations licites issues de la structure d'espace vectoriel. La fonction *prod v1 v2 prod_elem* permet de faire le produit coefficient par coefficient de deux vecteurs. Ce module possède également des fonctions permettant d'ajouter des zéros entre les coefficients (*dilate v s zero*) ou encore d'insérer (*put_in v1 v2*) ou d'extraire des vecteurs (*sub_vect v i j*).

Module $Matrix$

Ce module, quant à lui, permet de représenter les matrices par le type $'a\ Vect.vect\ Vect.vect$. Il est constitué des mêmes fonctions que le module $Vect$, à différence notable l'ajout de fonctions permettant de manipuler lignes et colonnes.

Module $Pixel\_matrix$

Enfin, ce dernier module permet de convertir des images sous le format BMP en matrices et inversèment.

II. Algorithme FWT

1) Une ondelette, c'est quoi ?

Définition 5 (Ondelette mère et ondelettes filles)

Soit $\psi \in L^2(\mathbb{R},\mathbb{R})$ *à support compact et telle que* $\int_{\mathbb{R}} \psi$ *converge et vaut* 0.

Soit $(\psi_{j,k})_{(j,k)\in\mathbb{Z}^2}$ *une famille de fonctions de* $L^2(\mathbb{R},\mathbb{R})$ *définie par :*

$$\forall (j,k) \in \mathbb{Z}^2, \forall x \in \mathbb{R}, \psi_{j,k}(x) = \frac{1}{\sqrt{2^j}}\psi(\frac{2^j x - k}{2^j})$$

On dit alors que ψ *est l'ondelette mère et que la famille* $(\psi_{j,k})$ *issue de translations et de dilatations de l'ondelette mère forme la famille des ondelettes filles.*

Exemple Ondelette de Haar

Soit ψ la fonction définie par : $\psi = \mathbb{1}_{[0;\frac{1}{2}[} - \mathbb{1}_{[\frac{1}{2};1[}$

ψ étant constante par morceaux, $\int_{\mathbb{R}} \psi$ converge et vaut 0.

De plus, Supp(ψ) est clairement borné donc, comme $\mathbb{R}$ est de dimension finie, Supp(ψ) est compact.

La famille orthonormale associée à l'ondelette mère ψ est alors :

$$(\psi_{j,k})_{(j,k)\in\mathbb{Z}^2} \text{ où } \forall j,k \in \mathbb{Z}, \forall x \in \mathbb{R}, \psi_{j,k}(x) = \frac{1}{\sqrt{2^j}}\psi(\frac{x-2^j k}{2^j})$$

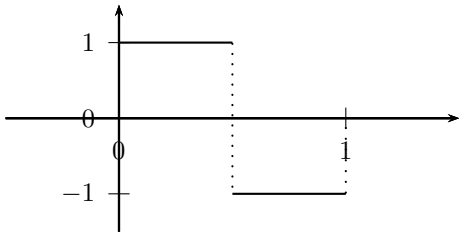

Figure 1: Courbe représentative de ψ

2) Analyse multirésolution

Définition 6 (Espace d'approximation)

Soit $j \in \mathbb{Z}$. On appelle espace d'aproximation à l'échelle 2^j l'espace vectoriel des fonctions de $L^2(\mathbb{R},\mathbb{R})$ constantes sur les intervalles de la forme $[2^j k; 2^j(k+1)[$, $k \in \mathbb{Z}$. On le note V_j.

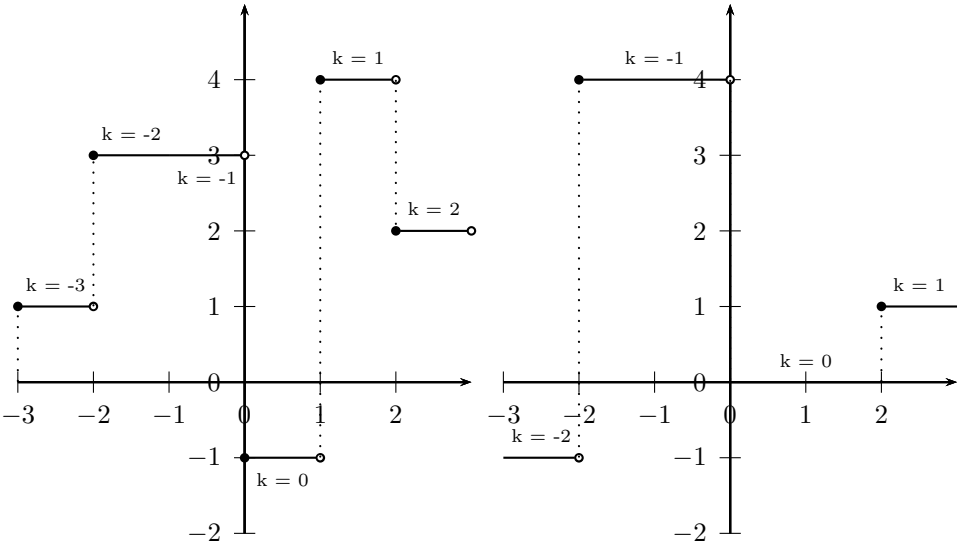

Figure 2: Exemple graphique d'une fonction de V_0 (à gauche) et de V_1 (á droite)

Propriété 5

Les espaces $(V_j)_{j\in\mathbb{Z}}$ vérifient les propriétés suivantes :

(*i*) $\forall j \in \mathbb{Z}, V_{j+1} \subseteq V_j$

(*ii*) $\forall j \in \mathbb{Z}, f \in V_j \Leftrightarrow \left(t \mapsto f\left(\frac{t}{2}\right)\right) \in V_{j+1}$

(*iii*) $\forall j,k \in \mathbb{Z}, f \in V_j \Leftrightarrow \left(t \mapsto f(t-2^j k)\right) \in V_j$

(*iv*) $\bigcup\limits_{j\in\mathbb{Z}} V_j$ *est dense dans* $L^2(\mathbb{R},\mathbb{R})$

i. e. $\forall \varepsilon > 0, \forall f \in L^2(\mathbb{R},\mathbb{R}), \exists j_\varepsilon \in \mathbb{Z}, \| f - P_{V_{j_\varepsilon}}(f) \| \leqslant \varepsilon$

(*v*) $\bigcap\limits_{j\in\mathbb{Z}} V_j = \left\langle \{t \longmapsto 1\} \right\rangle$

Demonstration 1

(*iv*) *Montrons la proprété (iv).*

Soit $\varepsilon > 0$. Soit $f \in L^2(\mathbb{R},\mathbb{R})$.

f est donc continue par morceaux sur des intervalles que l'on va noter $(D_n)_{n\in I}$ avec $I \subset \mathbb{N}$, I fini. De plus, comme f est de carré intégrable, on montre par l'absurde et en utilisant la définition de limite que $\lim\limits_{\pm\infty} f \in \mathbb{R}$.

On va donc limiter l'étude à un intervalle D_n pour $n \in I$.

On pose $D_n =]\alpha;\beta[$.

Cas 1 : $\alpha \neq -\infty$ et $\beta \neq +\infty$

Ainsi la fonction f est prolongeable par continuité sur $\overline{D_n} = [\alpha;\beta]$.

Par conséquent, f est continue sur un segment donc, par théorème de Heine, elle y est uniformément continue.

Par définition de continuité uniforme, on a :

$$\exists \delta_n > 0, \forall x, y \in D_n, \mid x - y \mid \leqslant \delta_n \ \Rightarrow \mid f(x) - f(y) \mid \leqslant \varepsilon$$

Or, d'autre part, $\lim_{j\to-\infty} 2^j = 0$ *et* $\lim_{j\to+\infty} 2^j = +\infty$ *donc*

$$\exists j_n \in \mathbb{Z}, 2^{j_n} \leqslant \delta_n < 2^{j_n+1}$$

Ainsi $\forall x, y \in D_n, |\, x-y \,| \leqslant 2^{j_n} \Rightarrow |\, f(x)-f(y) \,| \leqslant \varepsilon$.

Cas 2 : $\alpha \neq -\infty$ *ou* $\beta \neq +\infty$

Dans ce cas, on procéde de même en se ramenant à un segment en utilisant la définition de la limite appliquée à ε *en* $\pm\infty$.

Dans tous les cas, on pose alors une fonction $\psi_n \in V_{j_n}$ *telle que pour tout intervalle de la forme* $[2^{j_n}k; 2^{j_n}(k+1)[$ *où* $k \in \mathbb{Z}$, *il existe un* $x_{n,k} \in J_n$ *tel que* $\forall y \in [2^{j_n}k; 2^{j_n}(k+1)[$, $\psi_n(y) = f(x_{n,k})$ *(ce* $x_{n,k}$ *existe sous condition que* $[2^{j_n}k; 2^{j_n}(k+1)[\subset D_n$, *si ce n'est pas le cas, on pose alors* $\psi_n\,|_{[2^{j_n}k;2^{j_n}(k+1)[} = 0$*).*

On obtient donc un ensemble d'indices $J_n = \left\{ j_n, n \in I \right\}$ *et un ensemble de fonctions* $\left\{ \psi_n, n \in I \right\} \subset \bigcup_{j\in\mathbb{Z}} V_j$.

Par propriété d'inclusion (i), *en posant* $j_\varepsilon = \min J_n$ *(existe car* I *est fini), et en définissant* $P_{V_{j_\varepsilon}}$ *à l'aide des* ψ_n *pour* $n \in I$, *on obtient que :*

$$\boxed{\bigcup_{j\in\mathbb{Z}} V_j \text{ est dense dans } L^2(\mathbb{R},\mathbb{R})}$$

Ainsi, la dernière propriété montre que toute fonction de $L^2(\mathbb{R},\mathbb{R})$ est limite d'une suite d'éléments de $\bigcup_{j\in\mathbb{Z}} V_j$. C'est à dire que toute fonction peut être approximée par des fonctions des espaces V_j, $j \in \mathbb{Z}$.

Propriété 6 (Caractérisation des espaces d'approximation)

On a :

$$\forall j \in \mathbb{Z}, V_j = \left\langle \left\{ \phi_{j,k} = \tfrac{1}{\sqrt{2^j}} \mathbb{1}_{[2^j k; 2^j(k+1)[},\ k \in \mathbb{Z} \right\} \right\rangle$$

La famille $(\phi_{j,k})_{(j,k)\in\mathbb{Z}^2}$ *est une famille d'ondelettes filles issue de l'ondelette mère* $\phi = \mathbb{1}_{[0;1[}$.

L'idée de la transformation par ondelettes est ainsi de calculer les projections d'une fonction $f \in L^2(\mathbb{R},\mathbb{R})$ sur les espaces V_j pour un $j \in \mathbb{Z}$ donné. Plus le j diminue, plus l'approximation ainsi faite est précise.

Propriété 7 (Espace d'approximation et bi-orthonormalité)

La famille $(\phi_{j,k})_{(j,k)\in\mathbb{Z}^2}$ *de la propriété précédente est une famille bi-orthonormale.*

Propriété 8 (Projection d'une fonction sur V_j)

Soit $j \in \mathbb{Z}$. Soit $f \in L^2(\mathbb{R}, \mathbb{R})$.

