

TIPE: Introduction à la théorie des ondelettes

Xavier Friederich et Gaétan Bahl

22 août 2013

Table des matières

1	Premières définitions	4
1.1	Définition 1 : Ondelette	4
1.2	Exemples d'ondelettes mères :	5
1.2.1	Ondelette de Haar :	5
1.2.2	Ondelette "chapeau mexicain" :	5
1.2.3	Ondelette de Morlet :	5
2	Transformation en ondelettes	6
2.1	Transformation en ondelette continue	6
2.2	Transformation en ondelettes discrète.	8
3	La théorie de l'analyse multirésolution, un outil pour la construction de bases d'ondelettes	8
3.1	Définition d'une analyse multirésolution.	8
3.2	Intérêt d'une analyse multirésolution	9
4	Algorithme de décomposition en ondelettes de Stéphane Mallat (1989)	9
4.1	Principe	9
4.2	Démonstration dans un cas simple : le cas des ondelettes de Haar.	9
4.2.1	Introduction et définition des notations	10
4.2.2	Propriété mathématique : Orthogonalité dans les espaces euclidiens	11
4.2.3	Utilisation de la propriété : décomposition orthogonale en somme directe.	12
4.2.4	Étape principale de l'algorithme : passage à la résolution inférieure, détermination des coefficients à la résolution inférieure.	14
4.3	Schématisation de l'algorithme de Mallat (compression d'un signal Ψ_p par des ondelettes)	16
4.4	Représentation matricielle de l'algorithme utilisant les ondelettes de Haar	17
4.4.1	Exemple	18

5	Application à la compression des données	19
6	Utilisation pratique de la transformation par ondelettes discrète	19
6.1	Le choix des outils	20
6.1.1	Le langage : Python	20
6.1.2	La librairie de traitement d'image : PIL	20
6.1.3	La librairie d'interface graphique : Tkinter	20
6.1.4	Le module time	21
6.1.5	Le module struct	21
6.1.6	Le module socket	21
6.2	Exemples d'images traitées avec notre algorithme	21
6.3	L'application graphique	22
6.4	L'algorithme fasthaar	26
6.4.1	Fonctionnement	26
6.4.2	Coût et Performance	29
6.5	Transferts et échanges d'images par le réseau	30
6.5.1	But du programme	30
6.5.2	Processus d'archivage	30
6.5.3	Processus de restauration	30
7	Description du code	32
7.1	Le fichier ondelettes.py	32
7.1.1	La classe Matrice	32
7.1.2	La classe MatriceImage	33
7.2	Le fichier ondelettesGUI.py	34
7.2.1	La classe Appli	34
7.2.2	La classe DialogScale	35
7.2.3	La fonction main	36
7.3	Le fichier bench.py	36
7.3.1	La fonction image_gen	36
7.3.2	La fonction randomize	36
7.3.3	La fonction bench_square	36
7.3.4	La fonction fasthaar	36
7.3.5	La fonction main	36
7.4	Le fichier serveur.py	36
7.4.1	La fonction main	36
7.4.2	La fonction recvImage	37
7.4.3	La fonction fasthaar_srv	37
7.4.4	La fonction sendCoef	37
7.5	Le fichier client.py	37
7.5.1	La fonction main	37
7.5.2	La fonction sendImage	38
7.5.3	La fonction askcompress	38
7.5.4	La fonction storeImage	38
7.5.5	La fonction askcoeff	38

8	Annexes	39
8.1	Fichier ondelettes.py	39
8.2	Fichier ondelettesGUI.py	45
8.3	Fichier bench.py	49
8.4	Fichier serveur.py	51
8.5	Fichier client.py	54
9	Liens	56

Table des figures

1	Ondelette de Haar	5
2	Ondelette "chapeau mexicain"	5
3	Ondelette de Morlet	6
4	Dilatation d'ondelette	7
5	Translation d'ondelette	7
6	L'image de départ	22
7	Détail de l'image de départ	22
8	Détail de l'image compressée	23
9	Détail de l'image compressée à 100%	23
10	Fenêtre principale de l'application	24
11	Menus de l'application	24
12	Le sélecteur de seuil de compression	25
13	Fonctionnement de l'algorithme	26
14	Tableau créé par l'algorithme	27
15	Tableau après l'étape 1	27
16	Tableau après l'étape 2	27
17	Tableau après compression	27
18	Tableau après synthèse sur les colonnes	28
19	Tableau après synthèse sur les lignes	28
20	Tableau après synthèse sur les lignes	28
21	Pixels avant et après traitement	28
22	Coût temporel de l'algorithme fasthaar en fonction de la taille de l'image	29
23	Processus d'archivage	31
24	Processus de restauration	31

1 Premières définitions

La transformation en ondelettes est apparue pour la première fois dans le domaine de la géophysique vers 1980 pour l'analyse des données sismiques. Elle aura été formalisée par Morlet, Grassmann et Goupillard. De manière analogue à la théorie des séries de Fourier, les ondelettes sont principalement utilisées pour la décomposition de fonctions. La décomposition d'une fonction en ondelettes consiste à l'écrire comme une somme pondérée de fonctions obtenues à partir d'opérations simples effectuées sur une fonction principale appelée ondelette-mère. Ces opérations consistent en des translations et dilatations de la variable. Selon que ces translations et dilatations sont choisies de manière continue ou discrète, on parlera d'une transformée en ondelettes discrète ou continue. Le travail suivant fera l'objet du cas particulier de la transformation en ondelettes unidimensionnelle.

1.1 Définition 1 : Ondelette

Une ondelette est d'un point de vue géométrique et schématique une forme d'onde, l'idéalisation d'une note de musique, d'une durée limitée et qui a une valeur moyenne égale à 0.

Plus formellement, pour le cas d'une ondelette-mère (celle que l'on va pouvoir dilater et translater afin d'obtenir les autres ondelettes définissant ainsi une famille d'ondelettes), il s'agit d'une fonction ψ de l'espace de Lebesgue $L^2(\mathbb{R})$ (espace des fonctions à valeurs dans \mathbb{C} de carré intégrable) et telle que :

$$\int_{\mathbb{R}} \psi(t) \cdot dt = 0$$

ce qui provient de la condition $\int_{\mathbb{R}} \frac{|\hat{\psi}(\omega)|^2}{|\omega|} d\omega < \infty$ où $\hat{\psi}$ est la transformée de

Fourier de ψ , donnée par la formule $\hat{\psi}(\omega) = \int_{-\infty}^{+\infty} \psi(t) \cdot e^{-2i\pi\omega t} dt$

Cette condition, dite condition d'admissibilité est nécessaire pour que la transformée en ondelettes d'une fonction existe !

Si l'ondelette -fonction analysante- est convenablement choisie, la transformation en ondelettes est inversible et la fonction peut être reconstruite après analyse suivant l'équation :

$$f = C_{\psi}^{-1} \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \frac{1}{a^2} \langle f, \psi_{a,b} \rangle \psi_a \cdot b da \cdot db$$

Le coefficient C_{ψ} , si donc il existe, est donné par : $C_{\psi} = 2\pi \int_{\mathbb{R}} \frac{|\hat{\psi}(\omega)|^2}{|\omega|} d\omega < \infty$

De manière plus « imagée », l'ondelette doit osciller localement autour de l'axe des abscisses. Il existe une infinité de fonctions d'ondelettes car toute fonction oscillante localisée est une ondelette-mère possible. Différentes familles d'ondelettes peuvent être utilisées en fonction du problème à résoudre. C'est un des nombreux avantages de la transformée en ondelettes par rapport à la transformée de Fourier (qui est liée exclusivement aux fonctions sinus et cosinus) que

de pouvoir choisir l'ondelette à utiliser pour l'analyse.

1.2 Exemples d'ondelettes mères :

1.2.1 Ondelette de Haar :

$$\mathcal{H} : [0, 1] \longrightarrow \{-1; 1\}$$

Il s'agit de la fonction

$$x \longmapsto \begin{cases} 1 & \text{si } x \in [0; \frac{1}{2}] \\ -1 & \text{si } x \in]\frac{1}{2}; 1] \end{cases}$$

On pourra remarquer que \mathcal{H} est discontinue en $\frac{1}{2}$.

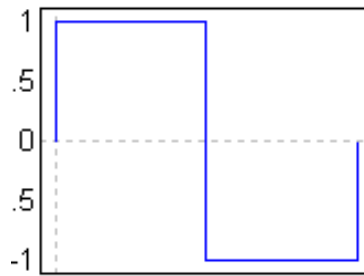


FIGURE 1 – Ondelette de Haar

1.2.2 Ondelette "chapeau mexicain" :

On peut définir cette fonction par

$$\begin{aligned} \psi : \mathbb{R} &\longrightarrow \mathbb{R} \\ t &\longmapsto \frac{2}{\sqrt{3}}\pi^{-\frac{1}{4}}(1-t^2)e^{-\frac{t^2}{2}} \end{aligned}$$

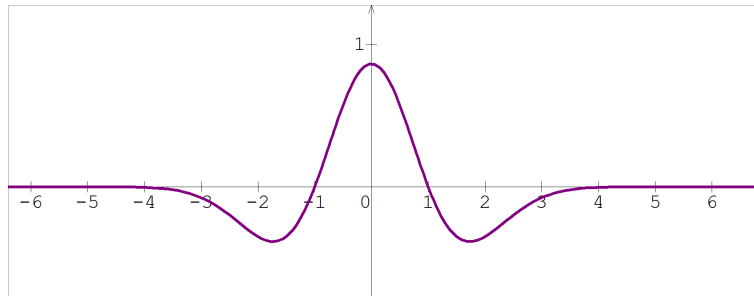


FIGURE 2 – Ondelette "chapeau mexicain"

1.2.3 Ondelette de Morlet :

On peut définir cette fonction par

$$\begin{aligned} \psi : \mathbb{R} &\longrightarrow \mathbb{R} \\ t &\longmapsto \cos(5t)e^{(-\frac{t^2}{2})} \end{aligned}$$

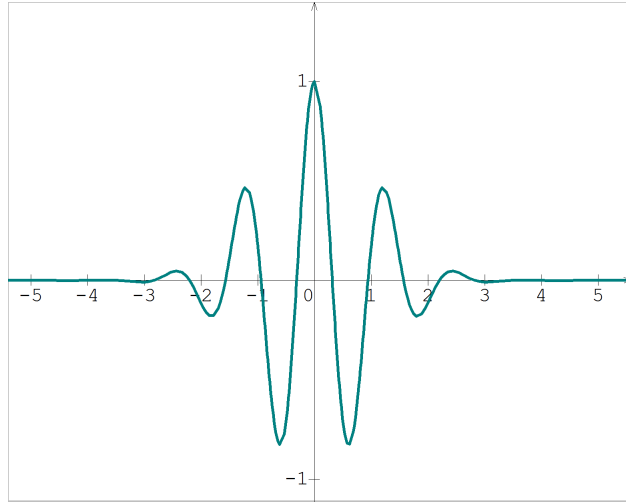


FIGURE 3 – Ondelette de Morlet

2 Transformation en ondelettes

La transformée en ondelettes est une transformée linéaire qui décompose un signal en fréquences en conservant une certaine localisation spatiale. Concrètement, le signal de départ est projeté sur un ensemble de fonctions de bases qui varient en fréquence et en espace.

2.1 Transformation en ondelette continue

La transformée en ondelette continue utilise des dilatations et des translations de la fonction ondelette mère.

Définition : produit scalaire

Soient f et g deux fonctions réelles; on définit sur le \mathbb{R} -espace vectoriel $F(\mathbb{R}, \mathbb{R})$ leur produit scalaire par l'intégrale suivante :

$$\langle f|g \rangle = \int_{\mathbb{R}} f(x)g(x)dx$$

Avec la condition d'admissibilité donnée en première page, la transformée en ondelette continue de la fonction f est définie à facteur constant près comme le produit scalaire de f et de ψ .

$$\mathcal{W}_{(a,b)}(f) = \frac{1}{\sqrt{a}} \int_{-\infty}^{+\infty} f(t) \cdot \psi\left(\frac{t-b}{a}\right) \cdot dt \text{ avec } a \in \mathbb{R}_+^*, b \in \mathbb{R}$$

Notons que a permet de donner *l'échelle* (c'est le coefficient de dilatation, de fréquence) et b détermine la position de l'ondelette sur l'axe des temps. $\frac{1}{\sqrt{a}}$ est le *facteur de normalisation* de l'énergie nécessaire pour que le signal transformé ait la même énergie à toutes les échelles.

Ex : dilatation. L'ondelette verte a été dilatée à partir de l'ondelette rouge (ondelette-mère). On a $b = 0$ et $a \neq 1$.

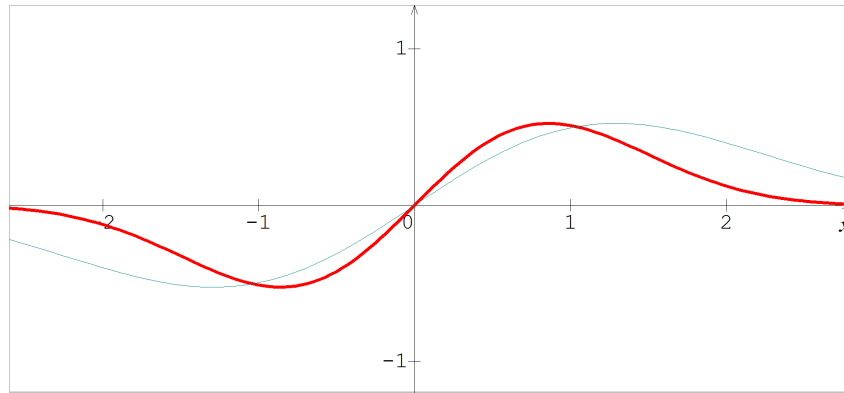


FIGURE 4 – Dilatation d'ondelette

Ex : translation. L'ondelette verte a été traduite à partir de l'ondelette rouge (ondelette-mère). On a $b \neq 0$ et $a = 1$.

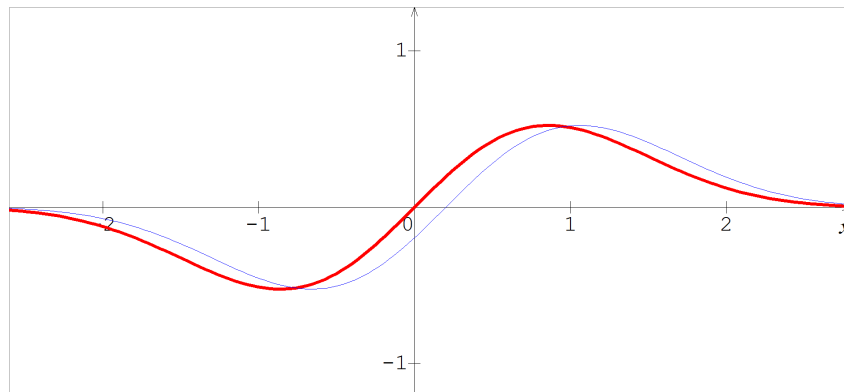


FIGURE 5 – Translation d'ondelette

2.2 Transformation en ondelettes discrète.

La transformation en ondelettes discrète qui a été introduite par Morlet se construit à partir de « bases » de fonctions du type :

$$f_{t_0, \Delta t}(t) = \frac{1}{\sqrt{\Delta t}} f\left(\frac{t - t_0}{\Delta t}\right) \text{ avec } \Delta t > 0, t_0 \in \mathbb{R}$$

Δt peut être choisi « géométriquement » ; les paramètres de translations t_0 et Δt sont proportionnels (c'est-à-dire $\exists k \in \mathbb{R}, t_0 = k \cdot \Delta t$).

