TIPE: Introduction à la théorie des ondelettes

Xavier Friederich et Gaétan Bahl

23 mai 2014

Table des matières

1 L'analyse de Fourier, outil certes efficace mais insuffisant **4**
- 1.1 Le cas des signaux périodiques : l'utilisation des séries de Fourier . . . 4
- 1.2 L'utilisation de la transformée de Fourier . . . 5
  - 1.2.1 Définition de la transformation de Fourier . . . 5
  - 1.2.2 Propriétés de la transformation de Fourier . . . 5
  - 1.2.3 Transformée inverse . . . 6
  - 1.2.4 Ce qu'apporte la transformée de Fourier d'un signal . . . 6
  - 1.2.5 Exemples de transformée de Fourier . . . 6
  - 1.2.6 Application de la transformation de Fourier . . . 7
- 1.3 Transformée de Fourier Discrète ou TFD et limites . . . 8
  - 1.3.1 Définition . . . 8
  - 1.3.2 Des applications de la TFD . . . 8
  - 1.3.3 Des limites à la Transformée de Fourier Discrète . . . 9

2 Premières définitions **10**
- 2.1 Définition 1 : Ondelette . . . 10
- 2.2 Exemples d'ondelettes mères : . . . 10
  - 2.2.1 Ondelette de Haar : . . . 10
  - 2.2.2 Ondelette "chapeau mexicain" : . . . 10
  - 2.2.3 Ondelette de Morlet : . . . 11

3 Transformation en ondelettes **12**
- 3.1 Transformation en ondelette continue . . . 12
- 3.2 Transformation en ondelettes discrète. . . . 12
- 3.3 Comparaison avec la transformation de Fourier . . . 14

4 La théorie de l'analyse multirésolution, un outil pour la construction de bases d'ondelettes **15**
- 4.1 Définition d'une analyse multirésolution. . . . 15
- 4.2 Intéret d'une analyse multirésolution . . . 15

5 Algorithme de décomposition en ondelettes de Stéphane Mallat (1989) **15**
- 5.1 Principe . . . 15
- 5.2 Démonstration dans un cas simple : le cas des ondelettes de Haar. . . . 15
  - 5.2.1 Introduction et définition des notations . . . 16
  - 5.2.2 Propriété mathématique : Orthogonalité dans les espaces euclidiens . . . 17
  - 5.2.3 Utilisation de la propriété : décomposition orthogonale en somme directe. . . . 17
  - 5.2.4 Étape principale de l'algorithme : passage à la résolution inférieure, détermination des coefficients à la résolution inférieure. . . . 18
- 5.3 Schématisation de l'algorithme de Mallat (compression d'un signal Ψ_p par des ondelettes) . . . 20
- 5.4 Représentation matricielle de l'algorithme utilisant les ondelettes de Haar . . . 21
  - 5.4.1 Exemple . . . 21

6 Application à la compression des données 22

7 Utilisation pratique de la transformation par ondelettes discrète 22
7.1 Le choix des outils . . . 23
7.1.1 Le langage : Python . . . 23
7.1.2 La librairie de traitement d'image : PIL . . . 23
7.1.3 La librairie d'interface graphique : Tkinter . . . 23
7.1.4 Le module time . . . 23
7.1.5 Le module struct . . . 23
7.1.6 Le module socket . . . 24
7.2 L'algorithme matriciel . . . 24
7.3 Exemples d'images traitées avec notre algorithme . . . 25
7.4 L'application graphique . . . 27
7.5 L'algorithme fasthaar . . . 29
7.5.1 Fonctionnement . . . 29
7.5.2 Coût et Performance . . . 32
7.6 Transferts et échanges d'images par le réseau . . . 33
7.6.1 But du programme . . . 33
7.6.2 Processus d'archivage . . . 33
7.6.3 Processus de restauration . . . 34

8 Description du code 35
8.1 Le fichier ondelettes.py . . . 35
8.1.1 La classe Matrice . . . 35
8.1.2 La classe MatriceImage . . . 35
8.2 Le fichier ondelettesGUI.py . . . 37
8.2.1 La classe Appli . . . 37
8.2.2 La classe DialogScale . . . 37
8.2.3 La fonction main . . . 38
8.3 Le fichier bench.py . . . 38
8.3.1 La fonction image_gen . . . 38
8.3.2 La fonction randomize . . . 38
8.3.3 La fonction bench_square . . . 38
8.3.4 La fonction fasthaar . . . 38
8.3.5 La fonction main . . . 38
8.4 Le fichier serveur.py . . . 38
8.4.1 La fonction main . . . 38
8.4.2 La fonction recvImage . . . 38
8.4.3 La fonction fasthaar_srv . . . 39
8.4.4 La fonction sendCoef . . . 39
8.5 Le fichier client.py . . . 39
8.5.1 La fonction main . . . 39
8.5.2 La fonction sendImage . . . 39
8.5.3 La fonction askcompress . . . 39
8.5.4 La fonction storeImage . . . 39
8.5.5 La fonction askcoeff . . . 40