D'après les deux propriétés précédentes, la projection de f sur V_j est donnée par :

$$P_{V_j}(f) = \sum_{k \in \mathbb{Z}} < f, \phi_{j,k} > \phi_{j,k}$$
$$\text{où } < f, \phi_{j,k} > = \frac{1}{\sqrt{2^j}} \int_{2^j k}^{2^j (k+1)} f(t)\, \mathrm{d}t$$

Définition 7 (Espace des détails)

On définit la notion d'espace de détails W_j, pour $j \in \mathbb{Z}$, tel que

$V_{j-1} = V_j \overset{\perp}{\bigoplus} W_j$

Ainsi, par définition, la connaissance de W_j et V_j permettent d'obtenir V_{j-1}.

Propriété 9

On a les propriétés suivantes :

. $\forall j \in \mathbb{Z}, \forall\, \omega_j \in W_j,\ \omega_j$ *est constante sur les intervalles de la forme* $[2^{j-1}k; 2^{j-1}(k+1)[, k \in \mathbb{Z}$

. $\forall j, k \in \mathbb{Z}, \forall\, \omega_j \in W_j,\ < \omega_j, \phi_{j,k} > = 0$ *i.e.* $\frac{1}{\sqrt{2^j}} \int_{2^j k}^{2^j (k+1)} \omega_j(t)\, \mathrm{d}t = 0$

. $\forall j \in \mathbb{Z},\ W_j = < \{\psi_{j,k},\ k \in \mathbb{Z}\} >$
avec $\forall j, k \in \mathbb{Z}, \psi_{j,k} = \frac{1}{\sqrt{2^j}} (\mathbb{1}_{[2^{j-1}k; 2^{j-1}(k+1)[} - \mathbb{1}_{[2^{j-1}(k+1); 2^{j-1}(k+2)[})$. *De plus, la famille* $(\psi_{j,k})_{(j,k) \in \mathbb{Z}^2}$ *est une famille d'ondelettes.*

Remarque 2

On remarque une relation de récurrence entre les ondelettes $\psi_{j,k}$ et $\phi_{j,k}$ pour tout $k, j \in \mathbb{Z}$:

$$\phi_{j,k} = \frac{1}{\sqrt{2}} (\phi_{j-1,2k} + \phi_{j-1,2k+1})$$
$$\psi_{j,k} = \frac{1}{\sqrt{2}} (\phi_{j-1,2k} - \phi_{j-1,2k+1})$$

Cette remarque nous amène ainsi à l'algorithme de Mallat qui permet la transformation par ondelettes d'une image.

3) Algorithme de transformation par ondelettes ou Algorithme de Mallat

Dans toute la suite, pour illustrer les propos, nous utiliserons l'image suivante

Figure 3: Image originale

Définition 8 (Vecteurs associés à la transformation par ondelettes de Haar)
D'après la remarque précédente, on pose les vecteurs associés à la transformation par ondelettes de Haar suivants [1] *:*

$$\begin{aligned}\mathcal{H}_a &= \tfrac{1}{2}(1,1) \in \mathbb{R}^2 \text{ pour les espaces } (V_j)_{j\in\mathbb{Z}}\\ \mathcal{H}_d &= \tfrac{1}{2}(1,-1) \in \mathbb{R}^2 \text{ pour les espaces } (W_j)_{j\in\mathbb{Z}}\end{aligned}$$

Définition 9 (Fonction filtre)
Soit $n \in \mathbb{N}$. Soient $b, x \in \mathbb{R}^n$. Soit $a \in \mathbb{R}^*$.
On définit un nouveau vecteur $y \in \mathbb{R}^n$, image de x par la fonction $\Phi_{a,b}$ tel que :

$$\forall i \in [1;n],\ e_k^*(y) = \tfrac{1}{a}\sum_{i=1}^{k} e_i^*(b)e_{k-i}^*(x)$$

Propriété 10 (Cas particulier de la fonction sur l'espace d'approximation)
Pour l'algorithme que l'on codera, on considérera la fonction $\Phi_{1,\mathcal{H}_a}$ tel que pour tout $x \in \mathbb{R}^n$,

$$\begin{aligned}e_1^*(\Phi_{1,\mathcal{H}_a}) &= \tfrac{1}{2}e_1^*(x)\\ \forall j \in [2;n], e_j^*(\Phi_{1,\mathcal{H}_a}) &= \tfrac{1}{2}(e_{j-1}^*(x) + e_j^*(x))\end{aligned}$$

On note alors cette fonction Φ.

Propriété 11 (Cas particulier de la fonction sur l'espace de détails)
On considérera dans ce cas, la fonction $\Phi_{1,\mathcal{H}_d}$ tel que pour tout $x \in \mathbb{R}^n$,

$$\begin{aligned}e_1^*(\Phi_{1,\mathcal{H}_d}) &= \tfrac{1}{2}e_1^*(x)\\ \forall j \in [2;n], e_j^*(\Phi_{1,\mathcal{H}_d}) &= \tfrac{1}{2}(e_{j-1}^*(x) + e_j^*(x))\end{aligned}$$

On note alors cette fonction Ψ.

Avant d'étudier l'algorithme, il reste encore à définir une fonction permettant d'extraire un sous vecteur.

Définition 10 (Fonction d'échantillonage)
On note $\downarrow_k$, pour $k \in [1;n]$ la fonction suivante :

$$\begin{aligned}\downarrow_k : \mathbb{R}^n &\longrightarrow \mathbb{R}^{\lfloor n/k \rfloor}\\ (x_1, x_2, ..., x_n) &\longmapsto (x_k, x_{2k}, ..., x_{\lfloor n/k \rfloor k})\end{aligned}$$

Algorithme de Mallat Considérons une image sous forme de matrice $M \in M_{n,m}(\mathbb{R})$, où l'on supposera que n et m sont deux multiples d'une puissance de 2, quitte à la normaliser en ajoutant des zéros.

$M_{1,1}$	$M_{1,2}$	$M_{1,3}$	$M_{1,4}$	$M_{1,5}$	$M_{1,6}$	$M_{1,7}$	$M_{1,8}$
$M_{2,1}$	$M_{2,2}$	$M_{2,3}$	$M_{2,4}$	$M_{2,5}$	$M_{2,6}$	$M_{2,7}$	$M_{2,8}$
$M_{3,1}$	$M_{3,2}$	$M_{3,3}$	$M_{3,4}$	$M_{3,5}$	$M_{3,6}$	$M_{3,7}$	$M_{3,8}$
$M_{4,1}$	$M_{4,2}$	$M_{4,3}$	$M_{4,4}$	$M_{4,5}$	$M_{4,6}$	$M_{4,7}$	$M_{4,8}$
$M_{5,1}$	$M_{5,2}$	$M_{5,3}$	$M_{5,4}$	$M_{5,5}$	$M_{5,6}$	$M_{5,7}$	$M_{5,8}$
$M_{6,1}$	$M_{6,2}$	$M_{6,3}$	$M_{6,4}$	$M_{6,5}$	$M_{6,6}$	$M_{6,7}$	$M_{6,8}$
$M_{7,1}$	$M_{7,2}$	$M_{7,3}$	$M_{7,4}$	$M_{7,5}$	$M_{7,6}$	$M_{7,7}$	$M_{7,8}$
$M_{8,1}$	$M_{8,2}$	$M_{8,3}$	$M_{8,4}$	$M_{8,5}$	$M_{8,6}$	$M_{8,7}$	$M_{8,8}$

Pour calculer V_{-j} et W_{-j}, on applique la fonction filtre en premier lieu sur les lignes de M.
On applique alors les fonctions Φ et Ψ au vecteur $M_{i,[1;m/j]} = (M_{i,1}, M_{i,2}, ..., M_{i,m/j})$ puis on applique au résultat ainsi obtenu la fonction de sous-échantillonage $\downarrow_2$. images que l'on notera respectivement $\downarrow_2 \Phi^i$ et $\downarrow_2 \Psi^i$

[1] Rigoureusement il faudrait utiliser le scalaire $\frac{1}{\sqrt{2}}$, cependant, suite à des problèmes liés à l'intensité des pixels lors de la compression, apparemment liés au choix de ce scalaire, j'ai décidé d'utiliser le scalaire $\frac{1}{2}$. (Voir l'annexe A pour un exemple).

Ainsi la nouvelle matrice devient :

$(\downarrow_2 \Phi^1)_{1,1}$	$(\downarrow_2 \Phi^1)_{1,2}$	$(\downarrow_2 \Phi^1)_{1,3}$	$(\downarrow_2 \Phi^1)_{1,4}$	$(\downarrow_2 \Psi^1)_{1,1}$	$(\downarrow_2 \Psi^1)_{1,2}$	$(\downarrow_2 \Psi^1)_{1,3}$	$(\downarrow_2 \Psi^1)_{1,4}$
$(\downarrow_2 \Phi^2)_{2,1}$	$(\downarrow_2 \Phi^2)_{2,2}$	$(\downarrow_2 \Phi^2)_{2,3}$	$(\downarrow_2 \Phi^2)_{2,4}$	$(\downarrow_2 \Psi^2)_{2,1}$	$(\downarrow_2 \Psi^2)_{2,2}$	$(\downarrow_2 \Psi^2)_{2,3}$	$(\downarrow_2 \Psi^2)_{2,4}$
$(\downarrow_2 \Phi^3)_{3,1}$	$(\downarrow_2 \Phi^3)_{3,2}$	$(\downarrow_2 \Phi^3)_{3,3}$	$(\downarrow_2 \Phi^3)_{3,4}$	$(\downarrow_2 \Psi^3)_{3,1}$	$(\downarrow_2 \Psi^3)_{3,2}$	$(\downarrow_2 \Psi^3)_{3,3}$	$(\downarrow_2 \Psi^3)_{3,4}$
$(\downarrow_2 \Phi^4)_{4,1}$	$(\downarrow_2 \Phi^4)_{4,2}$	$(\downarrow_2 \Phi^4)_{4,3}$	$(\downarrow_2 \Phi^4)_{4,4}$	$(\downarrow_2 \Psi^4)_{4,1}$	$(\downarrow_2 \Psi^4)_{4,2}$	$(\downarrow_2 \Psi^4)_{4,3}$	$(\downarrow_2 \Psi^4)_{4,4}$
$(\downarrow_2 \Phi^5)_{5,1}$	$(\downarrow_2 \Phi^5)_{5,2}$	$(\downarrow_2 \Phi^5)_{5,3}$	$(\downarrow_2 \Phi^5)_{5,4}$	$(\downarrow_2 \Psi^5)_{5,1}$	$(\downarrow_2 \Psi^5)_{5,2}$	$(\downarrow_2 \Psi^5)_{5,3}$	$(\downarrow_2 \Psi^5)_{5,4}$
$(\downarrow_2 \Phi^6)_{6,1}$	$(\downarrow_2 \Phi^6)_{6,2}$	$(\downarrow_2 \Phi^6)_{6,3}$	$(\downarrow_2 \Phi^6)_{6,4}$	$(\downarrow_2 \Psi^6)_{6,1}$	$(\downarrow_2 \Psi^6)_{6,2}$	$(\downarrow_2 \Psi^6)_{6,3}$	$(\downarrow_2 \Psi^6)_{6,4}$
$(\downarrow_2 \Phi^7)_{7,1}$	$(\downarrow_2 \Phi^7)_{7,2}$	$(\downarrow_2 \Phi^7)_{7,3}$	$(\downarrow_2 \Phi^7)_{7,4}$	$(\downarrow_2 \Psi^7)_{7,1}$	$(\downarrow_2 \Psi^7)_{7,2}$	$(\downarrow_2 \Psi^7)_{7,3}$	$(\downarrow_2 \Phi^7)_{7,4}$
$(\downarrow_2 \Phi^8)_{8,1}$	$(\downarrow_2 \Phi^8)_{8,2}$	$(\downarrow_2 \Phi^8)_{8,3}$	$(\downarrow_2 \Phi^8)_{8,4}$	$(\downarrow_2 \Psi^8)_{8,1}$	$(\downarrow_2 \Psi^8)_{8,2}$	$(\downarrow_2 \Psi^8)_{8,3}$	$(\downarrow_2 \Psi^8)_{8,4}$