Une gamme d'échelles Δt utilisée couramment est la gamme d'échelles dyadiques $\frac{1}{2^p}$

On a alors avec $t_0 = k \cdot \Delta t$:

$$f_{t_0, \Delta t}(t) = 2^{\frac{p}{2}} f(2^p \cdot x - k), \text{ c'est-à-dire on peut considérer la famille d'ondelettes } \psi_{k,p} = 2^{\frac{p}{2}} \psi(2^p x - k), (k, p) \in \mathbb{Z}^2$$

Il est intéressant de considérer des familles orthogonales d'ondelettes formant une base hilbertienne de $L^2(\mathbb{R})$ alors toute fonction f de cet espace peut s'écrire

$$f = \sum_{(k,p) \in \mathbb{Z}^2} f_{k,p} \psi_{k,p} \text{ où les } f_{k,p} = \langle f | \psi_{k,p} \rangle \text{ sont appelés coefficients d'ondelettes.}$$

La transformation en ondelettes discrète est presque naturellement associée à des algorithmes plus efficaces et plus rapides que des algorithmes du type FFT qui utilisent la transformée de Fourier.

Une famille d'ondelettes par exemple couramment utilisée dans la transformation en ondelettes discrète est la famille infinie des ondelettes orthogonales de Daubechies : c'est une des familles d'ondelettes les plus performantes.

3 La théorie de l'analyse multirésolution, un outil pour la construction de bases d'ondelettes

Pour construire des bases d'ondelettes orthonormées, les théoriciens Mallat et Meyer ont introduit la notion d'analyse multirésolution.

3.1 Définition d'une analyse multirésolution.

Une analyse multirésolution est une suite $\{V_k\}_{k \in \mathbb{Z}}$ de sous-espaces fermés de $\mathcal{L}^2(\mathbb{R})$ tels que :

- $\forall (k, l) \in \mathbb{Z}^2, f \in V_k \Leftrightarrow f(\cdot - 2^k l) \in V_k$ (propriété d'invariance par translation)
- $\forall k \in \mathbb{Z}, V_{k+1} \subset V_k$
- $\forall k \in \mathbb{Z}, f \in V_k \Leftrightarrow f(\frac{\cdot}{2}) \in V_{k+1}$
- $\lim_{j \rightarrow \infty} V_k = \bigcap_{k \in \mathbb{Z}} V_k = \emptyset$

- $\lim_{j \rightarrow -\infty} V_k = \overline{\bigcap_{k \in \mathbb{Z}} V_k} = \mathcal{L}^2(\mathbb{R})$ où la notation \bar{A} désigne l'adhérence de A .
- $\exists \phi, \{\phi(\cdot - n)\}_{n \in \mathbb{Z}}$ forme une base orthonormée de V_0 . ϕ est appelée fonction d'échelle associée à l'analyse multirésolution

3.2 Intérêt d'une analyse multirésolution

La fonction ϕ permet notamment la connaissance de la suite $\{V_k\}_{k \in \mathbb{Z}}$ et ainsi la déduction d'une base orthonormée de V_k pour tout $k \in \mathbb{Z}$. On peut alors définir une ondelette associée à l'analyse multirésolution : il s'agira de toute fonction ψ qui forme avec ses translatées entières une base orthonormée de W_0 , supplémentaire orthogonal de V_1 dans V_0 . En effet, il découle de la définition de W_k que $\mathcal{L}^2(\mathbb{R}) = \bigoplus_{k \in \mathbb{Z}} W_k$.

Par suite, la famille $\left\{ \frac{1}{\sqrt{2^m}} \psi \left(\frac{\cdot - 2^m n}{2^m} \right) \right\}$ forme une base orthonormée de $\mathcal{L}^2(\mathbb{R})$.

Les espaces W_k pour $k \in \mathbb{Z}$ sont appelés espaces des détails. Ils ne forment pas une famille d'espaces emboîtés mais les propriétés d'échelles et d'invariance par translation sont conservées. En effet, pour $k \in \mathbb{Z}$, W_{k-1} est orthogonal à V_{k-1} , d'où W_{k-1} orthogonal à W_k en vertu de l'égalité $\mathcal{L}^2(\mathbb{R}) = \bigoplus_{k \in \mathbb{Z}} W_k$.

4 Algorithme de décomposition en ondelettes de Stéphane Mallat (1989)

4.1 Principe

C'est un algorithme linéaire qui fait appel à un sous-échantillonnage. Concrètement, on procède à une décomposition par projections successives (c'est-à-dire de manière récursive) sur deux sous-espaces orthogonaux, l'un donnant l'allure générale de l'image (il s'agira de l'image en résolution moitié) et l'autre les détails. L'algorithme de Mallat a cependant le défaut de ne pas être invariant par translation.

On peut donner une démonstration mathématique de cet algorithme ; ici, pour simplifier, on va se limiter au cas particulier de décomposition d'un signal par les ondelettes de Haar.

4.2 Démonstration dans un cas simple : le cas des ondelettes de Haar.

La démonstration suivante montre comment on calcule les coefficients des ondelettes de Haar : on est bien évidemment dans le cadre d'une transformation en ondelettes discrètes.

4.2.1 Introduction et définition des notations

Considérons un signal échantillonné régulièrement sur $[0,1]$ en 2^p points notés x_k avec $x_k = \frac{k}{2^p}$.

On associe à cet échantillon une fonction f définie par $f(x) = \begin{cases} f_k & \text{si } x \in I_k = [x_k, x_{k+1}[\\ 0 & \text{sinon} \end{cases}$.

Quand l'échantillonnage varie, f varie en décrivant l'ensemble \mathbb{K}_p des fonctions constantes sur chacun des intervalles I_k et nulles sur $\mathbb{K} \setminus [0, 1]$.

$F(\mathbb{R}, \mathbb{R})$, ensemble des fonctions réelles à valeurs réelles, est un \mathbb{R} -espace vectoriel et on montre facilement que \mathbb{K}_p est un sous-espace vectoriel de $F(\mathbb{R}, \mathbb{R})$.

De plus, pour $p \in \mathbb{N}$, on a $\mathbb{K}_0 \subset \mathbb{K}_1 \subset \dots \subset \mathbb{K}_p \subset \mathbb{K}_{p+1} \subset \dots$ ce qui montre $\bigcup_{p \in \mathbb{N}} \mathbb{K}_p$ est un sous-espace vectoriel de $F(\mathbb{R}, \mathbb{R})$.

À partir de la fonction de Haar H , on définit la fonction $H_{p,k}$ par $H_{p,k}(x) = H(2^p x - k)$. $(p, k) \in \mathbb{N}^2$.

[Pour alléger l'écriture et les calculs, on peut comme ici choisir d'omettre le facteur $2^{\frac{p}{2}}$ devant $H(2^p x - k)$.]

Soit la fonction définie par

$$h_{p,k}(x) = \begin{cases} 1 & \text{si } x \in I_k = [\frac{k}{2^p}, \frac{k+1}{2^p}] = h(2^p x - k) \\ 0 & \text{sinon} \end{cases}$$

(avec h définie comme la fonction de Haar mais associant 1 quel que soit $x \in [0, 1]$).

Or toute fonction f de \mathbb{K}_p se décompose de manière unique sous la forme :

$$f = \sum_{k=0}^{2^p-1} f_k h_{p,k} = f_0 h_{p,0} + f_1 h_{p,1} + \dots + f_{2^p-1} h_{p,2^p-1}$$

$$\begin{aligned} \text{On a bien } \forall x \in [0, 1[, f(x) &= f_0 \times \begin{cases} 1 & \text{si } x \in I_0 \\ 0 & \text{sinon} \end{cases} + f_1 \times \begin{cases} 1 & \text{si } x \in I_1 \\ 0 & \text{sinon} \end{cases} + \\ \dots + f_{2^p-1} \times \begin{cases} 1 & \text{si } x \in I_{2^p-1} \\ 0 & \text{sinon} \end{cases} \end{aligned}$$

D'où $(h_{p,0}, h_{p,1}, \dots, h_{p,2^p-1})$ est une base de \mathbb{K}_p .

Avec $F(\mathbb{R}, \mathbb{R})$ muni du produit scalaire défini en *définition 2*, on a :

— Si $k \neq k'$:

$$\langle h_{p,k} | h_{p,k'} \rangle = \int_0^1 h_{p,k}(x) \cdot h_{p,k'}(x) = \int_0^1 0 \cdot dx = 0$$

— Si $k = k'$:

$$\begin{aligned}
\langle h_{p,k} | h_{p,k'} \rangle &= \int_0^1 (h_{p,k}(x))^2 dx \\
&= \int_0^{x_k} h_{p,k}(x))^2 dx + \int_{x_k}^{x_{k+1}} h_{p,k}(x))^2 dx + \int_{x_{k+1}}^1 h_{p,k}(x))^2 dx \\
&= \int_0^{x_k} 0^2 dx + \int_{x_k}^{x_{k+1}} 1^2 dx + \int_{x_{k+1}}^1 0^2 dx = 0 + [x]_{x_k}^{x_{k+1}} + 0 = x_{k+1} - x_k \\
&= \frac{1}{2^p}
\end{aligned}$$

Ainsi la base $(h_{p,0}, h_{p,1}, \dots, h_{p,2^p-1})$ est une base orthogonale ; ce qui fait des espaces \mathbb{K}_p des espaces euclidiens.

4.2.2 Propriété mathématique : Orthogonalité dans les espaces euclidiens

Énoncé Soient E un espace euclidien de dimension $n \geq 1$, $\langle \cdot | \cdot \rangle$ son produit scalaire et F un sous-espace vectoriel de E .

Alors F admet un supplémentaire orthogonal dans E et ce supplémentaire est unique. On le note : F^\perp .

Démonstration :

— Existence :

- Si $F = \{0_E\}$, on a $E = F \oplus E$ de manière immédiate. De plus, si $y \in F, y = 0_E$ et $\forall x \in E, \langle x | 0_E \rangle = 0$. D'où E est un supplémentaire orthogonal à F .
- Si $F = E$, par un raisonnement analogue, on trouve que $\{0_E\}$ est un supplémentaire orthogonal à F .
- Si $F \neq \{0_E\}$ et $F \neq E$, on considère $(e_i)_{1 \leq i \leq p}$ une base orthonormale de F avec $p = \dim F \in \mathbb{N}^*$.

L'ensemble $F^\perp = \{x \in E | \forall y \in F, \langle x | y \rangle = 0\}$ est par définition orthogonal à F .

Soit $x \in E$.

$$\exists (\lambda_1, \dots, \lambda_p) \in \mathbb{R}^p, x = \sum_{k=1}^p \lambda_k e_k + (x - \sum_{k=1}^p \lambda_k e_k)$$

Or le premier vecteur de la somme est dans F . Donc le second entre parenthèses appartient à F^\perp si et seulement si

$$\forall k \in [1, p], \langle e_k | x - \sum_{k=1}^p \lambda_k e_k \rangle = 0, \text{ c'est-à-dire } \lambda_k = \langle e_k | x \rangle$$

Avec x écrit de la manière suivante, on établit ainsi que $E = F + F^\perp$.

$$x = \sum_{k=1}^p \langle e_k | x \rangle \cdot e_k + (x - \sum_{k=1}^p \langle e_k | x \rangle \cdot e_k)$$

On a aussi immédiatement $F \cap F^\perp = \{0_E\}$, ce qui établit $E = F \oplus F^\perp$ et ainsi l'existence d'un supplémentaire orthogonal à F

— Unicité :

Soit G un sous-espace vectoriel supplémentaire de F dans E et orthogonal à F .

On a déjà $G \subset F^\perp$ puisque tous les vecteurs de G sont orthogonaux à tous les vecteurs de F .

De plus, $\dim G = \dim E - \dim F = \dim F^\perp$ car $\begin{cases} E = F \oplus F^\perp \\ E = F \oplus G \end{cases}$; on en déduit $G = F^\perp$ et l'unicité du supplémentaire orthogonal.

4.2.3 Utilisation de la propriété : décomposition orthogonale en somme directe.

Soit donc \mathbb{S}_p le supplémentaire orthogonal de \mathbb{K}_p dans \mathbb{K}_{p+1} .

On a $\mathbb{K}_{p+1} = \mathbb{S}_p \oplus \mathbb{K}_p$. D'où de proche en proche on arrive à :

$$\mathbb{K}_{p+1} = \mathbb{S}_p \oplus \mathbb{S}_{p-1} \oplus \mathbb{S}_{p-2} \oplus \dots \oplus \mathbb{S}_0 \oplus \mathbb{K}_0, \text{ soit encore } \mathbb{K}_p = \mathbb{K}_0 \oplus \bigoplus_{i=0}^{p-1} \mathbb{S}_i$$

On a défini à partir de \mathcal{H} la fonction $H_{p,k}$ par $H_{p,k}(x) = \mathcal{H}(2^p x - k)$; $(p, k) \in \mathbb{N}^2$.

$$\text{On alors } H_{p,k}(x) = \begin{cases} 1 & \text{si } x \in [\frac{k}{2^p}; \frac{k+1}{2^p}[\\ -1 & \text{si } x \in [\frac{k-1}{2^p}; \frac{k}{2^p}[\\ 0 & \text{dans les autres cas} \end{cases}$$

Facilement, on montre que $(h_{p,1}, h_{p,0}, \dots, h_{p,2^p-1}, H_{p,0}, H_{p,1}, \dots, H_{p,2^p-1})$ forme une base de \mathbb{K}_{p+1} .