9 Conclusion 40

10 Bibliographie, Liens et Remerciements 40

11 Annexes **41**
11.1 Fichier ondelettes.py . . . 41
11.2 Fichier ondelettesGUI.py . . . 46
11.3 Fichier bench.py . . . 48
11.4 Fichier serveur.py . . . 50
11.5 Fichier client.py . . . 52

Table des figures

1 Illustration du théorème de Fourier pour les signaux périodiques . . . 5
2 Spectre de Fourier en fréquence d'un signal périodique . . . 6
3 Allure d'une gaussienne . . . 7
4 Spectre d'amplitude de la fonction Π . . . 7
5 Ondelette de Haar . . . 11
6 Ondelette "chapeau mexicain" . . . 11
7 Ondelette de Morlet . . . 11
8 Dilatation d'ondelette . . . 12
9 Translation d'ondelette . . . 13
10 Le graphe de la fonction $\mathcal{W}(a;b)(\Pi)$ . . . 14
11 Comparaison des méthodes de compression . . . 14
12 L'algorithme matriciel . . . 24
13 La transformation appliquée aux matrices . . . 24
14 L'image de départ . . . 25
15 Détail de l'image de départ . . . 25
16 Détail de l'image compressée . . . 26
17 Détail de l'image compressée à 100% . . . 26
18 Fenêtre principale de l'application . . . 27
19 Menus de l'application . . . 28
20 Le sélecteur de seuil de compression . . . 28
21 Fonctionnement de l'algorithme . . . 29
22 Tableau créé par l'algorithme . . . 29
23 Tableau après l'étape 1 . . . 30
24 Tableau après l'étape 2 . . . 30
25 Tableau après compression . . . 30
26 Tableau après synthèse sur les colonnes . . . 30
27 Tableau après synthèse sur les lignes . . . 30
28 Tableau après synthèse sur les lignes . . . 31
29 Pixels avant et après traitement . . . 31
30 Coût temporel de l'algorithme fasthaar en fonction de la taille de l'image . . . 32
31 Processus d'archivage . . . 33
32 Processus de restauration . . . 34

1 L'analyse de Fourier, outil certes efficace mais insuffisant

Les séries de Fourier (pour les signaux périodiques) et la transformée de Fourier (pour un signal quelconque) ont longtemps été les outils essentiels de l'analyse harmonique.
Le but de cette première partie est de présenter brièvement l'analyse de Fourier et de montrer ses limites.

1.1 Le cas des signaux périodiques : l'utilisation des séries de Fourier

Considérons f fonction 2π-périodique de classe $\mathcal{C}^1$ par morceaux.
Notons $S_p(f)$ la p-ième somme partielle de Fourier de f, ou p-ième somme partielle de la série de Fourier de f.
On a :

$$S_p(f) = \sum_{n=-p}^{p} c_n(f) e^{int}$$

en notation exponentielle ou bien

$$S_p(f) = \frac{a_0}{2} + \sum_{n=1}^{p} (a_n(f)\cos(nt) + b_n(f)\sin(nt))$$

en notation trigonométrique ;
les a_n et b_n étant donnés par les relations

$$a_n = \frac{1}{\pi}\int_0^{2\pi} f(t)\cos(nt)\mathrm{d}t$$

$$b_n = \frac{1}{\pi}\int_0^{2\pi} f(t)\sin(nt)\mathrm{d}t$$

Le cours de mathématiques (plus particulièrement le théorème de Dirichlet) nous donne alors le résultat bien connu suivant, avec les hypothèses sur f données plus haut :

f est somme de sa série de Fourier, ce qui se réécrit de la façon suivante :

$$\forall t \in \mathbb{R}, f(t) = \frac{a_0}{2} + \sum_{-\infty}^{+\infty} (a_n(f)\cos(nt) + b_n(f)\sin(nt))$$

On peut ainsi très facilement décomposer un signal périodique en une somme infinie de sinusoïdes.