En deuxième lieu, on applique aux colonnes notées $M_{[1;n/j],j}$ de cette nouvelle matrice ces mêmes fonctions [2]:

$(\downarrow_2 (\Phi \circ \Phi)^1)_{1,1}$	$(\downarrow_2 (\Phi \circ \Phi)^1)_{1,2}$	$(\downarrow_2 (\Phi \circ \Phi)^1)_{1,3}$	$(\downarrow_2 (\Phi \circ \Phi)^1)_{1,4}$
$(\downarrow_2 (\Phi \circ \Phi)^2)_{2,1}$	$(\downarrow_2 (\Phi \circ \Phi)^2)_{2,2}$	$(\downarrow_2 (\Phi \circ \Phi)^2)_{2,3}$	$(\downarrow_2 (\Phi \circ \Phi)^2)_{2,4}$
$(\downarrow_2 (\Phi \circ \Phi)^3)_{3,1}$	$(\downarrow_2 (\Phi \circ \Phi)^3)_{3,2}$	$(\downarrow_2 (\Phi \circ \Phi)^3)_{3,3}$	$(\downarrow_2 (\Phi \circ \Phi)^3)_{3,4}$
$(\downarrow_2 (\Phi \circ \Phi)^4)_{4,1}$	$(\downarrow_2 (\Phi \circ \Phi)^4)_{4,2}$	$(\downarrow_2 (\Phi \circ \Phi)^4)_{4,3}$	$(\downarrow_2 (\Phi \circ \Phi)^4)_{4,4}$
$(\downarrow_2 (\Psi \circ \Phi)^5)_{5,1}$	$(\downarrow_2 (\Psi \circ \Phi)^5)_{5,2}$	$(\downarrow_2 (\Psi \circ \Phi)^5)_{5,3}$	$(\downarrow_2 (\Psi \circ \Phi)^5)_{5,4}$
$(\downarrow_2 (\Psi \circ \Phi)^6)_{6,1}$	$(\downarrow_2 (\Psi \circ \Phi)^6)_{6,2}$	$(\downarrow_2 (\Psi \circ \Phi)^6)_{6,3}$	$(\downarrow_2 (\Psi \circ \Phi)^6)_{6,4}$
$(\downarrow_2 (\Psi \circ \Phi)^7)_{7,1}$	$(\downarrow_2 (\Psi \circ \Phi)^7)_{7,2}$	$(\downarrow_2 (\Psi \circ \Phi)^7)_{7,3}$	$(\downarrow_2 (\Psi \circ \Phi)^7)_{7,4}$
$(\downarrow_2 (\Psi \circ \Phi)^8)_{8,1}$	$(\downarrow_2 (\Psi \circ \Phi)^8)_{8,2}$	$(\downarrow_2 (\Psi \circ \Phi)^8)_{8,3}$	$(\downarrow_2 (\Psi \circ \Phi)^8)_{8,4}$

$(\downarrow_2 (\Phi \circ \Psi)^1)_{1,1}$	$(\downarrow_2 (\Phi \circ \Psi)^1)_{1,2}$	$(\downarrow_2 (\Phi \circ \Psi)^1)_{1,3}$	$(\downarrow_2 (\Phi \circ \Psi)^1)_{1,4}$
$(\downarrow_2 (\Phi \circ \Psi)^2)_{2,1}$	$(\downarrow_2 (\Phi \circ \Psi)^2)_{2,2}$	$(\downarrow_2 (\Phi \circ \Psi)^2)_{2,3}$	$(\downarrow_2 (\Phi \circ \Psi)^2)_{2,4}$
$(\downarrow_2 (\Phi \circ \Psi)^3)_{3,1}$	$(\downarrow_2 (\Phi \circ \Psi)^3)_{3,2}$	$(\downarrow_2 (\Phi \circ \Psi)^3)_{3,3}$	$(\downarrow_2 (\Phi \circ \Psi)^3)_{3,4}$
$(\downarrow_2 (\Phi \circ \Psi)^4)_{4,1}$	$(\downarrow_2 (\Phi \circ \Psi)^4)_{4,2}$	$(\downarrow_2 (\Phi \circ \Psi)^4)_{4,3}$	$(\downarrow_2 (\Phi \circ \Psi)^4)_{4,4}$
$(\downarrow_2 (\Psi \circ \Psi)^5)_{5,1}$	$(\downarrow_2 (\Psi \circ \Psi)^5)_{5,2}$	$(\downarrow_2 (\Psi \circ \Psi)^5)_{5,3}$	$(\downarrow_2 (\Psi \circ \Psi)^5)_{5,4}$
$(\downarrow_2 (\Psi \circ \Psi)^6)_{6,1}$	$(\downarrow_2 (\Psi \circ \Psi)^6)_{6,2}$	$(\downarrow_2 (\Psi \circ \Psi)^6)_{6,3}$	$(\downarrow_2 (\Psi \circ \Psi)^6)_{6,4}$
$(\downarrow_2 (\Psi \circ \Psi)^7)_{7,1}$	$(\downarrow_2 (\Psi \circ \Psi)^7)_{7,2}$	$(\downarrow_2 (\Psi \circ \Psi)^7)_{7,3}$	$(\downarrow_2 (\Psi \circ \Psi)^7)_{7,4}$
$(\downarrow_2 (\Psi \circ \Psi)^8)_{8,1}$	$(\downarrow_2 (\Psi \circ \Psi)^8)_{8,2}$	$(\downarrow_2 (\Psi \circ \Psi)^8)_{8,3}$	$(\downarrow_2 (\Psi \circ \Psi)^8)_{8,4}$

[2] La matrice est séparée en deux uniquement pour plus de lisibilité

L'algorithme, pouvant être considéré comme une fonction notée TW, pour calculer la transformée par ondelettes sur les espaces V_{-j_V} et W_{-j_V} est donc le suivant :

Algorithme 1 Transformée par ondelettes d'une matrice

Entrée(s) une matrice $M \in M_{n,m}(\mathbb{R})$ et un entier $j_V \in \mathbb{N}$
  si n ne divise pas 2^{j_V} **alors**
    Ajouter $(n - n \,\%\, 2^{j_V})$ lignes de zéros à M
  fin du si
  si m ne divise pas 2^{j_V} **alors**
    Ajouter $(m - m \,\%\, 2^{j_V})$ colonnes de zéros à M
  fin du si
  pour $k = 0$ **à** $j_V - 1$ **faire**
    pour $i = 1$ **à** $n/2^k$ **faire**
      Affecter au vecteur $M_{i,[1;\frac{m}{2^{k+1}}]}$ le résultat de la fonction $(\downarrow_2 \circ \Phi)$ appliqué au vecteur $M_{i,[1;\frac{m}{2^k}]}$
      Affecter au vecteur $M_{i,[\frac{m}{2^{k+1}}+1;\frac{m}{2^k}]}$ le résultat de la fonction $(\downarrow_2 \circ \Psi)$ appliqué au vecteur $M_{i,[1;\frac{m}{2^k}]}$
    fin du pour
    pour $j = 1$ **à** $m/2^k$ **faire**
      Affecter au vecteur $M_{[1;\frac{n}{2^{k+1}}],j}$ le résultat de la fonction $(\downarrow_2 \circ \Phi)$ appliqué au vecteur $M_{[1;\frac{n}{2^k}],j}$
      Affecter au vecteur $M_{[\frac{n}{2^{k+1}}+1;\frac{n}{2^k}],j}$ le résultat de la fonction $(\downarrow_2 \circ \Psi)$ appliqué au vecteur $M_{[1;\frac{n}{2^k}],j}$
    fin du pour
  fin du pour
Sortie(s) La matrice M ainsi obtenue

Avec l'image témoin, on obtient l'image suivante pour V_{-1} i.e. $j_V = 1$:

Figure 4: Transformée par ondelettes de l'image témoin pour $j_V = 1$

Suite à l'application de l'algorithme pour $j_V \in \mathbb{N}$, on obtient une matrice de la forme :

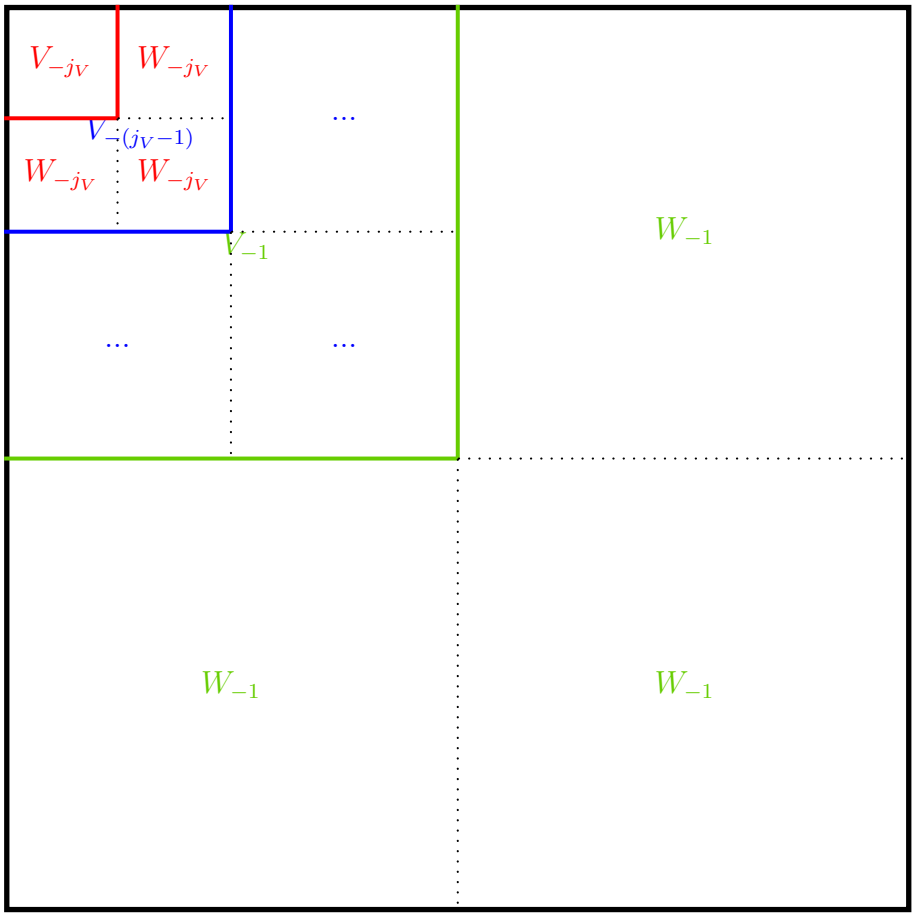

Figure 5: Schéma de la matrice issue de l'algorithme précédent pour $j_V \in \mathbb{N}$