De plus, on a :

— Si $k \neq k'$

$$\begin{aligned}
\langle h_{p,k} | H_{p,k'} \rangle &= \int_0^1 h_{p,k}(x) \cdot H_{p,k'}(x) \cdot dx \\
&= \int_0^{x_k} 0 dx + \int_{x_k}^{x_{k+1}} 1 \cdot 0 dx + \int_{x_{k+1}}^{x_{k'}} 0 dx + \\
&\quad \int_{x_{k'}}^{\frac{x_{k'} + x_{k'} + 1}{2}} 0 \cdot 1 dx + \int_{\frac{x_{k'} + x_{k'} + 1}{2}}^{x_{k'} + 1} 0(-1) dx + \int_{x_{k'} + 1}^1 0 \cdot dx \\
&= 0
\end{aligned}$$

— Si $k = k'$

$$\begin{aligned}
\langle h_{p,k} | H_{p,k'} \rangle &= \int_0^1 h_{p,k}(x) \cdot H_{p,k'}(x) \cdot dx \\
&= \int_0^{x_k} 0 dx + \int_{x_k}^{x_{k+1}} 1 \cdot H_{p,k}(x) dx + \int_{x_{k+1}}^1 0 \cdot dx \\
&= \int_{x_{k'}}^{\frac{x_{k'} + x_{k'} + 1}{2}} 1 dx + \int_{\frac{x_{k'} + x_{k'} + 1}{2}}^{x_{k'} + 1} -1 dx \\
&= 0.
\end{aligned}$$

De ce qui précède, il résulte que $(h_{p,1}, h_{p,0}, \dots, h_{p,2^p-1}, H_{p,0}, H_{p,1}, \dots, H_{p,2^p-1})$ forme une base orthogonale de \mathbb{K}_{p+1} .

Alors le système $(H_{p,0}, H_{p,1}, \dots, H_{p,2^p-1},)$ est une base orthogonale de l'orthogonal \mathbb{S}_p de \mathbb{K}_p dans \mathbb{K}_{p+1} .

De plus, on peut déjà remarquer :

$$\begin{aligned}
\langle H_{p,k} | H_{p,k} \rangle &= \int_0^1 (H_{p,k}(x))^2 \cdot dx = \int_{x_{k'}}^{\frac{x_{k'} + x_{k'} + 1}{2}} 1^2 \cdot dx + \int_{\frac{x_{k'} + x_{k'} + 1}{2}}^{x_{k'} + 1} (-1)^2 \cdot dx = \\
&\quad \frac{1}{2^p}
\end{aligned}$$

-Soit un signal $\Psi_p \in \mathbb{K}_p$.

$$\text{Alors } \exists! (\Psi_{p,0}, \Psi_{p,1}, \dots, \Psi_{p,2^p-1},) \in \mathbb{R}^{2^p}, \Psi_p = \sum_{k=0}^{2^p-1} \Psi_{p,k} h_{p,k}.$$

$$\text{Puisque } \mathbb{K}_p = \mathbb{K}_{p-1} \oplus \mathbb{S}_{p-1}, \exists! (\Psi_{p-1}, d_{p-1}) \in \mathbb{K}_{p-1} \times \mathbb{S}_{p-1}, \Psi_p = \Psi_{p-1} + d_{p-1}.$$

Et on peut décomposer Ψ_{p-1} et d_{p-1} comme suit :

$$\Psi_{p-1} = \sum_{k=0}^{2^p-1} \Psi_{p-1,k} h_{p-1,k} \text{ et } d_{p-1} = \sum_{k=0}^{2^p-1} d_{p-1,k} H_{p-1,k}.$$

4.2.4 Étape principale de l'algorithme : passage à la résolution inférieure, détermination des coefficients à la résolution inférieure.

-Déterminons les $\Psi_{p-1,k}$ et $d_{p-1,k}$:

Premières égalités L'orthogonalité de la base $(h_{p,1}, h_{p,0}, \dots, h_{p,2^p-1}, H_{p,0}, H_{p,1}, \dots, H_{p,2^p-1})$ avec $\Psi_p = \Psi_{p-1} + d_{p-1}$ et les résultats précédents sur les produits scalaires amène

$$\text{à : } \boxed{\langle \Psi_p | h_{p-1,k} \rangle = \frac{\Psi_{p-1,k}}{2^{p-1}}} \text{ et } \boxed{\langle \Psi_p | H_{p-1,k} \rangle = \frac{d_{p-1,k}}{2^{p-1}}}.$$

Démonstration On a

$\langle \Psi_p | h_{p-1,k} \rangle = \langle \Psi_{p-1} | h_{p-1,k} \rangle + \langle d_{p-1} | h_{p-1,k} \rangle$ par linéarité du produit scalaire.

$$\begin{aligned} &= \sum_{i \neq k} \Psi_{p-1,i} \langle h_{p-1,i} | h_{p-1,k} \rangle + \Psi_{p-1,k} \langle h_{p-1,k} | h_{p-1,k} \rangle + \sum_{i=0}^{2^{p-1}-1} d_{p-1,i} \langle H_{p-1,i} | h_{p-1,k} \rangle \\ &= \sum_{i \neq k} \Psi_{p-1,i} \cdot 0 + \frac{\Psi_{p-1,k}}{2^{p-1}} + \sum_{i=0}^{2^{p-1}-1} d_{p-1,i} \cdot 0 = \frac{\Psi_{p-1,k}}{2^{p-1}} \end{aligned}$$

Et on a

$$\begin{aligned} \langle \Psi_p | H_{p-1,k} \rangle &= \langle \Psi_{p-1} | H_{p-1,k} \rangle + \langle d_{p-1} | H_{p-1,k} \rangle \\ &= \sum_{i=0}^{2^{p-1}-1} \Psi_{p-1,i} \langle h_{p-1,i} | H_{p-1,k} \rangle + \sum_{i \neq k} d_{p-1,i} \langle H_{p-1,i} | H_{p-1,k} \rangle + d_{p-1,k} \langle H_{p-1,k} | H_{p-1,k} \rangle \\ &= \sum_{i=0}^{2^{p-1}-1} \Psi_{p-1,i} \cdot 0 + \sum_{i \neq k} d_{p-1,i} \cdot 0 + d_{p-1,k} \langle H_{p-1,k} | H_{p-1,k} \rangle \\ &= \frac{d_{p-1,k}}{2^{p-1}} \end{aligned}$$

Secondes égalités D'autre part, on peut montrer

$$\boxed{\langle \Psi_p | h_{p-1,k} \rangle = \frac{\Psi_{p,2k} + \Psi_{p,2k+1}}{2^p}}$$

$$\text{et } \boxed{\langle \Psi_p | H_{p-1,k} \rangle = \frac{\Psi_{p,2k} - \Psi_{p,2k+1}}{2^p}}$$

Démonstration On a $\langle h_{p,k} | h_{p-1,k'} \rangle = \int_0^1 h_{p,k}(x) \cdot h_{p-1,k'}(x) \cdot dx = \int_{x_k}^{x_{k+1}} h_{p-1,k'}(x) \cdot dx$

Or, on sait par définition que

$$h_{p-1,k'}(x) = \begin{cases} 1 & \text{si } x \in [\frac{k'}{2^{p-1}}; \frac{k'+1}{2^{p-1}}] \\ 0 & \text{sinon} \end{cases}$$

D'où :

$$\begin{cases} \langle h_{p,k} | h_{p-1,k'} \rangle = \int_{x_k}^{x_{k+1}} 1 \cdot dx = \frac{1}{2^p} \text{ si } \frac{k'}{2^{p-1}} \leq x_k \leq x_{k+1} \leq \frac{k'+1}{2^{p-1}}, \text{ ou encore } k \in \{2k', 2k'+1\} \\ \langle h_{p,k} | h_{p-1,k'} \rangle = \int_{x_k}^{x_{k+1}} 0 \cdot dx = 0 \text{ si } k \notin \{2k', 2k'+1\} \end{cases}$$

On a de même

$$\langle h_{p,k} | H_{p-1,k'} \rangle = \int_0^1 h_{p,k}(x) \cdot H_{p-1,k'}(x) \cdot dx = \int_{x_k}^{x_{k+1}} H_{p-1,k'}(x) \cdot dx$$

On sait par définition que

$$H_{p-1,k'}(x) = \begin{cases} 1 & \text{si } x \in [\frac{k'}{2^{p-1}}; \frac{k'+\frac{1}{2}}{2^{p-1}}] \\ -1 & \text{si } x \in [\frac{k'+\frac{1}{2}}{2^{p-1}}; \frac{k'+1}{2^{p-1}}] \\ 0 & \text{sinon} \end{cases}$$

D'où :

$$\begin{cases} \langle h_{p,k} | H_{p-1,k'} \rangle = \int_{x_k}^{x_{k+1}} 1 \cdot dx = \frac{1}{2^p} \text{ si } \frac{k'}{2^{p-1}} \leq x_k \leq x_{k+1} \leq \frac{k'+\frac{1}{2}}{2^{p-1}}, \text{ ou encore } k = 2k' \\ \langle h_{p,k} | H_{p-1,k'} \rangle = \int_{x_k}^{x_{k+1}} -1 \cdot dx = -\frac{1}{2^p} \text{ si } \frac{k'+\frac{1}{2}}{2^{p-1}} \leq x_k \leq x_{k+1} \leq \frac{k'+1}{2^{p-1}}, \text{ soit } k = 2k'+1 \\ \langle h_{p,k} | H_{p-1,k'} \rangle = \int_{x_k}^{x_{k+1}} 0 \cdot dx = 0 \text{ si } k \notin \{2k', 2k'+1\} \end{cases}$$

À partir de cela, il est facile de décomposer comme suit et d'obtenir les résultats :

$$\begin{aligned} \langle \Psi_p | h_{p-1,k} \rangle &= \left\langle \sum_{k=0}^{2^p-1} \Psi_{p,k} h_{p,k} | h_{p-1,k} \right\rangle \\ &= \sum_{i \notin \{2k', 2k'+1\}} \Psi_{p,i} \langle h_{p,i} | h_{p-1,k} \rangle + \sum_{i \in \{2k', 2k'+1\}} \Psi_{p,i} \langle h_{p,i} | h_{p-1,k} \rangle \\ &= 0 + \Psi_{p,2k} \cdot \frac{1}{2^p} + \Psi_{p,2k+1} \cdot \frac{1}{2^p} \end{aligned}$$

$$\begin{aligned}
\langle \Psi_p | H_{p-1,k} \rangle &= \left\langle \sum_{k=0}^{2^p-1} \Psi_{p,k} h_{p,k} | H_{p-1,k} \right\rangle \\
&= \sum_{i \notin \{2k', 2k'+1\}} \Psi_{p,i} \langle h_{p,i} | H_{p-1,k} \rangle + \sum_{i \notin \{2k', 2k'+1\}} \Psi_{p,i} \langle h_{p,i} | h_{p-1,k} \rangle \\
&= 0 + \Psi_{p,2k} \cdot \frac{1}{2^p} + \Psi_{p,2k+1} \cdot \frac{-1}{2^p}
\end{aligned}$$

Conclusion On obtient finalement avec les égalités encadrées les équations d'échelles suivantes.

$$\begin{cases} \Psi_{p-1,k} = \frac{\Psi_{p,2k} + \Psi_{p,2k}}{2} \| (\Psi_{p-1,k})_{k \in [0; 2^{p-1}-1]} \text{ est la famille des coefficients d'approximation à la résolution } 2^{p-1} \\ d_{p-1,k} = \frac{\Psi_{p,2k} - \Psi_{p,2k}}{2} \| (d_{p-1,k})_{k \in [0; 2^{p-1}-1]} \text{ est la famille des coefficients d'ondelettes} \end{cases}$$

Ainsi, lorsqu'on connaît les coefficients d'ondelettes à un niveau de résolution p , on peut aisément déterminer ceux du niveau $p-1$ et l'égalité des sous-espaces vectoriels en somme directe se comprend par :

$$\underbrace{\mathbb{K}_p}_{\text{Signal à la résolution } 2^p} = \underbrace{\mathbb{K}_{p-1}}_{\text{Signal à la résolution } 2^{p-1}} \oplus \underbrace{\mathbb{S}_{p-1}}_{\text{Détails (ou pertes)}}$$

L'intérêt principal de cet algorithme est qu'il permet de passer d'un échantillon de taille 2^p à un nouvel échantillon principal de taille 2^{p-1} et un échantillon de taille 2^{p-1} en utilisant que des sommes ou des différences.

4.3 Schématisation de l'algorithme de Mallat (compression d'un signal Ψ_p par des ondelettes)

$$\begin{array}{ccc}
& \text{Etape 1} & \\
\Psi_p & \rightarrow & \Psi_{p-1} \\
& \searrow & d_{p-1} \quad (\text{détails})
\end{array}$$

En réitérant le processus jusqu'à la dernière étape (étape p), on obtient la configuration suivante :

$$\begin{array}{ccccccc}
& \text{Etape 1} & & \text{Etape 2} & & \dots & \text{Etape p} \\
\Psi_p & \rightarrow & \Psi_{p-1} & \rightarrow & \Psi_{p-2} & \rightarrow & \Psi_0 \\
& \searrow & d_{p-1} & \searrow & d_{p-2} & \searrow & d_0
\end{array}$$

Ce travail est en réalité la première partie de l'algorithme, appelée *analyse*.

Il s'ensuit une deuxième partie appelée *synthèse*, qui correspond à l'opération inverse de l'analyse. Dans cette partie, les coefficients d'ondelettes « omis » dans l'analyse entraînent des erreurs.

Notons toutefois que l'algorithme posé par Stéphane Mallat se fonde sur la notion d'analyse multirésolution de $L^2(\mathbb{R})$ (qui a été d'ailleurs introduite afin de construire des bases orthogonales d'ondelettes). Il s'agit toutefois comme ici d'une suite de sous-espaces vectoriels fermés de l'espace $L^2(\mathbb{R})$ mais vérifiant certaines propriétés plus générales.

4.4 Représentation matricielle de l'algorithme utilisant les ondelettes de Haar

Une image peut être considérée comme un ensemble de pixels, chaque pixel représentant un niveau de gris si l'image est en noir et blanc, ou un niveau de rouge, de vert et de bleu si l'image est en couleur. On peut par conséquent représenter l'image par une matrice H_n carrée $2^n * 2^n$ de taille égale à la résolution de l'image.

Les équations d'échelle (c'est-à-dire le passage d'une résolution à la résolution inférieure) renseignent sur le type de matrice à utiliser dans l'algorithme spécifique de Haar.

$$H_2 = \begin{pmatrix} \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & -\frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 & -\frac{1}{2} \end{pmatrix}$$

est la matrice $4 * 4$ associée à l'algorithme utilisant les ondelettes de Haar.

On retrouve bien le fait que les deux premières colonnes (moitié gauche) représentent l'échantillon principal et que les deux dernières colonnes (moitié droite) de la matrice symbolisent les détails.

$$H_3 = \begin{pmatrix} \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 & -\frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & -\frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & -\frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & -\frac{1}{2} \end{pmatrix}$$

est la matrice $8*8$ associée à l'algorithme de Mallat.

L'intérêt du choix de telles matrices réside dans leur adaptation pour la multiplication matricielle (en raison de l'arrangement des nombres de la matrice suivant les colonnes et le nombre de zéros).

4.4.1 Exemple

On nomme M_2 une matrice 4×4 quelconque associée à une famille de pixels.

$$M_2 = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{pmatrix}$$

Alors on obtient la nouvelle matrice de pixels (représentant la résolution moitié) en effectuant le produit $M_2 \times H_2$.