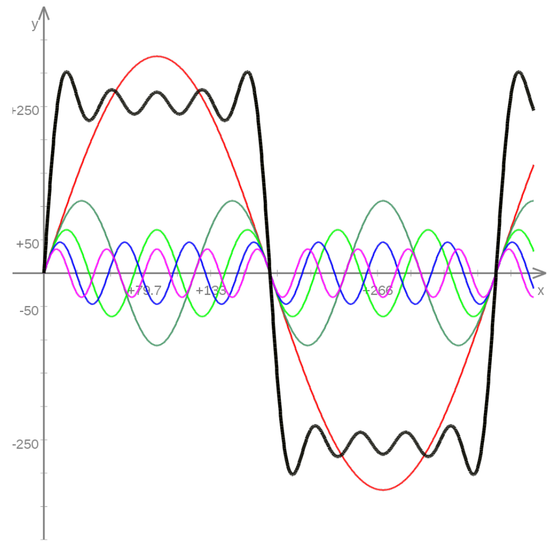

FIGURE 1 – Illustration du théorème de Fourier pour les signaux périodiques

Le signal "carré" en noir est la somme de toutes les sinusoïdes représentées, la sinusoïde rouge d'amplitude majeure étant appelée *fondamentale* et les autres sinusoïdes étant les *harmoniques*.

Bien évidemment, la plupart des signaux que l'on peut rencontrer sont non-périodiques et il est impossible d'obtenir une décomposition analogue à celle décrite ci-dessus. Dans le cas général et dans la limite de la possibilité de le faire, on effectue une transformation de Fourier.

1.2 L'utilisation de la transformée de Fourier

Il est nécessaire dans le cas plus général de fonctions/signaux non nécessairement périodiques de passer d'une écriture discrète en uneécriture en somme continue.
Le cadre le plus naturel pour définir les transformations de Fourier est celui des fonctions f intégrables [1].
On note alors traditionnellement $\mathcal{L}^1(\mathbb{R})$ l'ensemble des fonctions intégrables sur $\mathbb{R}$ et $\mathcal{L}^2(\mathbb{R})$ l'ensemble des fonctions de carré intégrable sur $\mathbb{R}$.

1.2.1 Définition de la transformation de Fourier

On appelle transformation de Fourier l'application notée $\mathcal{F}$ qui, à toute fonction f de $\mathcal{L}^1(\mathbb{R})$, associe la fonction $\hat{f}$ telle que $\forall \omega \in \mathbb{R}, \hat{f}(\omega) = \dfrac{1}{\sqrt{2\pi}} \displaystyle\int_{-\infty}^{\infty} f(t) e^{-i\omega t} \, \mathrm{d}t$
$\hat{f}$ est appelée transformée de Fourier de f.

Notons toutefois que l'on peut donner plusieurs versions de définitions : nous avons ici choisi la définition plus "physicienne", car on y voit directement les paramètres de temps (t) en s et de pulsation (ω) en $rad.s^{-1}$.

Notons aussi qu'il est possible de définir la transformée de Fourier pour des fonctions qui ne sont pas forcément dans $\mathcal{L}^1(\mathbb{R})$.

1.2.2 Propriétés de la transformation de Fourier

— $\mathcal{F}$ est clairement linéaire.

1. continues par morceaux et telles que $\exists M \in \mathbb{R}, \forall I \subset \mathbb{R}, \left| \int_I f(x) \right| \leq M$

— On peut montrer que $\mathcal{F}$ conserve la parité.

— Propriété de translation :
Soit $a \in \mathbb{R}$ et $f \in \mathcal{L}^1(\mathbb{R})$ de la variable t. En effectuant le changement de variable $u = t - a$, on obtient la transformée de Fourier de la fonction d'expression $f(t-a)$. En effet :

$$\mathcal{F}[f(t-a)] = \int_{-\infty}^{\infty} f(t-a)e^{-i\omega t}\,\mathrm{d}t = e^{-i\omega a}\int_{-\infty}^{\infty} f(u)e^{-i\omega u}\,\mathrm{d}u = e^{-i\omega a}.\hat{f}(\omega)$$

1.2.3 Transformée inverse

On utilise les mêmes notations que précédemment. Si $\hat{f}$ est elle-même une fonction intégrable, la formule dite de transformation de Fourier inverse, opération notée $\mathcal{F}^{-1}$, et appliquée à $\hat{f}$, permet (sous conditions appropriéees) de retrouver f :

$$f(t) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} \hat{f}(\omega)e^{i\omega t}\,\mathrm{d}\omega$$

Cette formule peut se démontrer facilement à partir de la formule sommatoire de Poisson.