Définition 11 (Fonction de dilatation)
On note $\uparrow_2$ la fonction suivante :

$$\begin{aligned} \uparrow_2 : \mathbb{R}^n &\longrightarrow \mathbb{R}^{2n} \\ (x_1, x_2, ..., x_n) &\longmapsto (x_1, 0, x_2, 0, ..., 0, x_n) \end{aligned}$$

Définition 12 (Vecteurs associés à la reconstruction)
D'après la remarque précédente, on pose les vecteurs associés à la reconstruction suivants :

$$\begin{aligned} \mathcal{R}_a &= (1,1) \in \mathbb{R}^2 \text{ pour les espaces } (V_j)_{j\in\mathbb{Z}} \\ \mathcal{R}_d &= (1,-1) \in \mathbb{R}^2 \text{ pour les espaces } (V_j)_{j\in\mathbb{Z}} \end{aligned}$$

Définition 13 (Fonctions filtres associés à la reconstruction)
De manière similaire aux fonctions filtres définies dans les propriétés 8 et 9, on définit les fonctions filtres de reconstruction suivantes :

$$\begin{aligned} \Gamma &= \Phi_{1,\mathcal{R}_a} \\ \Omega &= \Phi_{1,\mathcal{R}_b} \end{aligned}$$

D'après la définition 6, comme $\forall j \in \mathbb{Z}, V_{j-1} = V_j \overset{\perp}{\bigoplus} W_j$, on en déduit l'algorithme de reconstruction, pouvant être considéré comme une fonction notée R, suivant :

Algorithme 2 Reconstruction d'une matrice

Entrée(s) une matrice $M \in M_{n,m}(\mathbb{R})$ et un entier $j_V \in \mathbb{N}$
  pour $k = j_V - 1$ **à** 0 **faire**
    pour $j = 1$ **à** $m/2^k$ **faire**
      Affecter au vecteur $M_{[1;\frac{n}{2^k}],j}$ la somme du résultat de la fonction $(\Gamma \circ \uparrow_2)$ appliquée au vecteur $M_{[1;\frac{n}{2^{k+1}}],j}$ et du résultat de la fonction $(\Omega \circ \uparrow_2)$ appliquée au vecteur $M_{[\frac{n}{2^{k+1}}+1;\frac{n}{2^k}],j}$
    fin du pour
    pour $i = 1$ **à** $n/2^k$ **faire**
      Affecter au vecteur $M_{i,[1;\frac{m}{2^k}]}$ la somme du résultat de la fonction $(\Gamma \circ \uparrow_2)$ appliquée au vecteur $M_{i,[1;\frac{m}{2^{k+1}}]}$ et du résultat de la fonction $(\Omega \circ \uparrow_2)$ appliquée au vecteur $M_{i,[\frac{m}{2^{k+1}}+1;\frac{m}{2^k}]}$
    fin du pour
  fin du pour
Sortie(s) La matrice M ainsi obtenue

Remarque 3
La somme mentionnée permet de passer de deux vecteurs $(x_1, x_2, .., x_n)$ et $(y_1, y_2, ...y_n)$ au vecteur $(x_1, y_1, x_2, y_2, ..., x_n, y_n)$.

Propriété 12
Soit $j \in \mathbb{N}$. La fonction TW_j est inversible et $R_j = (TW_j)^{-1}$.

Demonstration 2
Soit $(x, y) \in \mathbb{R}^2$. Par définition des algorithmes, il s'agit en fait de démontrer que :

$$\begin{aligned} \Phi_{1,\mathcal{R}_a}\Big(\Phi_{1,\mathcal{H}_a}(x,y), \Phi_{1,\mathcal{H}_d}(x,y)\Big) &= x \\ \Phi_{1,\mathcal{R}_d}\Big(\Phi_{1,\mathcal{H}_a}(x,y), \Phi_{1,\mathcal{H}_d}(x,y)\Big) &= y \end{aligned}$$

C'est à dire, par définition des fonctions filtres, qu'il s'agit de montrer que :

$$\begin{aligned} \tfrac{x+y}{2} + \tfrac{x-y}{2} &= x \\ \tfrac{x+y}{2} - \tfrac{x-y}{2} &= y \end{aligned}$$

Ce qui est clairement le cas.

Avec l'image précédente (figure 4) issue de TW_1, en lui appliquant la fonction R_1, on obtient :

Figure 6: Recomposition de l'image figure 4

Il s'agit bien là de l'image originale !

III. Un algorithme efficace ?

1) Étude de la complexité

Propriété 13 (Complexité de la fonction TW)

Soit $j_V \in \mathbb{N}$. Pour une image de taille nxm pixels, la complexité de TW_{j_V} est :

$$\boxed{O\Big(nm + nm^2 + mn^2\Big)}$$

Demonstration 3

Soit $j_V \in \mathbb{N}$. Soit $M \in M_{n,m}(\mathbb{R})$.

La complexité de la normalisation est de l'ordre de $O(nm)$.

Soit $k \in [0; j_v - 1]$. Soit $i \in [1; n/2^k]$

Comme la complexité de l'affectation est linéaire en la taille du vecteur, l'affectation a une complexité de $\frac{m}{2^{k+1}} - 1 + 1 = \frac{m}{2^{k+1}}$.

De plus, la fonction $\downarrow_2 o\ \Phi$ s'effectue en un temps quadratique en la taille du vecteur (due à la fonction filtre).

Ainsi, le calcul du résultat de cette fonction appliqué au vecteur $M_{i,[1;\frac{m}{2^k}]}$ a une complexité de $C(\frac{m}{2^k})^2$ avec $C > 0$.

Or

$$\begin{aligned} 2\sum_{i=1}^{n/2^k}\Big(\frac{m}{2^{k+1}} + C\frac{m^2}{2^{2k}}\Big) &= 2\Big(\frac{m}{2^{k+1}} + C\frac{m^2}{2^{2k}}\Big)\Big(\frac{n}{2^k} - 1 + 1\Big) && (1)\\ &= C\frac{nm^2}{2^{3k-1}} + \frac{nm}{2^{2k}} && (2)\end{aligned}$$

Ainsi, la première boucle s'effectue en $O\Big(\frac{nm^2}{2^{3k-1}}\Big)$.

On démontre de même que la deuxième boucle s'effectue en $O\Big(\frac{mn^2}{2^{3k-1}}\Big)$.

Bilan : (les nombres C_1, C_2 et C_3 sont des réels strictements positifs et sont associés aux O)

$$\begin{aligned} C_1nm + \sum_{k=0}^{j_V-1}\Big(C_2\frac{nm^2}{2^{3k-1}} + C_3\frac{mn^2}{2^{3k-1}}\Big) &= C_1nm + 2(C_2nm^2 + C_3mn^2)\sum_{k=0}^{j_V-1}\Big(\frac{1}{2^3}\Big)^k && (3)\\ &= C_1nm + 2(C_2nm^2 + C_3mn^2)\frac{1-(\frac{1}{8})^{j_V}}{1-\frac{1}{8}} && (4)\\ &= C_1nm + 2(C_2nm^2 + C_3mn^2)\frac{8-(\frac{1}{8})^{j_V-1}}{7} && (5)\\ &= C_1nm + \frac{2}{7}\Big(C_2nm^2 + C_3mn^2\Big)\left(8 - \left(\frac{1}{8}\right)^{j_V-1}\right) && (6)\\ &\leqslant C_1nm + \frac{16}{7}\Big(C_2nm^2 + C_3mn^2\Big) && (7)\end{aligned}$$

Ainsi la complexité de cet algorithme est bien de $\boxed{O\Big(nm + nm^2 + mn^2\Big)}$

Remarque 4

En utilisant les résultats des propriétés 8 et 9 on peut modifier la fonction $filter$ pour faire chuter cette complexité à $O(nm)$. Cependant cette modification serait valable uniquement pour les ondelettes de Haar.

Voici quelques performances en temps de l'algorithme TW_{j_V} pour des matrices carrées (l'image d'origine est la même, seule diffère la taille). Les calculs se sont fait avec la configuration suivante :

- CPU : Intel Code 2 CPU 2.40 GHz
- RAM : 3.6 Go

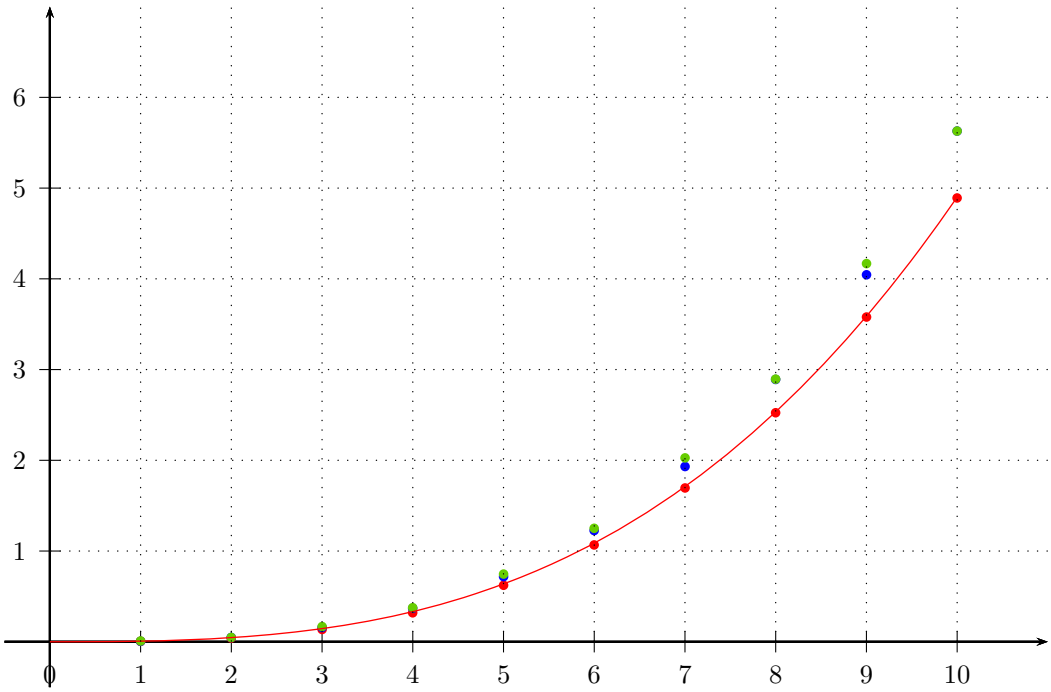

Figure 7: Complexité de la fonction TW_{j_V} (en x$\frac{1}{100}s$) en fonction de la taille de la matrice (x$\frac{1}{100}$) (en rouge $j_V = 1$, en bleu $j_V = 2$ et en vert $j_V = 3$). La courbe rouge a pour équation $y = 0.47x^3 + 0.2x^2$.