On obtient

$$M_1 = M_2 \times H_2 = \begin{pmatrix} \frac{a+b}{2} & \frac{c+d}{2} & \frac{a-b}{2} & \frac{c-d}{2} \\ \frac{e+f}{2} & \frac{g+h}{2} & \frac{e-f}{2} & \frac{g-h}{2} \\ \frac{i+j}{2} & \frac{k+l}{2} & \frac{i-j}{2} & \frac{k-l}{2} \\ \frac{m+n}{2} & \frac{o+p}{2} & \frac{m-n}{2} & \frac{o-p}{2} \end{pmatrix}$$

On obtient alors en première moitié verticale de la matrice le nouvel échantillon principal et en seconde moitié les coefficients représentant les nouveaux détails.

On réitère ensuite le processus et on obtient finalement à partir d'une matrice initiale de pixels M_p les matrices $M_{p-1}, M_{p-2}, \dots, M_1, M_0$ avec la relation de récurrence $M_{k-1} = M_k \times H_p$ ($k \in \{p, p-1, \dots, 1\}$) et H_p désigne la matrice carrée $2^p \times 2^p$ spécifique à l'algorithme de Haar, choisie de telle sorte que son nombre p de colonnes et de lignes soit celui des colonnes et lignes de la matrice initiale de pixels).

En reprenant l'exemple précédent, il resterait à calculer $M_0 = M_1 \times H_2$.

On obtiendrait

$$M_0 = \begin{pmatrix} \frac{a+b+c+d}{4} & \frac{a+c-(b+d)}{4} & \frac{a+b-(c+d)}{4} & \frac{a+b-(b+c)}{4} \\ \frac{e+f+g+h}{4} & \frac{e+g-(f+h)}{4} & \frac{e+f-(g+h)}{4} & \frac{e+h-(f+g)}{4} \\ \frac{i+j+k+l}{4} & \frac{i+k-(j+l)}{4} & \frac{i+j-(k+l)}{4} & \frac{i+l-(j+k)}{4} \\ \frac{m+n+o+p}{4} & \frac{m+o-(n+p)}{4} & \frac{m+n-(o+p)}{4} & \frac{m+p-(n+o)}{4} \end{pmatrix}$$

Mais en pratique, pour chaque matrice M_k calculée, on ne garde que les coefficients supérieurs à une certaine précision choisie ϵ on effectue une *compression*. Les coefficients d'ondelettes inférieurs à cette précision sont remplacés par des 0. Lors de l'étape inverse de *décompression* ou synthèse, pour réobtenir la matrice initiale M_p , il suffit de calculer les nouvelles matrices M'_1, M'_2, \dots, M'_p par la relation de récurrence suivante :

$M'_{k+1} = M'_k \times (H_p)^{-1}$, avec $k \in \{0, 1, \dots, p-1\}$, $(H_p)^{-1}$ désigne la matrice inverse de H_p et $M'_0 = M_0$

En reprenant l'exemple précédent, on aurait :

$$(H_2)^{-1} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{pmatrix}$$

Bien sûr, la matrice finale M'_p se trouve être quelque peu différente de la matrice M_p puisque certains coefficients sont devenus des 0.

5 Application à la compression des données

La transformation en ondelettes se révèle très efficace pour transformer la plupart des signaux que l'on peut rencontrer, notamment les images et il est facile d'en comprendre la raison.

En effet, la majeure partie des informations à laquelle nous sommes sensibles se trouve dans les contours de l'image où l'intensité varie brutalement, et les coefficients d'ondelettes correspondants sont significatifs, y compris aux petites échelles.

Or, une image contient généralement relativement peu de contours, et est régulière (lentement variable) sauf au voisinage des contours. Par conséquent, beaucoup de coefficients d'ondelettes sont faibles (surtout aux petites échelles) ; les détails étant presque nuls, ils peuvent être négligés sans que cela entraîne de distorsion visible sur l'image.

Il suffit alors de s'imposer une précision ϵ . On ne va garder ainsi que les coefficients d'ondelettes supérieurs à ϵ . On dit alors qu'on effectue une compression du signal.

Il y a notamment des applications de la compression par ondelettes dans le domaine de l'imagerie médicale. Le cinéma numérique a quant à lui adopté le format JPEG 2000 qui utilise également la transformée en ondelettes.

La théorie des ondelettes symbolise en quelque sorte l'évolution des sciences mathématiques induite par l'introduction de l'outil informatique. Bien que l'analyse par les ondelettes soit encore loin de nous donner une réponse universelle et finale au problème de la représentation et du codage des signaux, elle se révèle être un outil mathématique particulièrement performant dans plusieurs domaines.

6 Utilisation pratique de la transformation par ondelettes discrète

Nous avons choisi de mettre en pratique ce que nous avons vu plus haut de manière théorique. Notre but a été de mettre en place un algorithme de compression d'image utilisant la compression par ondelettes, ainsi que des applications graphiques qui utilisent cet algorithme.

Pour que notre travail rentre dans le cadre du thème de cette année, qui est "Transfert, Échange", nous avons aussi créé une application client/serveur utilisant la compression par ondelettes.

Le code complet de nos algorithmes est disponible en annexe à la fin de ce dossier, mais aussi sur le Github de notre projet (cf. section Liens).

6.1 Le choix des outils

Pour nos programmes, nous avons utilisé le langage Python, ainsi que plusieurs bibliothèques.

6.1.1 Le langage : Python

Nous avons choisi d'utiliser le langage Python pour plusieurs raisons. Celui-ci offre beaucoup de possibilités, est facile à utiliser (et à comprendre) et permet de créer de gros projets assez rapidement. C'est aussi un langage approprié pour un débutant en programmation. C'est un très bon langage de prototypage, ce qui permet de donner un aperçu assez fonctionnel d'une application pour ensuite pouvoir la réaliser dans un autre langage (plus rapide, par exemple). C'est un langage de script, ce qui permet une grande flexibilité du code, et enlève l'étape de la compilation.

La version 2.7 de Python a été utilisée pour notre projet, car celle-ci est plus stable et plus mûre que la plus récente version 3. Aussi, beaucoup de modules Python assez utiles n'ont toujours pas été portés vers Python 3 à l'heure actuelle et nous ne voulions pas être freinés par cela.

6.1.2 La bibliothèque de traitement d'image : PIL

Pour que notre application puisse supporter plusieurs types de fichiers, nous avons eu recours à une bibliothèque graphique. Nous l'avons utilisée simplement afin de récupérer un tableau contenant les pixels d'une image, ce qui aurait été redondant à programmer nous-même.

PIL nous donne aussi accès à l'écriture de fichiers image dans tous les formats, ce qui offre de la flexibilité à notre programme.

Nous n'avons pas utilisé les autres fonctionnalités de cette bibliothèque, bien entendu, puisqu'il s'agissait avant tout de concevoir notre propre algorithme de compression.

6.1.3 La bibliothèque d'interface graphique : Tkinter

Pour la partie «interface graphique utilisateur» (ou GUI), nous avons utilisé la bibliothèque Tkinter, qui est incluse par défaut avec l'installation standard de Python. Elle est simple d'utilisation et convenait parfaitement à ce que nous voulions en faire, c'est-à-dire une simple application montrant la compression d'une image en utilisant notre algorithme.

Beaucoup d'autres bibliothèques existent pour la réalisation de GUI, mais celles-ci sont à télécharger en plus de Python, et nous ne voulions pas surcharger notre projet. De plus, les fonctionnalités qu'elles apportent n'auraient pas été utilisées dans le cadre de notre projet.

6.1.4 Le module `time`

Ce module nous a été utile lors du chronométrage de nos algorithmes et pour introduire des délais dans l’envoi de données dans le programme client/serveur. Nous avons préféré celui-ci à d’autres (comme `timeit`) pour sa simplicité d’utilisation et sa précision suffisante.

6.1.5 Le module `struct`

Le module `struct` a pour but de transformer des chaînes de caractères (et autres objets Python) en variables binaires (telles que celles utilisées par le langage C). Par exemple, le nombre "123" utilise 3 octets quand il est écrit en chaîne de caractères Python, alors que quand il est écrit en mémoire en tant que variable de type `byte`, il n’utilise qu’un octet. Nous avons donc utilisé cela pour la sauvegarde de coefficients d’ondelettes et les transferts d’images entre client et serveur. En effet, les valeurs RGB des pixels ne peuvent être supérieures à 255 et tiennent donc dans un seul octet. Ce qui nous donne au final des paquets et des fichiers 3 fois plus petits. Cela augmente aussi la vitesse des transferts.

6.1.6 Le module `socket`

Ce module a été utilisé pour transférer des images à travers le réseau. Nous avons préféré utiliser ce module plutôt que les sockets d’une librairie externe (telle que `PySFML`), car celui-ci est inclus dans Python et nous voulions que le programme reste simple.

6.2 Exemples d’images traitées avec notre algorithme

Nous avons mis au point un algorithme de compression utilisant des matrices. Celui-ci applique deux fois la transformation par ondelettes de Haar (une fois pour la hauteur, une fois pour la largeur), et stocke les coefficients d’ondelettes dans des matrices séparées de l’image. On peut ensuite supprimer certains de ces coefficients au-delà d’un certain seuil, pour compresser l’image.

La figure 6 montre l’image à laquelle nous avons appliqué la compression. La figure 7 montre un détail de l’image.

La figure 8 représente le même détail de l’image une fois que la compression a été appliquée avec un seuil assez petit pour garder la plupart des détails importants de l’image mais assez grand pour compresser les «aplats» de couleur, les ombres, etc. L’image compressée occupe 35% de mémoire en moins par rapport à l’image de départ. Ce qui montre que la compression par ondelettes est plutôt efficace et que notre algorithme est fonctionnel.

Sur la figure 9, on peut voir le même détail quand l’image a été compressée avec le seuil maximal. Ici, tous les détails ont été éliminés. Cela revient simplement à diviser la résolution de l’image par deux.



FIGURE 6 – L'image de départ

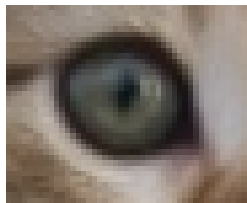


FIGURE 7 – Détail de l'image de départ

6.3 L'application graphique

L'application graphique que nous avons créée permet plusieurs choses :

- Ouverture d'une image
- Enregistrement d'une image

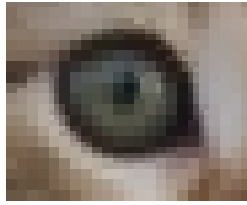


FIGURE 8 – Détail de l'image compressée

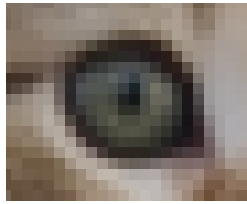


FIGURE 9 – Détail de l'image compressée à 100%

- Affichage d'une image dans une fenêtre
- Conversion d'une image en nuances de gris
- Compression d'une image par deux méthodes
- Réduction de la résolution d'une image de 50%
- Envoi d'une image à un serveur grâce au programme `client.py`

La figure 10 montre la fenêtre principale de l'application, avec une image en cours d'édition. L'interface est minimaliste, mais suffisante.

La figure 11 montre les menus de l'application, qui permettent de choisir entre toutes les actions décrites ci-dessus.

La figure 12 montre le sélecteur de seuil, qui apparaît lorsqu'on choisit «Compresser» ou «Compresser (new)» dans les menus.



FIGURE 10 – Fenêtre principale de l'application

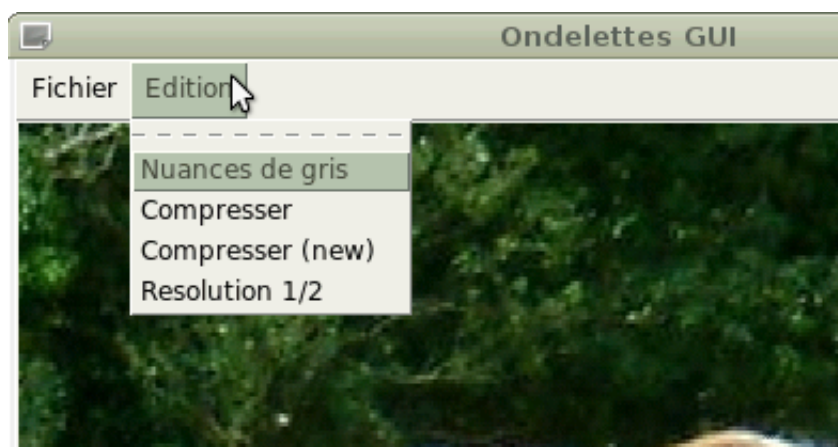


FIGURE 11 – Menus de l'application

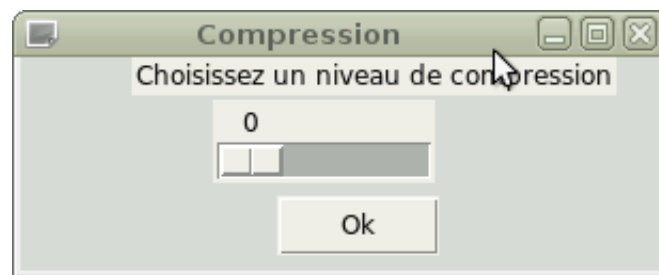


FIGURE 12 – Le sélecteur de seuil de compression

6.4 L'algorithme fasthaar

Nous avons mis au point un autre algorithme de compression n'utilisant pas de matrices. Celui-ci est plus rapide et moins gourmand en mémoire que **haar**, mais l'inconvénient est qu'il ne permet pas d'utiliser la transformation par ondelettes discrète à un niveau de récursivité plus grand que 1.