1.2.4 Ce qu'apporte la transformée de Fourier d'un signal

Dans le cas général, la transformation de Fourier d'une fonction produit comme transformée une fonction $\hat{f}$ à valeurs complexes. Ainsi, on peut obtenir deux informations de cette transformée :

Le spectre d'amplitude : il s'agit du tracé du module de $\hat{f}(\omega)$ en fonction de la pulsation ω.

Le spectre de phase : il s'agit du tracé de l'argument de $\hat{f}(\omega)$ en fonction de la pulsation ω.

Notons que l'on rencontre très souvent en traitement du signal les spectres en fréquence ; on passe de la pulsation ω à la fréquence par la relation de proportionnalité suivante : $f = \dfrac{1}{2\pi}\omega$.

On remarquera, comme le montre la figure ci-dessous, que le spectre d'amplitude d'un signal périodique est formé de traits verticaux.

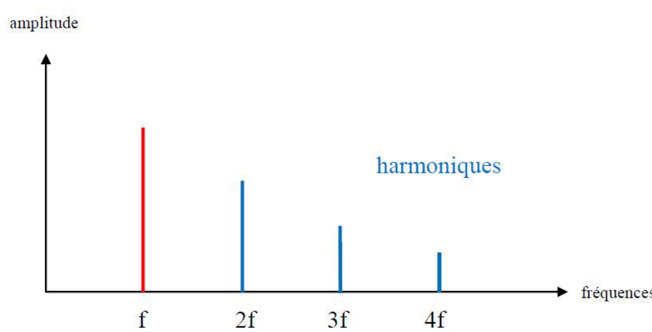

FIGURE 2 – Spectre de Fourier en fréquence d'un signal périodique

1.2.5 Exemples de transformée de Fourier

Facilement, on peut montrer que la transformée de Fourier d'une gaussienne [2] est une gaussienne.

Si on note Π la fonction porte définie par $\forall t \in \left[-\frac{1}{2}, \frac{1}{2}\right], \Pi(t) = 1$ et $\forall t \in \mathbb{R}\setminus\left[-\frac{1}{2}, \frac{1}{2}\right], \Pi(t) = 0$, on obtient directement par intégration de l'exponentielle complexe et en tenant compte de la relation $\sin x = \dfrac{e^{ix} - e^{-ix}}{2i}$,

2. une fonction en $e^{\frac{-x^2}{2}}$.

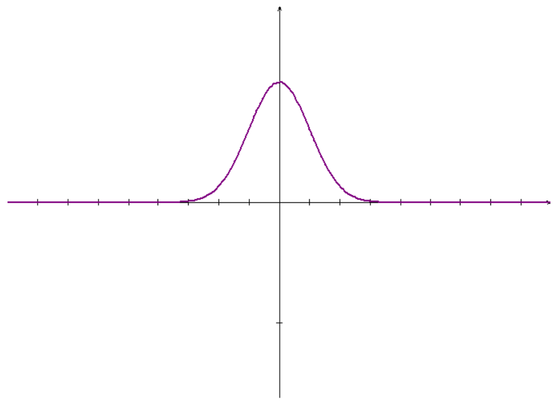

FIGURE 3 – Allure d'une gaussienne

$\mathcal{F}(\Pi)(\omega) = \text{sinc}(\frac{\omega}{2})$. [3]
La fonction étant réelle, son spectre de phase correspond à la fonction nulle et son spectre d'amplitude est le suivant :

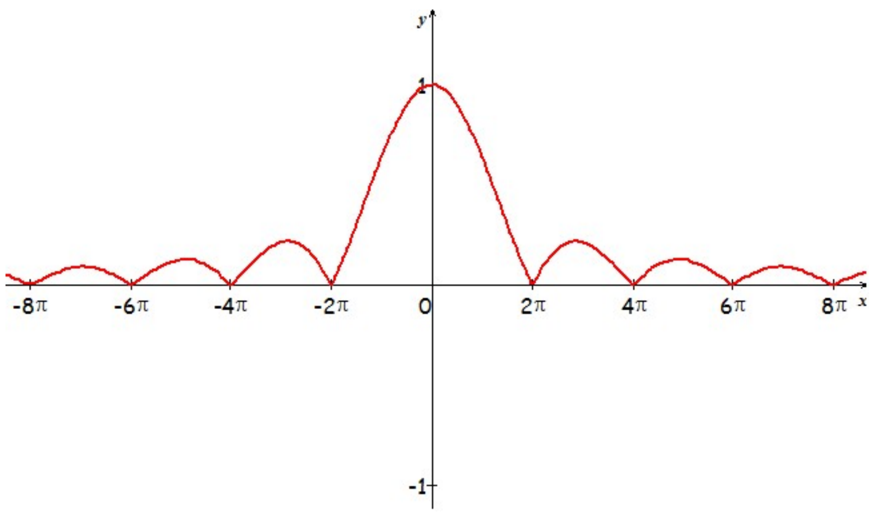

FIGURE 4 – Spectre d'amplitude de la fonction Π

1.2.6 Application de la transformation de Fourier

En physique, la transformation de Fourier permet de déterminer le spectre d'un signal.