2) Compression par ondelettes

Notation 1 (Nombre d'octets d'une image)
On notera dans la suite No la fonction qui renvoie le nombre d'octets d'une image.

Remarque 5
Dans toute la suite, on confondra image et matrice.

On remarque que la transformée par ondelettes ne diminue pas la taille en octets de l'image originale. Il faut donc trouver une solution permettant de diminuer la taille tout en gardant le maximum d'information contenue dans l'image ayant subie la fonction TW_{j_V} avec $j_V \in \mathbb{N}$.

L'idée est alors d'obtenir deux fichiers, un contenant l'image approximée sur V_{-j_V} et l'autre contenant les coefficients des détails non nuls (on stockera valeur et place sous cette forme : *ligne colonne valeur* de type $(int * int * int)$). Comme la majorité des coefficients contenus dans les espaces de détails W_{-j} pour $j \in [1; j_V]$ sont nuls, la taille du deuxième fichier et nécessairement plus basse.

Définition 14 (Fonction de compression)
Ainsi on définit la fonction de compression pour $j_V \in \mathbb{N}$ comme mentionné ci-dessus :

$$\gamma_{j_V} : M_{n,m}(\mathbb{R}) \longrightarrow M_{n/2^{j_V+1},m/2^{j_V+1}}(\mathbb{R})$$

Où les entiers n et m sont supposés être des puissances de 2.

Définition 15 (Taux de compression)

Soit $j_V \in \mathbb{N}$. On définit alors la fonction taux de compression notée τ_C tel que :

$$\begin{array}{c} \tau_C : M_{n,m}(\mathbb{R}) \longrightarrow [0;1] \\ e \longmapsto \frac{No(\gamma_{j_V}(e))}{No(e)} \end{array}$$

Ainsi, on peut tracer le graphique suivant représentant le taux de compression τ_C où it représente l'entier j_V. Les images utilisées sont carrées.

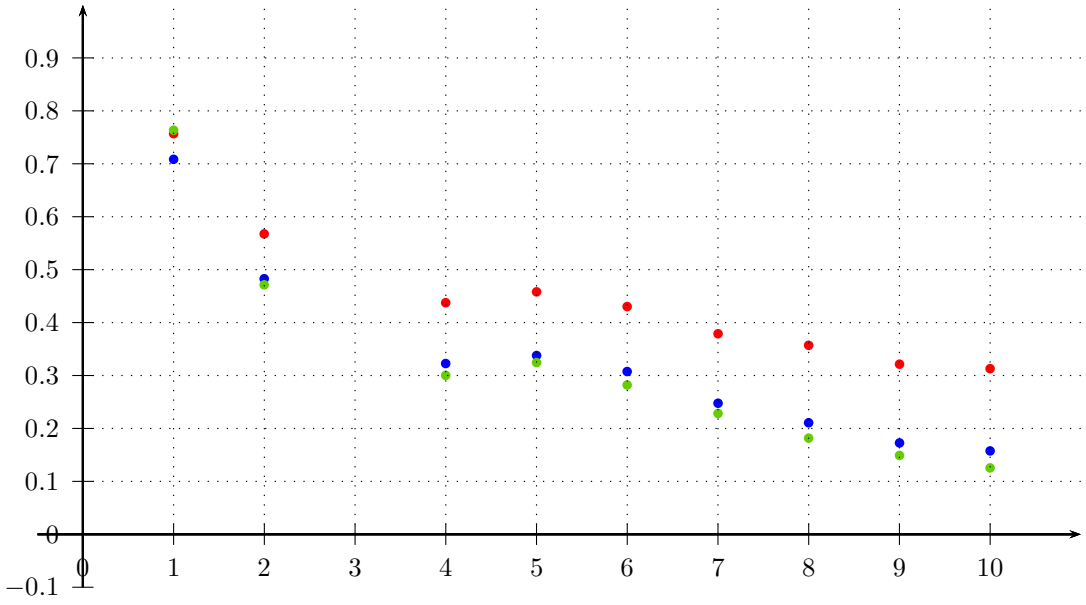

Figure 8: Taux de compression τ_C en fonction de la taille de la matrice (x$\frac{1}{100}$) (en rouge $j_V = 1$, en bleu $j_V = 2$ et en vert $j_V = 3$)

Remarque 6

Ainsi, d'après ce graphique, même si le temps recquis est long, la compression semble plus efficace pour les grandes images que pour les petites. De plus, plus j_V augmente, plus la compression est efficace (en effet, on divise à chaque fois la taille de l'image par 2).

3) Estimation de l'erreur

Comme on l'a vu dans la partie précédente, la fonction γ introduit une erreur qui provient du fait du passage d'une matrice de type $float$ au type int. De plus, l'appel à la fonction $Pixel\_matrix.intmatrix\_of\_matrix$ introduit une autre erreur qui est le passage d'un intervalle de $\mathbb{R}$ à l'ensemble $[0;255]$. Cependant, on le verra plus tard, le changement de signe n'influe pas sur le calcul de l'erreur.

On va donc introduire une fonction erreur. Pour cela, l'idée la plus simple est d'utiliser une fonction distance. Cette fonction distance devra respecter deux critères :

- prendre en compte la taille de la matrice
- prendre en compte tous les écarts relatifs entre les coefficients

Propriété 14 (Distance de compression)

On définit par conséquent la distance suivante :

$$\begin{array}{c} d : M_{n,m}(\mathbb{R}) \,\mathrm{x}\, M_{n,m}(\mathbb{R}) \longrightarrow \mathbb{R}^+ \\ (M,N) \longmapsto \frac{\|M-N\|}{nm} \end{array}$$

Définition 16 (Fonction erreur)

Soit $j_V \in \mathbb{N}$. On définit la fonction erreur comme suit :

$$\begin{aligned} \mathcal{E}_{j_V} : M_{n,m}(\mathbb{R}) &\longrightarrow \mathbb{R} \\ M &\longmapsto d\left(M, \left((\gamma_{j_V})^{-1} \, o \, \gamma_{j_V}\right)(M)\right) \end{aligned}$$

En reprenant l'image originale, on obtient alors :

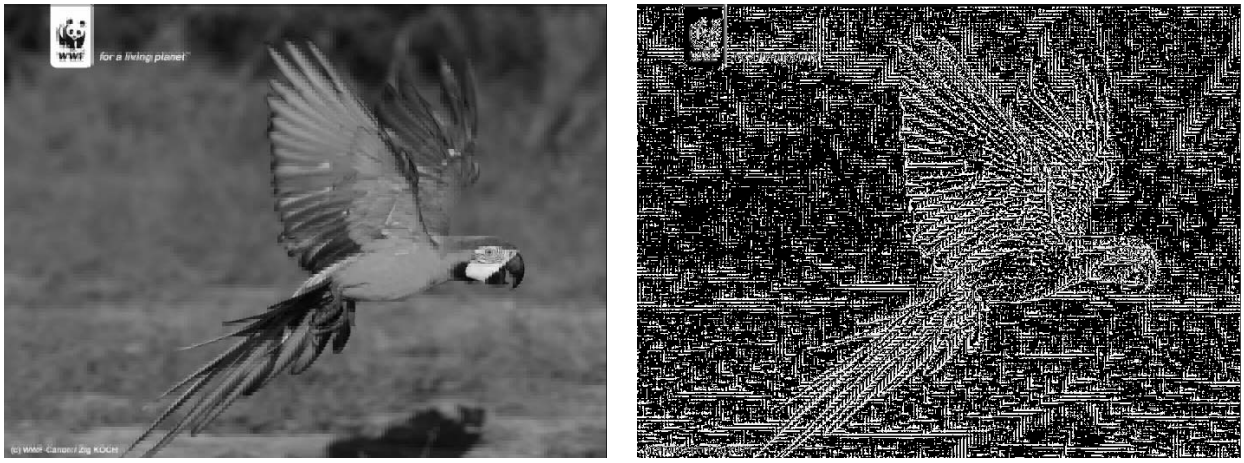

Figure 9: Image décompressée issue de $\left((\gamma_{j_V})^{-1} \, o \, \gamma_{j_V}\right)$ (à gauche) et image représentant l'erreur commise (à droite)

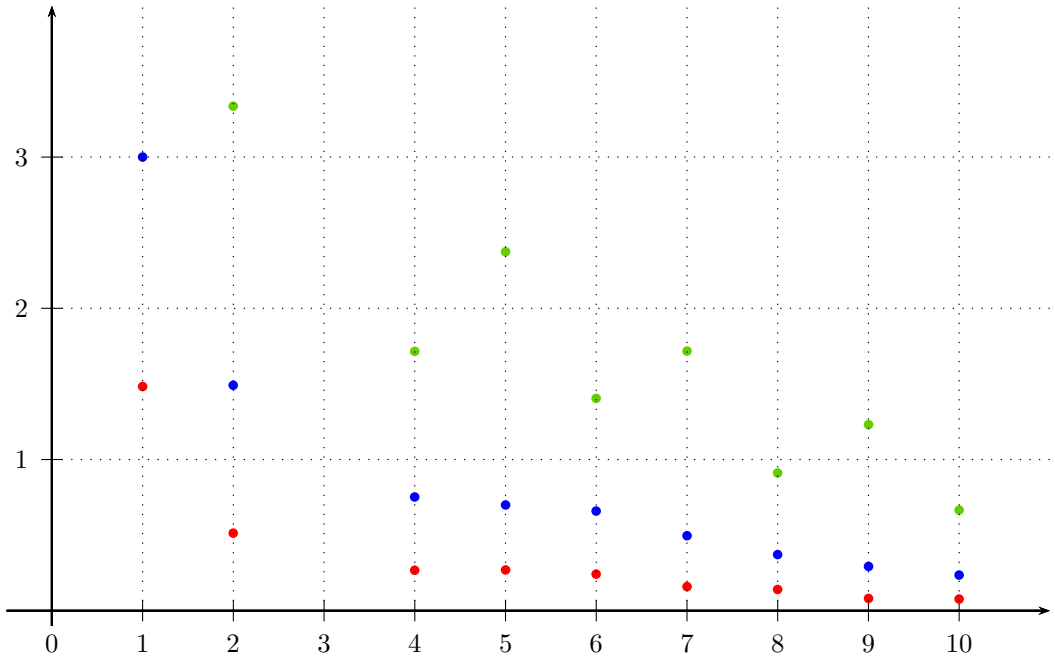

Figure 10: Erreur $\mathcal{E}_{j_V}$ liée à la compression ($\text{x}\frac{1}{100}$) en fonction de la taille de la matrice ($\text{x}\frac{1}{100}$) (en rouge $j_V = 1$, en bleu $j_V = 2$ et en vert $j_V = 3$)

Conclusion

Ainsi, l'algorithme de Mallat permet de transformer une image par ondelettes. Lorsque cette transformation est faite, comme le poids de l'image ne change pas, il a été nécessaire de trouver une solution afin de pouvoir effectivement compresser une image. Cette solution produit deux fichiers, ainsi nécessaire lors de la transmission de ces données. Peut être serait-il possible d'utiliser la stéganographie (autre domaine d'application des ondelettes) afin de récupérer plus qu'un seul fichier. De plus, l'image produite par la compression à diminuée de taille, on pourrait donc utiliser une fonction permettant de doubler sa taille en copiant les pixels.