6.4.1 Fonctionnement

L'algorithme découpe l'image en carrés de 4 pixels, qu'il va traiter à la suite, colonne par colonne. Comme montré sur la figure 13

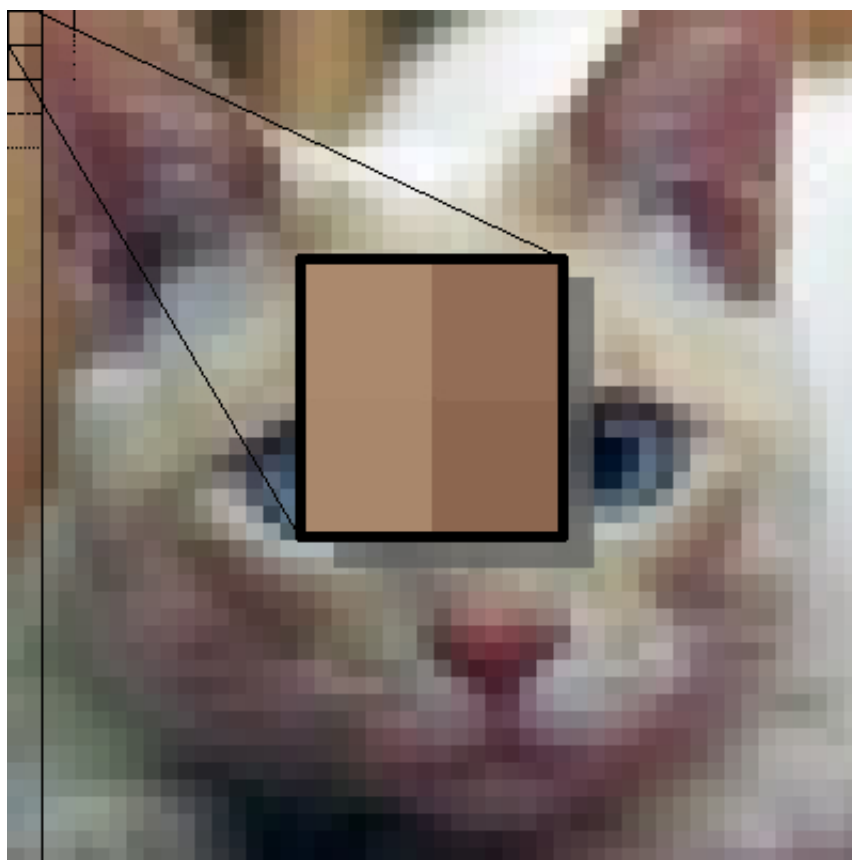


FIGURE 13 – Fonctionnement de l'algorithme

Analyse : Pour chaque carré de 4 pixels, l'algorithme applique la transformation par ondelettes discrète. On commence par stocker les valeurs RGB des 4 pixels dans un tableau. Nous allons, pour l'exemple, utiliser les 4 pixels mis en valeur sur la figure 13. Le tableau créé est montré par la figure 14.

R		G		B	
171	147	137	109	109	86
169	140	135	102	107	79

FIGURE 14 – Tableau créé par l'algorithme

L'algorithme remplace ensuite les valeurs des colonnes de gauche de chaque couleur par la moyenne de chaque ligne et les valeurs des colonnes de droite par la différence divisée par deux. Ce qui nous donne le tableau de la figure 15. Les coefficients d'ondelette sont surlignés.

R		G		B	
160	11	123	14	97.5	11.5
154.5	14.5	118.5	16.5	93	14

FIGURE 15 – Tableau après l'étape 1

Enfin, l'algorithme remplace la case en haut à gauche de chaque couleur par la moyenne des lignes et la case en bas à gauche par la différence divisée par deux. Le résultat est montré par la figure 16.

R		G		B	
157.25	11	120.75	14	95.25	11.5
2.75	14.5	2.25	16.5	2.25	14

FIGURE 16 – Tableau après l'étape 2

Compression et synthèse : Après la phase d'analyse vient la phase de compression. On va supprimer les coefficients d'ondelettes inférieurs à un certain seuil. Pour l'exemple, nous allons choisir un seuil de 12.

Après compression, le tableau est celui de la figure 17.

R		G		B	
157.25	0	120.75	14	95.25	0
0	14.5	0	16.5	0	14

FIGURE 17 – Tableau après compression

Ensuite, l'algorithme reconstitue l'image avec les coefficients restants. On effectue d'abord pour la colonne de gauche de chaque couleur une addition pour retrouver le coefficient du haut, et une soustraction pour le coefficient du bas.

Ce qui nous donne le tableau de la figure 18. Puisque les coefficients étaient égaux à 0, les pixels ne sont pas changés.

R		G		B	
157.25	0	120.75	14	95.25	0
157.25	14.5	120.75	16.5	95.25	14

FIGURE 18 – Tableau après synthèse sur les colonnes

L'algorithme effectue ensuite les mêmes opérations sur les lignes. Le résultat est donné par la figure 19

R		G		B	
157.25	157.25	134.75	106.75	95.25	95.25
171.75	143	137.25	104.25	109.25	81.25

FIGURE 19 – Tableau après synthèse sur les lignes

Enfin, on arrondit les valeurs à l'entier le plus proche. Le tableau final est donné par la figure 20. La figure 21 montre une comparaison entre l'état des pixels avant traitement, et l'état des pixels après.

R		G		B	
157	157	135	107	95	95
172	143	137	104	109	81

FIGURE 20 – Tableau après synthèse sur les lignes

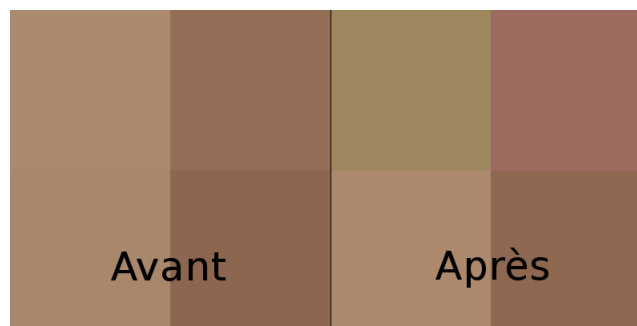


FIGURE 21 – Pixels avant et après traitement

Les pixels peuvent sembler réellement différents après traitement, mais la valeur moyenne des couleurs est conservée, ce qui, au final, donnera le même rendu quand les pixels seront à leur taille normale.

6.4.2 Coût et Performance

Nous avons été curieux de voir les performances de l'algorithme **fasthaar**.

Coût en mémoire : Cet algorithme est peu coûteux en mémoire, par rapport à la version matricielle. En effet, ce dernier devait stocker plusieurs matrices contenant l'image, les coefficients d'ondelettes, les différentes couleurs, etc. **fasthaar**, quant à lui, ne stocke que l'image et les valeurs de 4 pixels. Un pixel est un tuple contenant 3 entiers codés sur 8 bits (= 1 octet).

Soient L et l la longueur et la largeur de l'image à traiter. La place de l'algorithme **fasthaar** en mémoire est donc de $3 \times l \times L + 4 \times 3$ octets.

Coût temporel : Vu le principe de fonctionnement de **fasthaar**, on peut deviner que celui-ci a un coût en $O(l \cdot L)$ (ou $O(n^2)$ pour une image carrée de côté n). Nous avons créé un fichier **bench.py** servant à mesurer le temps que prend l'algorithme à traiter des images de différentes tailles. L'algorithme génère une image d'une taille donnée avec des pixels aléatoires puis lui applique **fasthaar** en chronométrant.

La figure 22 représente le temps de traitement d'une image (en secondes) en fonction de sa taille (pixels d'un côté divisé par 2). On vérifie donc que le coût est bien quadratique.

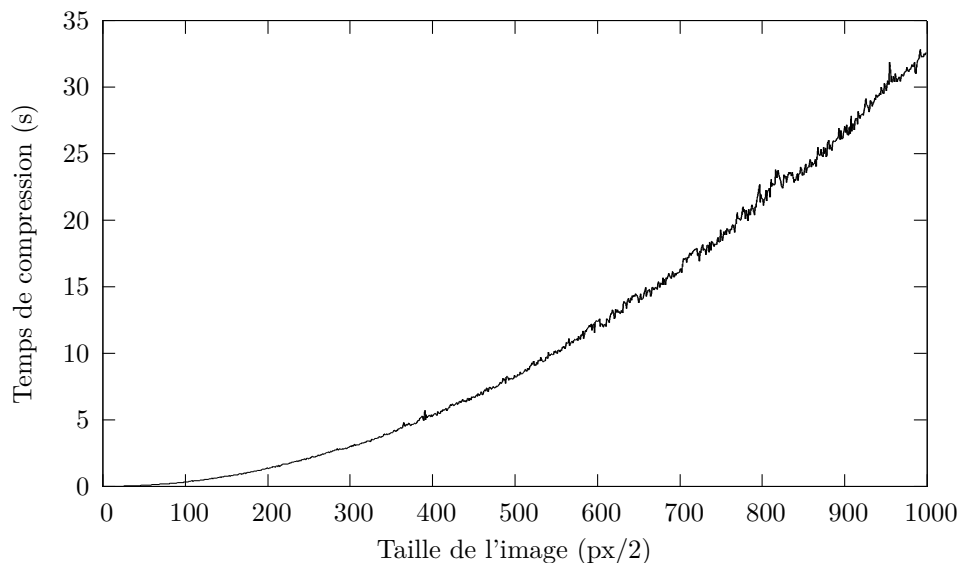


FIGURE 22 – Coût temporel de l'algorithme **fasthaar** en fonction de la taille de l'image

6.5 Transferts et échanges d'images par le réseau

Nous avons créé un programme serveur/client permettant de transférer des images et des coefficients d'ondelette par le réseau.

6.5.1 But du programme

Ce programme permet de stocker les coefficients d'ondelette d'une image sur un serveur distant pour ensuite ne garder sur la machine locale que les coefficients d'approximation (c-à-d l'image réduite de moitié sur chaque dimension). Ainsi une version légère de l'image est gardée pour que l'on puisse toujours savoir quelle image nous voulons (thumbnail), mais celle-ci occupe 4 fois moins de place en mémoire. Lorsque l'utilisateur veut afficher l'image en grandeur réelle, la machine locale demande les coefficients d'ondelette au serveur et reconstitue l'image de départ.

Ceci peut être très utile, par exemple dans le cas d'un périphérique de stockage amovible de taille réduite, tel qu'une clef USB - on peut ainsi stocker 4 fois plus d'images pour la même capacité.

6.5.2 Processus d'archivage

Étape 1 Le client ouvre l'image à stocker et encode les pixels, ainsi que les dimensions de l'image en binaire, puis envoie ces données au serveur par des sockets TCP. Chaque pixel de l'image occupe 3 bits, donc, si L et l sont les dimensions de l'image, le client doit envoyer $3 \cdot l \cdot L + 2$ octets au serveur.

Étape 2 Le serveur ouvre l'image qu'il vient de réceptionner pour la compresser et stocker les coefficients d'ondelette. Il applique pour cela une version modifiée de **fasthaar** qui ne garde que les coefficients d'ondelette et les encode en binaire grâce à **struct** pour les enregistrer dans un fichier.

Étape 3 Le client reçoit un signal de la part du serveur indiquant si oui ou non le transfert et la compression se sont bien déroulés. Si c'est le cas, le client crée une nouvelle image, de dimensions $\frac{L}{2}$ et $\frac{l}{2}$ et la remplit avec la moyenne des carrés de 4 pixels de l'image de départ (coefficients d'approximation).

La figure 23 montre le processus d'archivage.

6.5.3 Processus de restauration

Étape 1 Le client crée une image à partir de celle qui avait été stockée en doublant ses dimensions, ainsi, chaque pixel devient un carré de 4 pixels.

Étape 2 Le client demande les coefficients d'ondelette au serveur en spécifiant le nom de l'image. Le serveur lit le fichier qui contient les coefficients qu'il avait enregistrés et envoie directement les données au client.

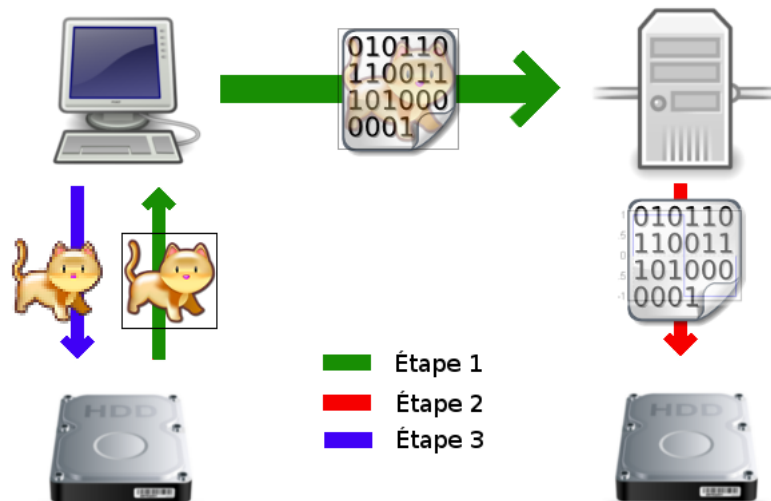


FIGURE 23 – Processus d’archivage

Étape 3 Le client décode les données binaires reçues et opère la transformation inverse en remplaçant les pixels de chaque carré par les valeurs qu’il obtient. Ensuite, il peut afficher l’image ou la stocker.

La figure 24 illustre ce processus

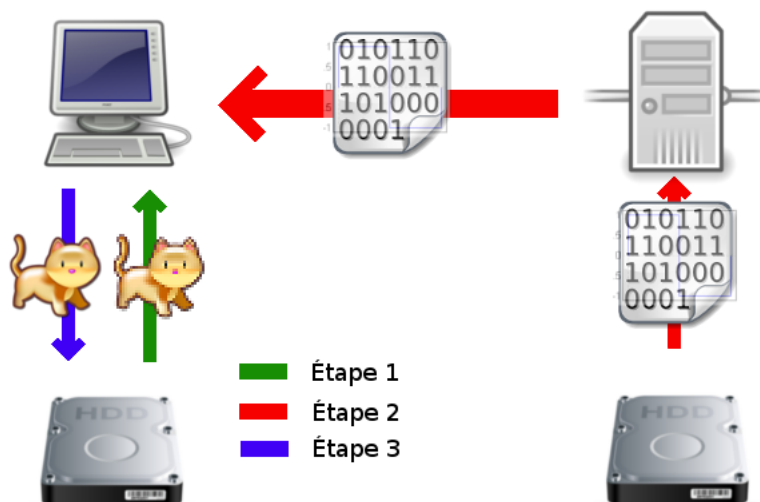


FIGURE 24 – Processus de restauration

7 Description du code

Ce qui suit est une description du code de tous nos programmes.

7.1 Le fichier `ondelettes.py`

Dans ce fichier, nous définissons des classes qui permettent le traitement des images. Il peut très bien être utilisé tout seul (sans l'interface graphique).

7.1.1 La classe `Matrice`

Au lieu d'utiliser la classe `Matrice` du module `numpy`, nous avons préféré créer la notre. En effet, nous n'avons pas besoin de toutes les fonctions qu'elle offre. Aussi, nous voulions faire un maximum de choses nous-mêmes.

La fonction `init` Cette fonction prend en argument la taille de la matrice et le premier élément qu'elle doit contenir pour ensuite créer un tableau de tableaux contenant ce premier élément. Elle enregistre aussi les dimensions de la matrice dans l'objet créé.

La fonction `transpose` Comme son nom l'indique, elle transforme la matrice en sa transposée. Elle est utilisée lors de la transformation par ondelettes de Haar.

La fonction `add` Elle permet l'addition de matrices. La matrice sur laquelle est appelée la fonction est remplacée par le résultat de l'addition. Elle n'est pas utilisée dans ce programme, mais pour que notre classe soit réutilisable dans d'autres projets, il fallait qu'elle soit créée.

La fonction `multiply` Cette fonction est analogue à la fonction `add` pour la multiplication.

La fonction `copy` Cette fonction transforme la matrice en celle qui est passée en argument.

La fonction `update` Cette fonction met à jour les dimensions de la matrice si le tableau a changé de taille.

Les fonctions `save` et `save1` Elles permettent de sauvegarder l'état de la matrice dans un tableau auxiliaire.

La fonction `restore` Elle remet en place le tableau sauvegardé par la fonction `save`.

7.1.2 La classe `MatriceImage`

Cette classe utilise la classe `Matrice` pour stocker une image afin de pouvoir travailler dessus. Elle contient également des fonctions de compression et d'enregistrement.

La fonction `init` Cette fonction prend en argument l'emplacement d'une image et la charge à l'aide de PIL. Elle crée puis remplit une `Matrice` avec les informations des pixels de l'image (cf. fonction `fill`).

La fonction `fill` Cette fonction stocke les pixels de l'image dans la `Matrice`.

Les fonctions `grayscalemean` et `grayscalemeanmatrix` Elles transforment l'image en nuances de gris en remplaçant chaque couleur de chaque pixel par la moyenne des trois composantes RGB du pixel. `grayscalemean` fait l'opération sur l'image directement alors que `grayscalemeanmatrix` la fait sur la `Matrice` de l'image.

Les fonctions `getmatrixred`, `getmatrixgreen` et `getmatrixblue` Créent des tableaux dans l'objet et les remplissent avec les composantes RGB (respectivement) de l'image. Cela permet de traiter chaque couleur séparément.

La fonction `save` Elle prend en argument une chaîne de caractère et sauvegarde l'image sous ce nom dans le dossier courant.

La fonction `ondelette_haar` Cette fonction prend en argument un tableau de valeurs et le nombre de récursions qu'elle doit faire. Elle renvoie le tableau des coefficients d'approximation ainsi que le tableau des coefficients d'ondelette.

Les fonctions `getcolonne` et `getligne` Ces fonctions servent à récupérer une liste contenant les données d'une colonne/d'une ligne, pour pouvoir ensuite être traitées par la fonction `ondelette_haar`.

La fonction `setcolonne` Elle permet de remettre une colonne dans une matrice après traitement.

Les fonctions `apply_haar_lig` et `apply_haar_col` Ces fonctions appliquent la transformation avec l'ondelette de haar à toutes les lignes/toutes les colonnes d'une matrice.

La fonction `haar_grayscale` Cette fonction convertit une image en nuances de gris, puis applique la transformation par ondelettes sur l'image. Elle l'applique d'abord sur les lignes, puis transpose la `Matrice`, puis la réapplique sur les lignes.

La fonction `update` Elle est similaire à celle qui a été décrite plus haut.

La fonction `create_coef_matrix` Elle crée les tableaux nécessaires à la mise en mémoire des coefficients d'ondelette de chaque couleur.

La fonction `haar` Elle a le même effet que la fonction `haar_{}grayscale`, mais elle s'applique aux 3 composantes de l'image.

La fonction `makeimage` Cette fonction crée une image à partir des matrices des coefficients d'approximation.

La fonction `compression` Cette fonction, une fois que les matrices de coefficients d'ondelette ont été créées, permet de supprimer les coefficients qui sont inférieurs au `epsilon` passé en paramètre.

Les fonctions `syntheseligne` et `synthesecolonne` Ces fonctions appliquent la transformation inverse à l'image, à partir des coefficients d'approximation et des coefficients d'ondelette qui ont été conservés par la fonction `compression`.

La fonction `clearimage` Elle supprime toutes les données d'une image en la remplissant d'une couleur définie.

La fonction `fasthaar` Cette fonction permet d'appliquer la transformation par ondelettes sans passer par les matrices. Pour cela elle prend tous les «carrés» de 4 pixels d'une image et leur applique la transformation et la compression. Elle est plus rapide que les autres fonctions, mais l'inconvénient est que la récursivité n'est pas permise. Elle est donc utilisée pour compresser légèrement une image. Les résultats restent tout de même impressionnants puisqu'elle peut faire gagner jusqu'à 45% d'espace en seulement quelques secondes.

7.2 Le fichier `ondelettesGUI.py`

Ce fichier est l'interface graphique du programme, qui requiert le fichier `ondelettes.py` pour fonctionner. En effet, l'interface graphique n'est qu'une façade pour les fonctions qui ont été décrites plus haut.

7.2.1 La classe `Appli`

Cette classe représente la fenêtre principale de l'application.

La fonction `init` Cette fonction définit simplement l'instance de la fenêtre en appelant la fonction `initUI` décrite plus bas.

La fonction `initUI` Cette fonction crée tous les éléments de la fenêtre (le titre et les menus, entre autres).

La fonction `askopenfilename` Cette fonction ouvre une boîte de dialogue demandant quel fichier ouvrir. La boîte de dialogue est incluse dans l'installation de Tkinter. Une fois que le fichier a été choisi, la fonction crée la zone où l'image sera affichée.

La fonction `asksaveasfilename` Cette fonction ouvre une boîte de dialogue demandant où enregistrer le fichier qui a été modifié.

Les fonctions `askcompression` et `askcompression2` Elles servent à ouvrir une boîte de dialogue du type `DialogScale` (défini plus bas) pour demander à l'utilisateur le seuil de compression à utiliser lors de l'appel des fonctions `compression` ou `compression2`.

Les fonctions `compression` et `compression2` Ces fonctions utilisent respectivement les fonctions `haar` et `fasthaar` de `ondelettes.py` pour réaliser la compression de l'image avec le seuil donné par les fonctions `askcompression` et `askcompression2`. L'ancienne image est ensuite effacée de l'écran et la nouvelle est affichée grâce à `displayimage` ;

La fonction `grayscale` Elle utilise les fonctions de `ondelettes.py` pour convertir l'image en nuances de gris.

La fonction `displayimage` Elle crée une zone pour afficher la nouvelle image qui a été créée par compression, puis l'affiche dans cette zone.

La fonction `onExit` Elle est appelée lors de la fermeture de l'application et est là pour être sûr que tout est fermé correctement.

7.2.2 La classe `DialogScale`

C'est la classe qui définit la boîte de dialogue de compression (Fig.12). Elle est construite sur la même base de fenêtre que la classe `Appli`.

La fonction `initUI` Cette fonction crée le texte de la boîte de dialogue, la règlette qui permet de choisir la compression, et le bouton «Ok».

La fonction `onScale` Elle est appelée lorsque l'utilisateur modifie la position de la règlette. Elle stocke la position de celle-ci en mémoire.

La fonction `ok` Elle est appelée lorsque l'utilisateur clique sur le bouton «ok». Elle sauvegarde la valeur de la position de la règlette, puis ferme la boîte de dialogue.

7.2.3 La fonction `main`

Elle crée simplement une instance de la classe `Appli` et l'exécute.

7.3 Le fichier `bench.py`

Ce fichier a été utilisé pour le chronométrage de l'algorithme `fasthaar`, expliqué plus haut.

7.3.1 La fonction `image_gen`

Cette fonction crée une instance d'une image PIL dont la taille est passée en paramètre.

7.3.2 La fonction `randomize`

Elle prend en paramètre une image créée par `image_gen`, pour la remplir de pixels aléatoires. Afin d'être sûr que l'algorithme `fasthaar` fera des calculs.

7.3.3 La fonction `bench_square`

Cette fonction prend en argument un tableau de nombres qui vont servir à créer des images de tailles différentes avec `image_gen`, puis les remplir avec `randomize` et enfin chronométrer le traitement de celles-ci avec le module `time` de Python.

7.3.4 La fonction `fasthaar`

C'est la même que celle du fichier `ondelettes.py` sauf qu'elle n'a pas besoin de la classe `ImageMatrice` pour fonctionner.

7.3.5 La fonction `main`

Elle appelle la fonction `bench_square` sur un tableau de nombres allant de 1 à 1000 (pour des images de 2 à 2000 pixels de côté).

7.4 Le fichier `serveur.py`

Ce fichier est celui qui est exécuté en permanence sur le serveur, comme décrit à la page 30.

7.4.1 La fonction `main`

C'est la fonction principale du programme qui contient la boucle d'exécution du serveur. Elle commence par créer un `socket` qui va attendre un client sur le port 13337.

Lors de la connexion d'un client, le serveur entre dans la boucle d'exécution, qui tourne jusqu'à ce que le client envoie "stop" au serveur. À chaque fois que le

client envoie un message au serveur, ce dernier en analyse les premières lettres pour connaître la commande que le client veut exécuter, puis l'exécute.

7.4.2 La fonction `recvImage`

Cette fonction est exécutée quand le client envoie "sending" au serveur. Le serveur récupère les dimensions de l'image, puis se met en attente des pixels de l'image, qu'il réceptionne dans une chaîne de caractères qui fait office de buffer. La réception se termine quand le client envoie "end". Le serveur décode ensuite la chaîne de caractère pixel par pixel en la parcourant par morceaux de 3 bytes, et insère les pixels décodés dans une image aux dimensions égales à celles qui ont été envoyées. Enfin, il enregistre l'image et signale au client que l'opération s'est complétée avec succès.

7.4.3 La fonction `fasthaar_srv`

C'est une fonction qui ressemble à `fasthaar` du fichier `ondelettes.py`, mais elle ouvre elle même un image à partir du nom envoyé par le client. Ensuite elle opère la transformation par ondelettes discrète et sauvegarde ceux-ci après les avoir encodés en binaire, uniquement s'ils sont supérieurs au paramètre optionnel epsilon (fixé à 0 par défaut, pour garder des images entières). Une fois que la transformation est terminée, le serveur le signale au client.

7.4.4 La fonction `sendCoef`

Cette fonction est appelée quand le client veut reconstituer une image. Il envoie alors `'coef nom_image'`. Le serveur ouvre le fichier qui contient les coefficients d'ondelette, et les envoie au client sans les décoder. Une fois que c'est terminé, le client envoie "ok" en binaire au serveur pour lui signaler que l'opération s'est bien déroulée.

7.5 Le fichier `client.py`

Ce fichier est exécuté par le l'utilisateur quand il veut demander quelque chose au serveur (soit demander le stockage d'une image, soit demander des coefficients). Celui-ci est appelé de cette façon :

```
./client.py [arguments:-rsd] image.jpg [optionnel : serveur]
```

7.5.1 La fonction `main`

Cette fonction a pour arguments ceux passés par l'utilisateur en ligne de commande. Ils peuvent être :

- **r** pour communiquer avec un serveur distant. Ce paramètre peut être omis pour établir la connection à un serveur local.
- **s** pour stocker une image. C'est à dire : envoi de l'image au serveur, compression de l'image par le serveur, enregistrement de l'image réduite par le client.

- **d** pour récupérer les coefficients d'ondelette d'une image et la reconstituer.

Dans le cas de l'utilisation de la commande **r**, on peut utiliser un serveur autre que celui défini par défaut en en spécifiant l'adresse après le nom de l'image à traiter. Une fois que l'échange client/serveur est terminé, la fonction envoie **"stop"** au serveur, lui signifiant la fin de la connexion.

7.5.2 La fonction **sendImage**

Cette fonction est appelée lors de l'utilisation de la commande **s**, elle envoie au serveur **"sending"** suivi des dimensions du nom de l'image et de ses dimensions. Ensuite l'image est encodée en binaire puis envoyée au serveur. Le client signale ensuite que le transfert est terminé en envoyant **"end"** au serveur. Le client affiche ensuite la réponse du serveur.

7.5.3 La fonction **askcompress**

C'est la deuxième fonction appelée quand la commande **s** est utilisée. Elle demande au serveur de compresser l'image qui a été envoyée plus tôt, avec **"compress"** suivi du nom de l'image. Le client affiche ensuite la réponse du serveur.

7.5.4 La fonction **storeImage**

C'est la dernière fonction appelée quand la commande **s** est utilisée. Cette fonction crée une image dont les dimensions ont été divisées par deux (par rapport à l'image originale) et y copie les coefficients d'approximation. Cette image est très légère et peut être affichée.

7.5.5 La fonction **askcoeff**

Cette fonction est appelée lors de l'utilisation de la commande **d**. Elle demande au serveur d'envoyer les coefficients d'ondelette de l'image. La fonction décode ensuite les coefficients reçus, puis ouvre l'image réduite qui avait été enregistrée par **storeImage**. Le client crée une image de la taille de l'image originale, puis opère la transformation par ondelettes inverse. Après ça, le client enregistre l'image.

8 Annexes

Tout le code de ce projet est sous licence Creatice Commons BY-SA (voir p.56).



8.1 Fichier ondelettes.py

```
1  #
2  # Name:      ondelettes.py
3  # Purpose:   definition de classes de traitement d'image
4  #
5  # Author:    Gaetan
6  #
7  # Created:   10/07/2013
8  # Copyright: (c) Gaetan Bahl/Xavier Friederich 2013
9  # Licence:   CC-BY-SA
10 #
11
12 import Image, math
13 import time
14
15
16 class Matrice:
17
18     def __init__(self,x,y,what):
19         self.tableau = []
20         self.x, self.y = x,y
21
22         for i in range(self.x):
23             self.tableau.append([])
24             for j in range(self.y):
25                 self.tableau[i].append(what)
26
27     def transpose(self):
28         self.tableau = [[row[i] for row in self.tableau] for i in
29                         range(self.y)]
30         self.x,self.y = self.y,self.x
31
32     def add(self,matrice):
33
34         for i in range(n):
35             for j in range(p):
36                 self.tableau[i][j] += matrice.tableau[i][j]
37
38     def multiply(self,matrice):
39
40         for i in range(self.x):
41             for j in range(self.y):
42                 somme = 0
43                 for k in range(self.y):
```

```

43         somme += self.tableau[i][k]*matrice.tableau[k][
44             j]
45     def copy(self,matrice):
46
47         for i in range(self.x):
48             for j in range(self.y):
49                 self.tableau[i][j] = matrice.tableau[i][j]
50
51     def update(self):
52         self.x = len(self.tableau)
53         self.y = len(self.tableau[0])
54
55     def save(self):
56         self.tableau2 = list(self.tableau)
57
58     def save1(self):
59         self.mat_orig = list(self.tableau)
60
61     def restore(self):
62         self.tableau = list(self.mat_orig)
63
64 class MatriceImage():
65
66     def __init__(self,lienimage):
67         self.lienimage = lienimage
68         self.image = Image.open(lienimage)
69         self.pix = self.image.load()
70         self.sizeX,self.sizeY = self.image.size
71         self.matrice = Matrice(self.sizeX,self.sizeY,self.pix[0,0])
72         self.fill()
73
74     def fill(self):
75         for i in range(self.sizeX):
76             for j in range(self.sizeY):
77                 self.matrice.tableau[i][j] = self.pix[i,j]
78
79     def grayscalemean(self):
80         for i in range(self.sizeX):
81             for j in range(self.sizeY):
82                 mean = (self.pix[i,j][0] + self.pix[i,j][1] + self.
83                     pix[i,j][2])/3
84                 self.pix[i,j] = (mean,mean,mean)
85
86     def grayscalemeanmatrix(self):
87         self.matrixgray = Matrice(self.sizeX,self.sizeY,0)
88         for i in range(self.sizeX):
89             for j in range(self.sizeY):
90                 mean = (self.pix[i,j][0] + self.pix[i,j][1] + self.
91                     pix[i,j][2])/3
92                 self.matrixgray.tableau[i][j] = mean
93
94     def getmatrixred(self):
95         self.matrixred = Matrice(self.sizeX,self.sizeY,self.pix
96             [0,0][0])
97         for i in range(self.sizeX):
98             for j in range(self.sizeY):