On a également vu que l'algorithme de Mallat n'introduit pas d'erreur, c'est uniquement dû à la conversion d'une matrice de type $float$ au type int que l'erreur est introduite. De plus, cette erreur diminue lorsque la taille de l'image augmente. Tout comme le taux de compression. Cependant, la complexité temporelle est un sérieux frein, même si une complexité de l'ordre de $O(n^3)$ est acceptable pour un algorithme travaillant sur les matrices. En effet, la transformation requiert la plupart des coefficients de la matrice et la fonction $filter$ s'effectue en temps quadratique (que l'on pourrait diminuer en un temps linéaire dans le cas de l'ondelette de Haar). En titre de comparaison, la complexité du produit matriciel naïf (c'est à dire, l'algorithme issue de la définition) a une complexité de $O(n^3)$.

Enfin, lorsque l'on augmente le nombre d'itérations de l'algorithme de Mallat (i.e. le nombre j_V), le taux de compression augmente mais l'erreur également. Il est donc peut être profitable de rester dans des petites valeurs de j_V afin de garder l'intégrité et la cohésion des images.

Par conséquent, l'algorithme proposé ici est plus efficace pour les grandes images plutôt que les petites, même si l'algorithme s'effectue dans un temps plus long. Il faut donc trouver un compromis pour chaque taille d'image. Cependant, les problématiques actuelles de compression concernent surtout les grandes images et non les petites. Finalement, on peut aussi remarquer que cet algorithme est utilisé pour le format JPEG2000.

Bibliography

[1] Pascal Szacherski, *Compression, ondelettes et algorithmes afférents*, Janvier 2008

[2] Phillip K. Poon, *Wavelets*, College of Optical Sciences, University of Arizona, 2012

[3] Philippe Carré et Rémi Cornillet, *Code Matlab*, respectivements professeur de l'université de Poitiers et responsable de l'équipe Icones, étudiant à l'École Normale Supérieure de Rennes

[4] René Alt, *La transformation en ondelettes*, Professeur à l'université Pierre et Marie Curie

[5] Olivier Rioul, *Ondelettes régulières: application á la compression d'images fixes*, Télécom ParisTech, 1993

[6] Marc Lorenzi, *Code pour les images bmp*, Professeur au lycée Camille Guérin

Appendix A

Figure A.1: Image originale

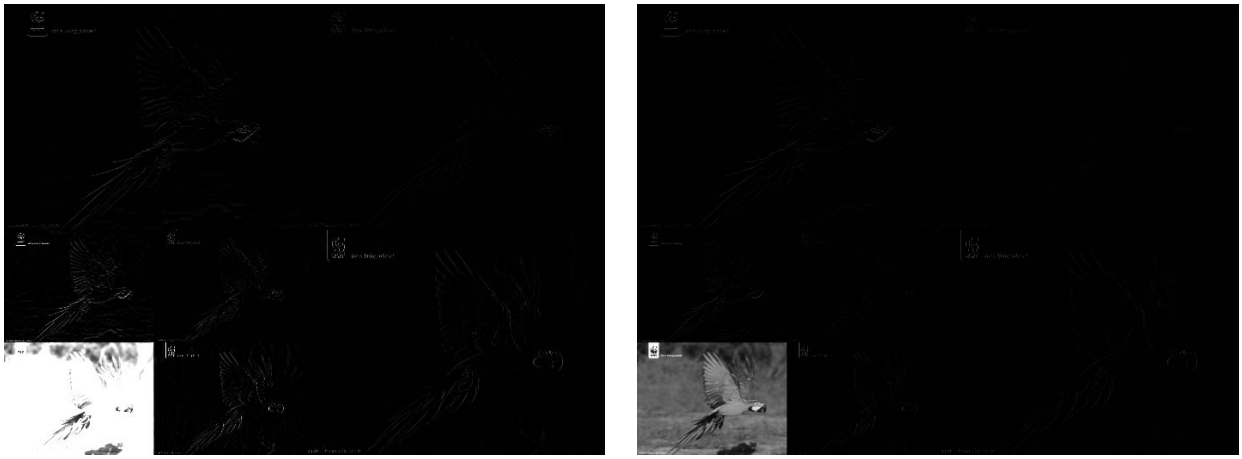

Figure A.2: Exemple de la différence entre le choix du scalaire $\frac{1}{\sqrt{2}}$ (à gauche) et du scalaire $\frac{1}{2}$ (à droite)

On voit ainsi très clairement que le scalaire $\frac{1}{2}$ permet de garder une bonne luminosité de l'image originale.

Appendix B

Fichier *array.ml*

```
let range n =
  let t = Array.make n 0 in
  for i = 1 to n - 1 do
    t.(i) <- i
  done;
  t

let ceil i d =
  if i mod d = 0 then i / d
  else i / d + 1

module type VECT =
  sig
    type 'a vect

    val vect_of_array : 'a array -> 'a vect
    val array_of_vect : 'a vect -> 'a array
    val length : 'a vect -> int
    val create : 'a -> int -> 'a vect
    val create_empty : 'a -> 'a vect
    val make_matrix : 'a -> int -> int -> 'a vect vect
    val alternate_id : int -> 'a -> ('a -> 'a) -> 'a vect
    val extend : 'a vect -> int -> 'a -> 'a vect

    val affect : 'a vect -> 'a -> int -> unit
    val value : 'a vect -> int -> 'a
    val reverse : 'a vect -> 'a vect
    val concat : 'a vect -> 'a vect -> 'a vect
    val sum : 'a vect -> 'a vect -> ('a -> 'a -> 'a) -> 'a vect
    val prod : 'a vect -> 'a vect -> ('a -> 'a -> 'a) -> 'a vect

    val sub_vect : 'a vect -> int -> int -> 'a vect
    val put_in : 'a vect -> 'a vect -> int -> unit
    val dilate : 'a vect -> int -> 'a -> 'a vect
    val extr_vect : 'a vect -> int -> int -> int -> 'a vect

    val filter : 'a vect -> 'a vect -> 'a vect -> ('a -> 'a -> 'a) -> ('a -> 'a -> 'a) -
  end;;
```

```
module Vect : VECT =
  struct
    type 'a vect = 'a array

    let vect_of_array t = t
    let array_of_vect v = v

    let length v = Array.length v

    let create x n = Array.make n x

    let create_empty x = Array.make 0 x

    let value v ind = v.(ind)

    let affect v x ind = v.(ind) <- x

    let make_matrix x n m = Array.make_matrix n m x

    let alternate_id n one opp =
      let v = create one n in
      for i = 0 to n - 1 do
        if i mod 2 = 1 then
          affect v (opp one) i
      done; v

    let reverse v =
      let n = length v in
      let v_rev = create (value v 0) n in
      for i = 0 to n - 1 do
        affect v_rev (value v (n - 1 - i)) i
      done; v_rev

    let concat v1 v2 =
      let n = length v1 and p = length v2 in
      if n = 0 then v2
      else if p = 0 then v1
      else
        let v = create (value v1 0) (n + p) in
        for i = 1 to (n + p - 1) do
          if i < n then
            affect v (value v1 i) i
          else
            affect v (value v2 (i - n)) i
        done; v

    let sum v1 v2 sum_elem =
      let n = length v1 in
      if not (n = length v2) then
        failwith "Vect.sum"
      else
        let v = create (value v1 0) n in
        for i = 0 to n - 1 do
          affect v (sum_elem (value v1 i) (value v2 i)) i
        done; v
```

```
let prod v1 v2 prod_elem =
  let n = length v1 in
  if not (n = length v2) then
    failwith "Vect.prod"
  else
    let v = create (value v1 0) n in
    for i = 0 to n - 1 do
      affect v (prod_elem (value v1 i) (value v2 i)) i
    done; v

let sub_vect v i j =
  let v_sub = create (value v i) (j - i) in
  for k = 1 to j - i - 1 do
    affect v_sub (value v (k + i)) k
  done; v_sub

let put_in v1 v2 i =
  let n = length v2 in
  if (i + n) > (length v1) then
    failwith "Vect.put_in"
  else
    for j = 0 to n - 1 do
      affect v1 (value v2 j) (j + i)
    done

let extend x m zero =
  let n = length x in
  if m = 0 then x
  else
    let y = create zero (n + m) in
    put_in y x 0;
    y

let dilate v s zero =
  let n = length v in
  let v_dil = create zero (n*s) in
  for i = 0 to n - 1 do
    affect v_dil (value v i) (s*i)
  done;
  v_dil

let extr_vect v ind_beg diff_ind ind_end =
  let n = ceil (ind_end - ind_beg) diff_ind in
  let v_ex = create (value v 0) n in
  let p = ref ind_beg and q = ref 0 in
  while !p < ind_end do
    affect v_ex (value v !p) !q;
    incr q; p := !p + diff_ind
  done; v_ex
```

```
    let filter b a x sum prod inv zero =
      let n = length x in
      let y = create (value x 0) n in
      let nb = length b in

      let b = (if n > nb then extend b (n - nb) zero else b) in

      for i = 0 to n - 1 do
        let s = ref zero in
        for j = 0 to i do
          s := sum !s (prod (value b j) (value x (i - j)))
        done;
        s := prod !s (inv (value a 0));
        affect y !s i
      done;
      y

  end

module type MATRIX =
  sig
    type 'a matrix

    val dim : 'a matrix -> (int * int)
    val create : 'a -> int -> int -> 'a matrix
    val matrix_of_vect : 'a Vect.vect -> 'a matrix

    val value : 'a matrix -> (int * int) -> 'a
    val affect : 'a matrix -> 'a -> (int * int) -> unit
    val vect : 'a matrix -> int -> 'a Vect.vect
    val affect_vect : 'a matrix -> 'a Vect.vect -> (int * int) -> unit
    val put_in : 'a matrix -> 'a matrix -> (int * int) -> unit
    val line : 'a matrix -> int -> 'a Vect.vect
    val affect_line : 'a matrix -> 'a Vect.vect -> (int * int) -> unit

    val sum : 'a matrix -> 'a matrix -> ('a -> 'a -> 'a) -> 'a matrix
    val prod_scal_cano : 'a matrix -> 'a matrix -> ('a -> 'a -> 'a) -> ('a -> 'a -> 'a)
    val norm_eucli : 'a matrix -> ('a -> 'a -> 'a) -> ('a -> 'a -> 'a) -> 'a -> 'a

    val extr_matrix : 'a matrix -> (int * int) -> (int * int) -> 'a matrix
  end;;

module Matrix : MATRIX =
  struct
    type 'a matrix = ('a Vect.vect) Vect.vect

    let value mat (i, j) = Vect.value (Vect.value mat i) j

    let affect mat x (i, j) =
      let vect = Vect.value mat i in
      Vect.affect vect x j; Vect.affect mat vect i

    let dim mat = (Vect.length mat, Vect.length (Vect.value mat 0))

    let create x n m = Vect.make_matrix x n m
```

```
let matrix_of_vect v = Vect.create v 1

let vect mat j =
  let (n, m) = dim mat in
  let vect_mat = Vect.create (value mat (0, j)) n in
  for i = 1 to n - 1 do
    Vect.affect vect_mat (value mat (i, j)) i
  done; vect_mat

let affect_vect mat vect (i, j) =
  for k = 0 to (Vect.length vect - 1) do
    affect mat (Vect.value vect k) (i + k, j)
  done

let line mat i = Vect.value mat i

let affect_line mat vect (i, ind_beg) =
  let line_mat = line mat i in
  Vect.put_in line_mat vect ind_beg;
  Vect.affect mat line_mat i

let put_in mat1 mat2 (i, j) =
  for k = 0 to fst (dim mat2) - 1 do
    affect_line mat1 (line mat2 k) (i + k, j)
  done

let sum mat1 mat2 sum_elem =
  let size = dim mat1 in
  if not (size = dim mat2) then
    failwith "Matrix.sum"
  else
    let mat = create (value mat1 (0, 0)) (fst size) (snd size) in
    for i = 0 to fst size - 1 do
      affect_line mat (Vect.sum (line mat1 i) (line mat2 i) sum_elem) (i, 0)
    done;
    mat

let prod_scal_cano mat1 mat2 sum_elem prod_elem zero =
  let (n, m) = dim mat1 in
  if (n, m) <> dim mat2 then
    failwith "Matrix.prod_scal_cano"
  else
    let prod_scal = ref zero in
    for i = 0 to n - 1 do
      for j = 0 to m - 1 do
        prod_scal := sum_elem !prod_scal (prod_elem (value mat1 (i, j)) (value mat2
      done
    done;
    !prod_scal

let norm_eucli mat sum_elem prod_elem zero =
  prod_scal_cano mat mat sum_elem prod_elem zero
```

```
    let extr_matrix mat (i, j) (n2, m2) =
      let (n, m) = dim mat in
      if (i + n2 > n || j + m2 > m) then
        failwith "Matrix.extr_matrix";
      let mat_extr = create (value mat (i, j)) n2 m2 in
      for k = 0 to n2 - 1 do
        for l = 0 to m2 - 1 do
          affect mat_extr (value mat (i + k, j + l)) (k, l)
        done
      done;
      mat_extr

  end;;

#use "bmp.ml"

module type PIXEL_MATRIX =
  sig
    type pixel
    type pixel_matrix

    val barycenter : pixel -> float

    val read_pixels : bitmapFileHeader -> bitmapInfoHeader -> in_channel -> pixel_matrix
    val write_pixels : out_channel -> pixel_matrix -> unit
    val read_bmp : string -> pixel_matrix
    val write_bmp : string -> pixel_matrix -> unit

    val intmatrix_of_matrix : float Matrix.matrix -> int Matrix.matrix

    val gray_levels : pixel_matrix -> float Matrix.matrix
    val pick_color : pixel -> int -> int
    val extr_uplet : pixel_matrix -> int -> float Matrix.matrix
    val red_filter : pixel_matrix -> float Matrix.matrix
    val blue_filter : pixel_matrix -> float Matrix.matrix
    val green_filter : pixel_matrix -> float Matrix.matrix
    val combine_filters : int Matrix.matrix -> int Matrix.matrix -> int Matrix.matrix ->
  end;;

module Pixel_matrix : PIXEL_MATRIX =
  struct
    type pixel = (int * int * int)
    type pixel_matrix = pixel Matrix.matrix

    let barycenter (r, g, b) =
      ( (float_of_int r) +. (float_of_int g) +. (float_of_int b)) /. 3.
```

```
let read_pixels fh ih channel =
  let w = ih.biWidth
  and h = ih.biHeight in
  let offs = offset w in
  let m = Matrix.create (0, 0, 0) w h in
  for j = 0 to h - 1 do
    for i = 0 to w - 1 do
      let b = input_byte channel in
      let g = input_byte channel in
      let r = input_byte channel in
      Matrix.affect m (r, g, b) (i, j)
    done;
    for i = 1 to offs do
      let _ = input_byte channel in ()
    done
  done;
  m

let write_pixels channel m =
  let (w, h) = Matrix.dim m in
  let offs = offset w in
  for j = 0 to h - 1 do
    for i = 0 to w - 1 do
      let r, g, b = Matrix.value m (i, j) in
      output_byte channel b;
      output_byte channel g;
      output_byte channel r;
    done;
    for i = 1 to offs do
      output_byte channel 0
    done
  done

let read_bmp filename =
  let channel = open_in_bin filename in
  let fh = read_file_header channel in
  let ih = read_info_header channel in
  let m = read_pixels fh ih channel in
  close_in channel;
  m

let write_bmp filename m =
  let channel = open_out_bin filename in
  let (w, h) = Matrix.dim m in
  let fh = make_file_header w h
  and ih = make_info_header w h in
  write_file_header channel fh;
  write_info_header channel ih;
  write_pixels channel m;
  close_out channel
```

```
let intmatrix_of_matrix matrix =
  let (w, h) = Matrix.dim matrix in
  let intmatrix = Matrix.create 0 w h in
  for i = 0 to w - 1 do
    for j = 0 to h - 1 do
      let matrix_i_j = Matrix.value matrix (i, j) in
      if matrix_i_j > 255. then
        Matrix.affect intmatrix 255 (i, j)
      else if matrix_i_j < 0. then
        Matrix.affect intmatrix 0 (i, j)
      else
        Matrix.affect intmatrix (int_of_float matrix_i_j) (i, j)
    done
  done;
  intmatrix

let gray_levels m =
  let (w, h) = Matrix.dim m in
  let m1 = Matrix.create (0.) w h in
  for i = 0 to w - 1 do
    for j = 0 to h - 1 do
      let p = Matrix.value m (i, j) in
      let c = barycenter p in
      Matrix.affect m1 c (i, j)
    done
  done;
  m1

let pick_color (r, g, b) i =
  match i with
    | 1 -> r
    | 2 -> g
    | 3 -> b
    | _ -> failwith "Pixel_matrix.pick_uplet"

let extr_uplet mat c =
  let (w, h) = Matrix.dim mat in
  let m1 = Matrix.create (0.) w h in
  for i = 0 to w - 1 do
    for j = 0 to h - 1 do
      let p = Matrix.value mat (i, j) in
      let u = float_of_int (pick_color p c) in
      Matrix.affect m1 u (i, j)
    done
  done;
  m1

let red_filter mat = extr_uplet mat 1
let green_filter mat = extr_uplet mat 2
let blue_filter mat = extr_uplet mat 3
```

```
    let combine_filters r_f g_f b_f =
      let (w, h) = Matrix.dim r_f in
      if (not ((w,h) = Matrix.dim g_f)) || (not ((w,h) = Matrix.dim b_f)) then
        failwith "Pixel_matrix.combine_filters"
      else
        let m = Matrix.create (0, 0, 0) w h in
        for i = 0 to w - 1 do
          for j = 0 to h - 1 do
            let r = Matrix.value r_f (i, j)
            and g = Matrix.value g_f (i, j)
            and b = Matrix.value b_f (i, j)
            in
            Matrix.affect m (r, g, b) (i, j)
          done
        done;
        m

  end
```

Fichier *fwt.ml*

```
#use "array.ml"

let op x = -. x
let inv x = 1. /. x
let prod = ( *. )
let sum = ( +. )
let zero = 0.
let one = 1.
let two = sum one one
let vect_1 = Vect.vect_of_array [|one|]

let round_2 n =
  if n mod 2 = 0 then n/2
  else n/2 + 1

let rec pow x n =
  if n = 0 then 1
  else
    let y = pow x (n / 2) in
    if n mod 2 = 0 then
      y * y
    else
      x * y * y

let normalise x (n, m) pow_2 =
  let offs_n = (if (n mod pow_2 = 0) then 0 else pow_2 - (n mod pow_2))
  and offs_m = (if (m mod pow_2 = 0) then 0 else pow_2 - (m mod pow_2)) in
  let y = Matrix.create zero (n + offs_n) (m + offs_m) in
  Matrix.put_in y x (0, 0);
  y, (n + offs_n , m + offs_m)
```

```
let denormalise x (n, m) =
  Matrix.extr_matrix x (0, 0) (n, m)

let aco f x =
  let n = Vect.length x
  and p = Vect.length f in
  let xpadded = ref (Vect.create_empty (Vect.value x 0)) in
  (if p < n then
    xpadded := (Vect.concat x (Vect.sub_vect x 0 p))
  else
    let z = Vect.create zero p in
    for i = 0 to p - 1 do
      let imod = (i mod n) in
      Vect.affect z (Vect.value x imod) i
    done;
    xpadded := (Vect.concat x z));
  let fflip = Vect.reverse f in
  let ypadded = Vect.filter fflip vect_1 !xpadded sum prod inv zero in
  Vect.sub_vect ypadded (p - 1) (n + p - 1)

let hi_up x g0 =
  let tmp = ref (Vect.dilate x 2 zero) in
  let len_tmp = Vect.length !tmp - 1 in
  tmp := Vect.concat (Vect.create (Vect.value !tmp len_tmp) 1)
        (Vect.sub_vect !tmp 0 len_tmp);
  aco g0 !tmp

let lo_conv x qmf =
  let d = aco qmf x in
  Vect.extr_vect d 0 2 (Vect.length d - 1)

let ico f x =
  let n = Vect.length x
  and p = Vect.length f in
  let xpadded = ref (Vect.create (Vect.value x 0) 1) in
  (if p <= n then
    xpadded := (Vect.concat (Vect.sub_vect x (n - p) n) x)
  else
    let z = Vect.create zero p in
    for i = 0 to p - 1 do
      let imod = ( (p * n - p + i) mod n ) in
      Vect.affect z (Vect.value x imod) i
    done;
    xpadded := (Vect.concat z x));
  let ypadded = Vect.filter f vect_1 !xpadded sum prod inv zero in
  Vect.sub_vect ypadded p (n + p)

let hi_conv s qmf =
  let s2 = Vect.concat (Vect.sub_vect s 1 (Vect.length s))
          (Vect.create (Vect.value s 0) 1) in
  let d = ico qmf s2 in
  Vect.extr_vect d 0 2 (Vect.length d - 1)
```

```
let fwt sens x l h0 =
  let len_h0 = Vect.length h0 in
  let g0 = Vect.prod (Vect.alternate_id len_h0 one op) h0 prod in

  if sens = 0 then
  (let wc, (n, m) = normalise x (Matrix.dim x) (pow 2 l) in
  let nc = ref n
  and mc = ref m in

  for jsqual = 1 to l do
    for ix = 0 to !nc - 1 do
      let row = Vect.sub_vect (Matrix.line wc ix) 0 !mc in
      Matrix.affect_line wc (lo_conv row h0) (ix, 0);
      Matrix.affect_line wc (hi_conv row g0) (ix, (!mc / 2));
    done;

    for iy = 0 to !mc - 1 do
      let row = Vect.sub_vect (Matrix.vect wc iy) 0 !nc in
      Matrix.affect_vect wc (hi_conv row g0) (!nc / 2, iy);
      Matrix.affect_vect wc (lo_conv row h0) (0, iy);
    done;

    nc := !nc / 2; mc := !mc / 2
  done; wc)

  else
    (let (n, m) = Matrix.dim x in
    let pow_2 = pow 2 (l - 1) in
    let nc = ref (n / pow_2)
    and mc = ref (m / pow_2) in

    for jsqual = 1 to l do
      for iy = 0 to !mc - 1 do
        Matrix.affect_vect x (Vect.sum
                (ico h0 (Vect.dilate
                            (Vect.sub_vect (Matrix.vect x iy) 0 (!nc / 2))
                            2 zero))
                (hi_up (Vect.sub_vect (Matrix.vect x iy) (!nc / 2) !nc) g0)
                sum) (0, iy)
      done;

      for ix = 0 to !nc - 1 do
        Matrix.affect_line x (Vect.sum
                (ico h0 (Vect.dilate
                            (Vect.sub_vect (Matrix.line x ix) 0 (!mc / 2))
                            2 zero))
                (hi_up (Vect.sub_vect (Matrix.line x ix) (!mc / 2) !mc) g0)
                sum) (ix, 0)
      done;

      nc := !nc * 2; mc := !mc * 2;
    done; x)
```