```



```

96         self.matrixred.tableau[i][j] = self.matrice.tableau
97             [i][j][0]
98     def getmatrixblue(self):
99         self.matrixblue = Matrice(self.sizeX, self.sizeY, self.pix
100             [0,0][2])
101         for i in range(self.sizeX):
102             for j in range(self.sizeY):
103                 self.matrixblue.tableau[i][j] = self.matrice.
104                     tableau[i][j][2]
105     def getmatrixgreen(self):
106         self.matrixgreen = Matrice(self.sizeX, self.sizeY, self.pix
107             [0,0][1])
108         for i in range(self.sizeX):
109             for j in range(self.sizeY):
110                 self.matrixgreen.tableau[i][j] = self.matrice.
111                     tableau[i][j][1]
112     def save(self, nom):
113         self.image.save(nom)
114     def ondelette_haar(self, tableau_valeurs, ordre):
115         longueur = len(tableau_valeurs)
116         coeff_approx, coeff_ondelettes = [], []
117         for i in range(longueur/2):
118             coeff_approx.append((tableau_valeurs[2*i] +
119                 tableau_valeurs[2*i+1])/2)
120             coeff_ondelettes.append((tableau_valeurs[2*i] -
121                 tableau_valeurs[2*i+1])/2)
122         if ordre == 1:
123             return coeff_approx, coeff_ondelettes
124         else:
125             return ondelette_haar(coeff_approx, ordre - 1),
126                 coeff_ondelettes
127     def getcolonne(self, tab, num):
128         col = []
129         for i in range(len(tab)):
130             col.append(tab[i][num])
131         return col
132     def setcolonne(self, matrice, tab, num):
133         for i in range(len(tab)):
134             matrice[num][i] = tab[i]
135     def getligne(self, tab, num):
136         return tab[num]
137     def apply_haar_lig(self, matrice, chiffre):
138         for i in range(len(matrice)):
139             matrice[i] = self.ondelette_haar(matrice[i], 1)[chiffre]
140     def apply_haar_col(self, matrice):
141         for i in range(len(matrice[0])):

```

```

145         col = self.getcolonne(matrice, i)
146         col = self.ondelette_haar(col, 1)[0]
147         self.setcolonne(matrice, col, i)
148
149     def makeimagegray(self, matriceimage):
150         matriceimage.update()
151         im = Image.new("RGB", (matriceimage.x, matriceimage.y), "
            white")
152         pix = im.load()
153
154         for i in range(matriceimage.x):
155             for j in range(matriceimage.y):
156                 pix[i, j] = (int(matriceimage.tableau[i][j]), int(
                    matriceimage.tableau[i][j]), int(matriceimage.
                        tableau[i][j]))
157
158         return im
159
160     def haar_grayscale(self):
161         self.grayscalemeanmatrix()
162
163         self.apply_haar_lig(self.matrixgray.tableau, 0)
164         self.matrixgray.update()
165         self.matrixgray.transpose()
166         self.matrixgray.update()
167         self.apply_haar_lig(self.matrixgray.tableau, 0)
168         self.matrixgray.update()
169         self.matrixgray.transpose()
170         self.imagehaargray = self.makeimagegray(self.matrixgray)
171
172     def update(self):
173         self.size_x = matrice.x
174         self.size_y = matrice.y
175
176     def create_coef_matrix(self):
177         self.matrixcoefr = Matrice(self.matrixred.x, self.matrixred.
            y, 0)
178         self.matrixcoefg = Matrice(self.matrixred.x, self.matrixred.
            y, 0)
179         self.matrixcoefb = Matrice(self.matrixred.x, self.matrixred.
            y, 0)
180         self.matrixcoefr.copy(self.matrixred)
181         self.matrixcoefg.copy(self.matrixgreen)
182         self.matrixcoefb.copy(self.matrixblue)
183
184     def haar(self):
185
186         for i in [self.matrixred, self.matrixgreen, self.matrixblue]:
187             self.apply_haar_lig(i.tableau, 0)
188             i.update()
189             i.save()
190             i.transpose()
191             i.update()
192             self.apply_haar_lig(i.tableau, 0)
193             i.update()
194             i.transpose()
195

```

```

196         for i in [self.matrixcoefr, self.matrixcoefg, self.
197                   matrixcoefb]:
198             i.save1()
199             self.apply_haar_lig(i.tableau, 1)
200             i.update()
201             i.save()
202             i.update()
203             i.restore()
204             self.apply_haar_lig(i.tableau, 0)
205             i.update()
206             i.transpose()
207             i.update()
208             self.apply_haar_lig(i.tableau, 1)
209             i.update()
210             i.transpose()
211
212
213     def makeimage(self):
214         im = Image.new("RGB", (self.matrixred.x, self.matrixred.y),
215                          "white")
216         pix = im.load()
217
218         for i in range(self.matrixred.x):
219             for j in range(self.matrixred.y):
220                 pix[i, j] = (int(self.matrixred.tableau[i][j]), int(
221                     self.matrixgreen.tableau[i][j]), int(self.
222                     matrixblue.tableau[i][j]))
223
224         return im
225
226
227     def compression(self, epsilon):
228         for tab in [self.matrixcoefr.tableau, self.matrixcoefg.
229                     tableau, self.matrixcoefb.tableau]:
230             for i in tab:
231                 for j in range(len(i)):
232                     if abs(i[j]) < epsilon:
233                         i[j] = 0
234
235         for tab in [self.matrixcoefr.tableau2, self.matrixcoefg.
236                     tableau2, self.matrixcoefb.tableau2]:
237             for i in tab:
238                 for j in range(len(i)):
239                     if abs(i[j]) < epsilon:
240                         i[j] = 0
241
242
243     def syntheseselignes(self):
244         for i in range(self.sizey/2):
245             for j in range(self.sizey/2):
246                 self.pix[2*i, 2*j] = (int(self.matrixred.tableau[i][
247                     j] + self.matrixcoefr.tableau[i][j]), int(self.
248                     matrixgreen.tableau[i][j] + self.matrixcoefg.
249                     tableau[i][j]), int(self.matrixblue.tableau[i
250                     ][j] + self.matrixcoefb.tableau[i][j]))
251                 self.pix[2*i+1, 2*j] = (int(self.matrixred.tableau[i
252                     ][j] - self.matrixcoefr.tableau[i][j]), int(
253                     self.matrixgreen.tableau[i][j] - self.
254                     matrixcoefg.tableau[i][j]), int(self.

```

```

matrixblue.tableau[i][j] - self.matrixcoefb.
tableau[i][j]))
240
241 def synthesecolonnes(self):
242
243     for y in range(len(self.matrixcoefr.tableau2[0])):
244
245         for x in range(len(self.matrixcoefr.tableau2)):
246
247             self.pix[x,2*y],self.pix[x,2*y+1] = (int(self.
                pix[x,2*y][0] + self.matrixcoefr.tableau2[x][y
                ]), int(self.pix[x,2*y][1] + self.matrixcoefg.
                tableau2[x][y]), int(self.pix[x,2*y][2] + self
                .matrixcoefb.tableau2[x][y]),(int(self.pix[x
                ,2*y][0] - self.matrixcoefr.tableau2[x][y]),
                int(self.pix[x,2*y][1] - self.matrixcoefg.
                tableau2[x][y]), int(self.pix[x,2*y][2] - self
                .matrixcoefb.tableau2[x][y]))
248
249 def clearimage(self):
250     for i in range(self.sizeX):
251         for j in range(self.sizeY):
252             self.pix[i,j] = (255,0,0)
253
254 def fasthaar(self, epsilon, xa,xb,ya,yb):
255
256     for x in range(xa/2,xb/2):
257
258         for y in range(ya/2,yb/2):
259             #copier les 4 pixels dans un carre
260
261             carres = []
262
263             for i in range(3):
264                 carres.append( [ [self.pix[2*x,2*y][i],self.
                    pix[2*x,2*y+1][i] ], [ self.pix[2*x
                    +1,2*y][i], self.pix[2*x+1,2*y+1][i]]])
265
266             ##ANALYSE
267             ondlhaut = [(carres[i][0][0] - carres[i][1][0])/2
                for i in range(3)]
268             ondlbas = [(carres[i][0][1] - carres[i][1][1])/2
                for i in range(3)]
269
270             for i in range(3):
271                 carres[i][0][0] = (carres[i][0][0] + carres[i
                    ][1][0])/2
272                 carres[i][0][1] = (carres[i][0][1] + carres[i
                    ][1][1])/2
273
274             ondlmix = [(carres[i][0][0] - carres[i][0][1])/2
                for i in range(3)]
275
276             for i in range(3):
277                 carres[i][0][0] = (carres[i][0][0] + carres[i
                    ][0][1])/2
278

```

```

279     ##SYNTHESE
280     for i in range(3):
281         if abs(ondlmix[i]) < epsilon:
282             carres[i][0][1] = carres[i][0][0]
283         else:
284             carres[i][0][1] = carres[i][0][0] - ondlmix
285             carres[i][0][0] = carres[i][0][0] + ondlmix
286
287             if abs(ondlhaut[i]) < epsilon:
288                 carres[i][1][0] = carres[i][0][0]
289             else:
290                 carres[i][1][0] = carres[i][0][0] -
291                 ondlhaut[i]
292                 carres[i][0][0] = carres[i][0][0] +
293                 ondlhaut[i]
294
295                 if abs(ondlbas[i]) < epsilon:
296                     carres[i][1][1] = carres[i][0][1]
297                 else:
298                     carres[i][1][1] = carres[i][0][1] - ondlbas
299                     carres[i][0][1] = carres[i][0][1] + ondlbas
300
301             for i in [2*x, 2*x+1]:
302                 for j in [2*y, 2*y+1]:
303                     self.pix[i, j] = (carres[0][i-2*x][j-2*y],
304                     carres[1][i-2*x][j-2*y], carres[2][i-2*x]
305                     [j-2*y])
306
307     def main():
308         image = MatriceImage("chat.jpg")
309         start = time.time()
310
311         image.fasthaar(2, 0, image.size[0], image.size[1])
312
313         end = time.time()
314         print end - start
315         image.image.save("chat_compress.jpg", 'JPEG', quality = 100)
316
317     if __name__ == '__main__':
318         main()

```

8.2 Fichier ondelettesGUI.py

```

1  #
2  # Name:         ondelettesGUI.py
3  # Purpose:      interface graphique du TIPE sur les ondelettes
4  #
5  # Author:       Gaetan Bahl

```

```

6  #
7  # Created:      08/08/2013
8  # Copyright:    (c) Gaetan Bahl 2013
9  # Licence:      CC-BY-SA
10 #
11
12 from Tkinter import *
13 import tkFileDialog, ImageTk
14 from ondelettes import *
15 from ttk import Frame, Style
16 from Tkconstants import *
17 from subprocess import call
18
19
20 class Appli(Frame):
21
22     def __init__(self, parent):
23         Frame.__init__(self, parent)
24
25         self.parent = parent
26         self.initUI()
27
28     def initUI(self):
29
30         self.parent.title("Ondelettes_GUI")
31         self.pack(fill=BOTH, expand=1)
32         menubar = Menu(self.parent)
33         self.parent.config(menu=menubar)
34
35         fileMenu = Menu(menubar)
36         fileMenu.add_command(label="Ouvrir", command=self.
37                               askopenfilename)
38         fileMenu.add_command(label="Enregistrer", command=self.
39                               asksaveasfilename)
40         fileMenu.add_command(label="Exit", command=self.onExit)
41         fileMenu.add_command(label="Send_to_Server", command=self.
42                               send)
43         menubar.add_cascade(label="Fichier", menu=fileMenu)
44
45         editMenu = Menu(menubar)
46         editMenu.add_command(label="Nuances_de_gris", command=self.
47                               grayscale)
48         editMenu.add_command(label="Compresser", command=self.
49                               askcompression)
50         editMenu.add_command(label="Compresser_(new)", command=self.
51                               .askcompression2)
52         editMenu.add_command(label="Resolution_1/2", command=self.
53                               onExit)
54         menubar.add_cascade(label="Edition", menu=editMenu)
55
56         Style().configure("TFrame", background="#FFF")
57
58     def askopenfilename(self):
59
60         filename = tkFileDialog.askopenfilename(defaultextension =
61           "jpg")
62         self.filename = filename

```

```

55         self.matriceimage = MatriceImage(filename)
56         self.image = self.matriceimage.image
57         self.imgtk = ImageTk.PhotoImage(self.image)
58         self.labelimg = Label(self, image=self.imgtk)
59         self.labelimg.image = self.imgtk
60         self.labelimg.place(x = 0,y=0)
61         self.labelimg.pack()
62
63
64         self.parent.geometry(str(self.matriceimage.size+5)+"x"+str
65                                 (self.matriceimage.sizey+5)+"100+300")
66
67     def asksaveasfilename(self):
68
69         filename = tkFileDialog.asksaveasfilename(defaultextension
70                                                     = ".jpg")
71
72         self.image.save(filename, 'JPEG', quality = 100)
73
74     def askcompression(self):
75         global compress
76         fen = Tk()
77         fen.geometry("300x100+300+300")
78         box = DialogScale(fen)
79         fen.mainloop()
80         fen.destroy()
81         self.compression()
82
83     def askcompression2(self):
84         global compress
85         fen = Tk()
86         fen.geometry("300x100+300+300")
87         box = DialogScale(fen)
88         fen.mainloop()
89         fen.destroy()
90         self.compression2()
91
92     def compression(self):
93
94         self.matriceimage.getmatrixblue()
95         self.matriceimage.getmatrixgreen()
96         self.matriceimage.getmatrixred()
97         self.matriceimage.create_coef_matrix()
98         self.matriceimage.haar()
99         self.matriceimage.compression(compress)
100        self.matriceimage.syntheselignes()
101        self.matriceimage.synthesecolonne()
102        self.labelimg.destroy()
103
104        self.displayimage()
105
106    def compression2(self):
107
108        self.matriceimage.fasthaar(compress, 0, self.matriceimage.
109                                   size, 0, self.matriceimage.sizey)

```

```

109         self.labelimg.destroy()
110
111         self.displayimage()
112
113     def grayscale(self):
114         self.matriceimage.grayscalemeanmatrix()
115         self.image = self.matriceimage.makeimagegray(self.
            matriceimage.matrixgray)
116         self.matriceimage.image = self.matriceimage.makeimagegray(
            self.matriceimage.matrixgray)
117         self.labelimg.destroy()
118
119         self.displayimage()
120
121     def displayimage(self):
122         self.imgtk = ImageTk.PhotoImage(self.matriceimage.image)
123         self.labelimg = Label(self, image=self.imgtk)
124         self.labelimg.image = self.imgtk
125         self.labelimg.place(x = 0,y=0)
126         self.labelimg.pack()
127         self.update()
128
129     def send(self):
130         call(["./client.py", "sr", self.filename])
131
132
133
134     def onExit(self):
135         self.quit()
136
137
138 class DialogScale(Frame):
139     def __init__(self, parent):
140         Frame.__init__(self, parent)
141
142         self.parent = parent
143         self.initUI()
144
145     def initUI(self):
146
147         self.parent.title("Compression")
148         self.style = Style()
149         self.style.theme_use("default")
150
151         self.pack(fill=BOTH, expand=1)
152
153         scale = Scale(self, from_=0, to=255,command=self.onScale,
            orient= HORIZONTAL)
154         scale.place(x=90, y=20)
155
156
157         self.label2 = Label(self, text="Choisissez un niveau de
            compression")
158         self.label2.place(x=52, y=0)
159         self.quitButton = Button(self, text="Ok",command=
            self.ok)
160         self.quitButton.place(x=120, y=65)

```



```

161
162     def onScale(self, val):
163
164         self.variable = int(val)
165
166     def ok(self):
167         global compress
168         compress = self.variable
169         self.quit()
170
171
172 def main():
173
174     root = Tk()
175     root.geometry("250x250+300+300")
176     app = Appli(root)
177     root.mainloop()
178
179
180 if __name__ == '__main__':
181     main()

```

8.3 Fichier bench.py

```

1  #
2  # Name:      bench.py
3  # Purpose:   benchmarking d'algorithmes
4  #
5  # Author:    Gaetan Bahl
6  #
7  # Created:   19/08/2013
8  # Copyright: (c) Gaetan Bahl 2013
9  # Licence:   CC-BY-SA
10 #
11
12
13 import Image
14 import time
15 import random
16 from ondelettes import *
17
18
19 def image_gen(x):
20     im = Image.new("RGB", (x, x), "white")
21     return im
22
23
24 def randomize(pix, x):
25     for i in range(x):
26         for j in range(x):
27             pix[i, j] = (random.randrange(255), random.randrange(255),
28                          random.randrange(255))
29
30

```

```

31 def bench_square(ran):
32
33     for i in ran:
34         image = image_gen(2 * i)
35         pix = image.load()
36         randomize(pix, 2 * i)
37         start = time.time()
38         fasthaar(pix, 0, 0, 2 * i, 0, 2 * i)
39         end = time.time()
40         print str(i) + "_" + str(end - start)
41
42
43 def fasthaar(pix, epsilon, xa, xb, ya, yb):
44
45     for x in range(xa / 2, xb / 2):
46
47         for y in range(ya / 2, yb / 2):
48
49             carres = []
50
51             for i in range(3):
52                 carres.append([[pix[2 * x, 2 * y][i], pix[2 * x, 2
53                     * y + 1][i]],
54                     [pix[2 * x + 1, 2 * y][i], pix[2 * x + 1, 2 *
55                         y + 1][i]]])
56
57             ondlhaut = [(carres[i][0][0] - carres[i][1][0]) / 2
58                 for i in range(3)]
59             ondlbas = [(carres[i][0][1] - carres[i][1][1]) / 2
60                 for i in range(3)]
61
62             for i in range(3):
63                 carres[i][0][0] = (carres[i][0][0] + carres[i
64                     ][1][0]) / 2
65                 carres[i][0][1] = (carres[i][0][1] + carres[i
66                     ][1][1]) / 2
67
68             ondlmix = [(carres[i][0][0] - carres[i][0][1]) / 2
69                 for i in range(3)]
70
71             for i in range(3):
72                 carres[i][0][0] = (carres[i][0][0] + carres[i
73                     ][0][1]) / 2
74
75             for i in range(3):
76                 if abs(ondlmix[i]) < epsilon:
77                     carres[i][0][1] = carres[i][0][0]
78                 else:
79                     carres[i][0][1] = carres[i][0][0] - ondlmix[i]
80                     carres[i][0][0] = carres[i][0][0] + ondlmix[i]
81
82             if abs(ondlhaut[i]) < epsilon:
83                 carres[i][1][0] = carres[i][0][0]
84             else:
85                 carres[i][1][0] = carres[i][0][0] - ondlhaut[i]
86                 carres[i][0][0] = carres[i][0][0] + ondlhaut[i]

```

```

83         if abs(ondlbas[i]) < epsilon:
84             carres[i][1][1] = carres[i][0][1]
85         else:
86             carres[i][1][1] = carres[i][0][1] - ondlbas[i]
87             carres[i][0][1] = carres[i][0][1] + ondlbas[i]
88
89     for i in [2 * x, 2 * x + 1]:
90         for j in [2 * y, 2 * y + 1]:
91             pix[i, j] = (carres[0][i - 2 * x][j - 2 * y],
92                          carres[1][i - 2 * x][j - 2 * y],
93                          carres[2][i - 2 * x][j - 2 * y])
94
95
96 def main():
97     bench_square(range(1,1000))
98
99
100 if __name__ == '__main__':
101     main()

```

8.4 Fichier serveur.py

```

1  #!/usr/bin/python2
2  #
3  # Name:          serveur.py
4  # Purpose:       serveur du TIPE sur les ondelettes
5  #
6  # Author:        Gaetan Bahl
7  #
8  # Created:       20/08/2013
9  # Copyright:     (c) Gaetan Bahl 2013
10 # Licence:       CC-BY-SA
11 #
12
13
14 import socket
15 import Image
16 import time
17 import struct
18
19
20
21 def recvImage(conn,msg):
22     title = msg.split('_')
23     sizex,sizy = int(title[1]),int(title[2])
24     image = Image.new("RGB", (sizex,sizy), "white" )
25     pix = image.load()
26     conn.send(b"ok")
27     msg, img = "", ""
28     unpacker = struct.Struct("BBB")
29     while msg.endswith("end") == False:
30         img += msg
31         msg = conn.recv(3)
32
33

```

```

34     print "pixels_recus"
35
36     for i in range(sizex):
37         for j in range(sizey):
38             pixel = unpacker.unpack(img[(i*sizey + j)*unpacker.size
39                                     : (i*sizey + j + 1)*unpacker.size])
40             pix[i, j] = (int(pixel[0]), int(pixel[1]), int(pixel
41                             [2]))
42
43     print "image_created"
44
45     image.save("srv" + title[3], 'JPEG', quality = 100)
46     print "image_enregistree"
47     conn.send(b'saved_image')
48
49 def fasthaar_srv(socket, msg, epsilon=0):
50     t = msg.split('_')
51     image = Image.open(t[1])
52     pix = image.load()
53
54     fichier = open("coef/" + t[1][: -4] + ".odl", 'wb')
55     sizex, sizey = image.size
56
57     print "compression_en_cours"
58     for x in range(sizex/2):
59         for y in range(sizey/2):
60
61             carres = []
62
63             for i in range(3):
64                 carres.append([[pix[2 * x, 2 * y][i], pix[2 * x, 2
65                                     * y + 1][i]],
66                               [pix[2 * x + 1, 2 * y][i], pix[2 * x + 1, 2 *
67                                     y + 1][i]]])
68
69             ondlhaut = [(carres[i][0][0] - carres[i][1][0]) / 2
70                         for i in range(3)]
71             ondlbas = [(carres[i][0][1] - carres[i][1][1]) / 2
72                        for i in range(3)]
73
74             for i in range(3):
75                 carres[i][0][0] = (carres[i][0][0] + carres[i
76                                     ][1][0]) / 2
77                 carres[i][0][1] = (carres[i][0][1] + carres[i
78                                     ][1][1]) / 2
79
80             ondlmix = [(carres[i][0][0] - carres[i][0][1]) / 2
81                        for i in range(3)]
82
83             for i in [ondlmix, ondlhaut, ondlbas]:
84                 fichier.write(struct.pack("bbb", i[0], i[1], i[2]))
85
86     fichier.close()

```

```

85     print 'compression_OK'
86     socket.send(b"OK")
87
88
89 def sendCoef(sock, msg):
90     banana = msg.split("_")
91     fichier = open("coef/srv" + banana[1][: -4] + ".odl", 'rb')
92     print "envoi_des_coefficients"
93     txt = fichier.readlines()
94     for i in txt:
95         sock.send(i)
96     print "coefficients_envoyes"
97     sock.send(b'end')
98     ms = sock.recv(2)
99     print ms.decode()
100    fichier.close()
101
102
103
104 def main():
105
106     hote = ''
107     port = 13337
108
109     connexion = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
110     connexion.bind((hote, port))
111     while 1:
112         connexion.listen(5)
113
114         msg_recu = b""
115         print "attente_d'un_client"
116         connexion_client, infos_connexion = connexion.accept()
117
118         print "client_connected"
119
120         while msg_recu != b"stop":
121             time.sleep(0.1)
122             msg = msg_recu.decode()
123
124             if msg.startswith('sending'):
125                 print "image_en_cours_de_reception"
126                 recvImage(connexion_client, msg)
127             if msg.startswith('compress'):
128                 fasthaar_srv(connexion_client, msg)
129             if msg.startswith('coef'):
130                 sendCoef(connexion_client, msg)
131
132
133             msg_recu = connexion_client.recv(1024)
134
135
136
137
138     print("Exiting")
139     connexion_client.close()
140     connexion.close()
141

```

```

142 if __name__ == "__main__":
143     main()

```

8.5 Fichier client.py

```

1  #!/usr/bin/python2
2  #
3  # Name:      client.py
4  # Purpose:   client du TIPE sur les ondelettes
5  #
6  # Author:    Gaetan Bahl
7  #
8  # Created:   20/08/2013
9  # Copyright: (c) Gaetan Bahl 2013
10 # Licence:   CC-BY-SA
11 #
12
13 import socket
14 import Image
15 import time
16 import struct
17 import sys
18
19 def sendImage(socket, link):
20     image = Image.open(link)
21     pix = image.load()
22     x,y = image.size
23     string = "sending_" + str(x) + "_" + str(y) + "_" + link
24     socket.send(string.encode('ascii'))
25     reply = socket.recv(32)
26     if reply == b"ok":
27         t = ""
28         for i in range(x):
29
30             for j in range(y):
31                 r,g,b = pix[i,j]
32                 packer = struct.Struct("BBB")
33                 pixel = packer.pack(r, g, b)
34                 socket.send(pixel)
35
36
37     socket.send(b"end")
38     reply = socket.recv(11)
39
40     print "server_says:_:" + reply
41
42 def askcompress(connexion, nom):
43
44     mess = "compress_" + nom
45     connexion.send(mess.encode('ascii'))
46     reply = connexion.recv(2)
47     print "server_says:_:" + reply
48     return reply
49
50 def storeImage(link):

```

```

51 image = Image.open(link)
52 pix = image.load()
53 x,y = image.size
54
55 im = Image.new("RGB", (x / 2, y / 2), "white")
56 pi = im.load()
57
58 for i in range(x / 2):
59     for j in range(y / 2):
60         r = (((pix[2 * i, 2 * j][0] + pix[2 * i + 1, 2 * j][0])
61              / 2) + ((pix[2 * i, 2 * j + 1][0] + pix[2 * i + 1,
62              2 * j + 1][0]) / 2)) / 2
63         g = (((pix[2 * i, 2 * j][1] + pix[2 * i + 1, 2 * j][1])
64              / 2) + ((pix[2 * i, 2 * j + 1][1] + pix[2 * i + 1,
65              2 * j + 1][1]) / 2)) / 2
66         b = (((pix[2 * i, 2 * j][2] + pix[2 * i + 1, 2 * j][2])
67              / 2) + ((pix[2 * i, 2 * j + 1][2] + pix[2 * i + 1,
68              2 * j + 1][2]) / 2)) / 2
69         pi[i, j] = (r,g,b)
70
71 im.save("stor" + link, 'JPEG', quality = 100)
72
73 def askcoeff(conn, nom):
74     unpacker = struct.Struct("bbbbbbbb")
75
76     print 'demande_en_cours'
77     mess = "coef_" + nom
78     conn.send(mess.encode("ascii"))
79
80     print "reception_coefs"
81     coefs = ''
82     msg = ''
83     while msg.endswith(b'end') == False:
84         coefs += msg
85         msg = conn.recv(9)
86     print str(len(coefs)) + "_octets_recus"
87     print "reconstitution_image"
88     image = Image.open("stor" + nom)
89     pix = image.load()
90     dim = image.size
91     im = Image.new("RGB", (dim[0] * 2, dim[1] * 2), "white")
92     px = im.load()
93     for i in range(dim[0]):
94         for j in range(dim[1]):
95             tab = unpacker.unpack(coefs[(i*dim[1] + j)* unpacker.
96                                     size : (i*dim[1] + j + 1)* unpacker.size])
97
98             px[2*i,2*j] = ( pix[i,j][0] + tab[0] + tab[3], pix[i,j]
99                             [1] + tab[1] + tab[4], pix[i,j][2] + tab[2] + tab
100                             [5])
101             px[2*i+1,2*j] = ( pix[i,j][0] + tab[0] - tab[3], pix[i,
102                             j][1] + tab[1] - tab[4], pix[i,j][2] + tab[2] - tab
103                             [5])

```

```

96         px[2*i,2*j + 1] = ( pix[i,j][0] - tab[0] + tab[6], pix[
          i,j][1] - tab[1] + tab[7], pix[i,j][2] - tab[2] +
          tab[8])
97         px[2*i+1,2*j+1] = ( pix[i,j][0] - tab[0] - tab[6], pix[
          i,j][1] - tab[1] - tab[7], pix[i,j][2] - tab[2] -
          tab[8])
98
99
100     print "enregistrement"
101     im.save('2' + nom, 'JPEG', quality = 100)
102     conn.send(b'ok')
103
104
105 def main(arg):
106
107     connexion = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
108     if 'r' in arg[1]:
109         try :
110             connexion.connect((argv[3], 13337))
111         except:
112             connexion.connect(('srv.ordiclic.eu', 13337))
113     else:
114         connexion.connect(('localhost', 13337))
115
116     if 's' in arg[1]:
117         sendImage(connexion, arg[2])
118         askcompress(connexion, 'srv' + arg[2])
119         storeImage(arg[2])
120     if 'd' in arg[1]:
121         askcoeff(connexion, arg[2])
122
123     connexion.send(b"stop")
124     connexion.close()
125
126
127 if __name__ == "__main__":
128     main(sys.argv)

```

9 Liens

- Tous les fichiers .tex, .py de ce document : <https://github.com/timosis/TIPE2013-2014>
- L'interpréteur Python : <http://www.python.org/>
- La librairie PIL pour Python : <http://www.pythonware.com/products/pil/>
- La licence GNU GPL : <http://www.gnu.org/licenses/gpl-3.0.en.html>
- La licence Creative Commons BY-SA (sous laquelle notre code est placé) : <http://creativecommons.org/licenses/by-sa/3.0/>
- Source http://www.cmi.univ-mrs.fr/~melot/Master2/TPsignal_PS.html