Fichier *compression_wt.ml*

```
#use "array.ml";;
#use "fwt.ml";;

let abs x =
  if x < zero then op x
  else x

let abs2 x =
  if x < 0 then - x
  else x

let haar_filter = Vect.prod_scal (Vect.vect_of_array [|1. ; 1.|]) 2. ( /. );;
let haar_filter2 = Vect.vect_of_array [|1. ; 1.|];;

let decompo_wt_matrix img_m it =
  fwt 0 img_m it haar_filter;;

let decompo_wt img it =
  let img_matrix = Pixel_matrix.read_bmp img in
  let gray_matrix = Pixel_matrix.gray_levels img_matrix in
  let decompo = decompo_wt_matrix gray_matrix it in
  decompo;;

let rec calc_indice_div it (n, m) =
  if it = 0 then
    (n, m)
  else
    calc_indice_div (it - 1) (n / 2, m / 2)

let rec calc_indice_mult it (n, m) =
  if it = 0 then
    (n, m)
  else
    calc_indice_mult (it - 1) (n * 2, m * 2)

let txt_of_matrix mat it filename =
  let (n, m) = Matrix.dim mat in
  let (size_n, size_m) = calc_indice_div it (n, m) in
  let extr_mat = Matrix.extr_matrix mat (0, 0) (size_n, size_m) in
  let file = open_out filename in
  output_string file ((string_of_int it) ^ "\n");
  for i = 0 to n - 1 do
    for j = 0 to m - 1 do
      let mat_i_j = Matrix.value mat (i, j) in
      if (i >= size_n || j >= size_m) && mat_i_j <> 0 then
        let str = (string_of_int i) ^ " " ^ (string_of_int j) ^
                  " " ^ (string_of_int mat_i_j) ^ "\n" in
        output_string file str
    done
  done;
  close_out file;
  extr_mat
```

```
let decompo_line line =
  let tab = Array.make 3 0 in
  let n = String.length line in
  let str = ref "" in
  let j = ref 0 in
  for i = 0 to n - 1 do
    if line.[i] <> ' ' then
      (str := !str ^ (String.make 1 line.[i]);
      if i = n - 1 then
        tab.(!j) <- int_of_string !str)
    else
      (tab.(!j) <- int_of_string !str;
      str := "";
      incr j)
  done;
  (tab.(0), tab.(1), tab.(2))

let matrix_of_txt mat filename =
  let file = open_in filename in
  let mult = int_of_string (input_line file) in
  let (size_n, size_m) = Matrix.dim mat in
  let (n, m) = calc_indice_mult mult (size_n, size_m) in

  let complete_mat = Matrix.create 0. n m in
  Matrix.put_in complete_mat mat (0, 0);

  let end_of_file = ref false in

  while (not !end_of_file) do
    try
      let line = input_line file in
      let (i, j, value) = decompo_line line in
      Matrix.affect complete_mat (float_of_int value) (i, j)
    with
      | End_of_file -> end_of_file := true
  done;
  close_in file;

  (complete_mat, mult);;

let compresser_img img it filename_target =
  let mat = decompo_wt img it in
  let intmat = Pixel_matrix.intmatrix_of_matrix mat in
  let extr_mat = txt_of_matrix intmat it filename_target in
  Pixel_matrix.write_bmp (filename_target ^ ".bmp")
      (Pixel_matrix.combine_filters extr_mat extr_mat extr_mat);;

let decompresser_img img file target =
  let mat = Pixel_matrix.read_bmp img in
  let gray_mat = Pixel_matrix.gray_levels mat in
  let (mat_recomp, it) = matrix_of_txt gray_mat file in
  let decomp = fwt 1 mat_recomp it haar_filter2 in
  let intdecomp = Pixel_matrix.intmatrix_of_matrix decomp in
  let combine = Pixel_matrix.combine_filters intdecomp intdecomp intdecomp in
  Pixel_matrix.write_bmp target combine;;
```

```
let error img it i =
  let mat = Pixel_matrix.read_bmp img in
  let original = Pixel_matrix.gray_levels mat in
  let comp = decompo_wt_matrix original it in

  let intcomp = Pixel_matrix.intmatrix_of_matrix comp in
  let extr = txt_of_matrix intcomp it ("text" ^ (string_of_int (i + 1))) in

  let pixextr = Pixel_matrix.combine_filters extr extr extr in
  let extrfloat = Pixel_matrix.gray_levels pixextr in

  let (recomp, _) = matrix_of_txt extrfloat ("text" ^ (string_of_int (i + 1))) in

  let decomp = fwt 1 recomp it haar_filter2 in

  let denorm_decomp = denormalise decomp (Matrix.dim original) in

  let intoriginal = Pixel_matrix.intmatrix_of_matrix original
  and intdecomp = Pixel_matrix.intmatrix_of_matrix denorm_decomp in

  let mat_diff = Matrix.sum intoriginal intdecomp ( - ) in
  Matrix.norm_eucli mat_diff ( + ) ( * ) 0
```

Appendix C

Fichier *bmp.ml*

```
#load "graphics.cma";;
open Graphics;;

type word  = int;;
type dword = int;;

type bitmapFileHeader = {
  bfType      : string;
  bfSize      : dword;
  bfReserved1 : word;
  bfReserved2 : word;
  bfOffBits   : dword;
};;

type bitmapInfoHeader = {
  biSize          : dword;
  biWidth         : dword;
  biHeight        : dword;
  biPlanes        : word;
  biBitCount      : word;
  biCompression   : dword;
  biSizeImage     : dword;
  biXPelsPerMeter : dword;
  biYPelsPerMeter : dword;
  biClrUsed       : dword;
  biClrImportant  : dword;
};;
let read_type channel =
  let s = "__" in
  s.[0] <- input_char channel;
  s.[1] <- input_char channel;
  s;;

let write_type channel =
  output_char channel 'B';
  output_char channel 'M';;
```

```
let read_dword channel =
  let a = input_byte channel in
  let b = input_byte channel in
  let c = input_byte channel in
  let d = input_byte channel in
  (d lsl 24) lor (c lsl 16) lor (b lsl 8) lor a;;

let write_dword channel x =
  let a = x lsr 24
  and b = (x lsr 16) land 255
  and c = (x lsr 8) land 255
  and d = x land 255 in
  output_byte channel d;
  output_byte channel c;
  output_byte channel b;
  output_byte channel a;;

let read_word channel =
  let a = input_byte channel in
  let b = input_byte channel in
  (b lsl 8) lor a;;

let write_word channel x =
  let c = (x lsr 8) land 255
  and d = x land 255 in
  output_byte channel d;
  output_byte channel c;;

let read_file_header channel =
  let t = read_type channel in
  let sz = read_dword channel in
  let r1 = read_word channel in
  let r2 = read_word channel in
  let off = read_dword channel in
  {
    bfType = t;
    bfSize = sz;
    bfReserved1 = r1;
    bfReserved2 = r2;
    bfOffBits = off;
  };;

let write_file_header channel fh =
  write_type channel;
  write_dword channel fh.bfSize;
  write_word channel fh.bfReserved1;
  write_word channel fh.bfReserved2;
  write_dword channel fh.bfOffBits;;
```

```
let read_info_header channel =
  let sz = read_dword channel in
  let w = read_dword channel in
  let h = read_dword channel in
  let pl = read_word channel in
  let bc = read_word channel in
  let compr = read_dword channel in
  let szim = read_dword channel in
  let xpm = read_dword channel in
  let ypm = read_dword channel in
  let clru = read_dword channel in
  let clri = read_dword channel in
  {
    biSize = sz;
    biWidth = w;
    biHeight = h;
    biPlanes = pl;
    biBitCount = bc;
    biCompression = compr;
    biSizeImage = szim;
    biXPelsPerMeter= xpm;
    biYPelsPerMeter= ypm;
    biClrUsed = clru;
    biClrImportant = clri;
  };;

let write_info_header channel ih =
  write_dword channel ih.biSize;
  write_dword channel ih.biWidth;
  write_dword channel ih.biHeight;
  write_word channel ih.biPlanes;
  write_word channel ih.biBitCount;
  write_dword channel ih.biCompression;
  write_dword channel ih.biSizeImage;
  write_dword channel ih.biXPelsPerMeter;
  write_dword channel ih.biYPelsPerMeter;
  write_dword channel ih.biClrUsed;
  write_dword channel ih.biClrImportant;;

let offset w =
  let r = (3 * w) mod 4 in
  if r = 0 then 0
  else 4 - r;;

let make_file_header w h =
  let off = offset w in
  {
    bfType = "BM";
    bfSize = (w + off) * h * 3 + 54;
    bfReserved1  = 0;
    bfReserved2 = 0;
    bfOffBits = 54;
  };;
```

```
let make_info_header w h =
  let off = offset w in
  {
    biSize = 40;
    biWidth = w;
    biHeight  = h;
    biPlanes  = 1;
    biBitCount = 24;
    biCompression = 0;
    biSizeImage = (w + off) * h * 3;
    biXPelsPerMeter = 0;
    biYPelsPerMeter = 0;
    biClrUsed = 0;
    biClrImportant = 0;
  };;
```