En traitement d'images, on effectue des transformations de Fourier à deux dimensions : si f est une fonction de $\mathbb{R}^2$, sa transformée de Fourier est définie par :

$$\hat{f}(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y).e^{-i(ux+vy)}\,\mathrm{d}x\,\mathrm{d}y$$

On comprendra que la très grande majorité des signaux sont numériques et que les définitions mathématiques données jusqu'à présent ne sont pas adaptées au domaine du discret.

3. la fonction sinc (sinus cardinal) est au premier sens mathématique la fonction définie sur $\mathbb{R}^*$ par $\text{sinc}(x) = \frac{\sin x}{x}$.

1.3 Transformée de Fourier Discrète ou TFD et limites

Bien évidemment, la transformée de Fourier telle qu'elle est utilisée dans un ordinateur (transformée de Fourier discrète (TFD) possède une définition numérique différente de la définition mathématique donnée plus haut.

1.3.1 Définition

Nous nous placerons dans le cas complexe (le cas réel en découle) sur un intervalle de temps fini correspondant à N échantillons. Quand N tend vers l'infini, on peut penser que l'on s'approche du cas continu mais il faut garder à l'esprit que la TFD suppose que le signal est périodique de période N.

Nous proposons la définition de la TFD d'un point de vue de l'algèbre linéaire, qui semble plus schématique : On définit ainsi la TFD comme un endomorphisme de $\mathbb{C}^N$ ayant pour matrice dans la base canonique de $\mathbb{C}^N$ la matrice S de terme général $s_{j,k} = \dfrac{1}{N}e^{-2i\pi\frac{jk}{N}}$, où $j, k \in \{0, 1, ..., N-1\}$.

La TFD s'applique ainsi à des suites de longueur N (ou de période N) et on peut d'ailleurs remarquer que la TFD est périodique de période N.

On obtient ainsi, en l'appliquant à un vecteur $f = (f_0, ..., f_{N-1})$ de $\mathbb{C}^N$ de matrice $F \in \mathcal{M}_{N,1}(\mathbb{C})$, le vecteur $g = (g_0, ..., g_{N-1})$ de matrice $G = S\times F$.
On a clairement de la définition et du produit matriciel :

$$\forall j \in \{0, 1, ..., N-1\}, g_j = \sum_{k=0}^{N-1} \frac{1}{N}e^{-2i\pi\frac{jk}{N}} f_k$$

En notant W l'inverse de la racine N-ième de l'unité [4], il vient bien sûr :

$$\forall j \in \{0, 1, ..., N-1\}, g_j = \sum_{k=0}^{N-1} W^{kj} f_k \tag{1}$$

1.3.2 Des applications de la TFD

La TFD a plusieurs applications, parmi lesquelles :

1. L'analyse spectrale des signaux.
   Il est intéressant pour un électronicien de mesurer par exemple la largeur de la bande de fréquence occupée par la transmission d'un signal, ceci grâce à une analyse spectrale.

2. La compression de données.
   On applique sur les signaux la TFD pour réduire leur complexité. La suite des coefficients obtenus (en appliquant la formule (1) est alors quantifiée avec des pas de quantification plus élevés pour les hautes fréquences, considérées comme négligeables pour la perception humaine. Le gain en compression vient de la réduction de précision de ces coefficients (voire leur suppression totale) : cela nécessite de ce fait moins de bits pour le codage.

4. $W = e^{-\dfrac{2i\pi}{N}}$

3. La multiplication des grands nombres.
   Certains des algorithmes les plus rapides (type FFT) pour la multiplication de grands nombres entiers sont fondés sur la TFD.

Néanmoins, toutes ces applications nécessitent l'existence d'un algorithme rapide de calcul de la TFD et de son inverse. Les multiplications dans les cas où N est petit sont "triviales", mais quand N devient grand il est en effet indispensable d'utiliser un tel algorithme permettant de diminuer le nombre de multiplications.

1.3.3 Des limites à la Transformée de Fourier Discrète

La TFD présente des limites considérables.
On ne peut pas analyser un morceau de musique avec une TFD simple. En effet, on perdrait l'information temporelle. Prenons par exemple deux signaux semblables :

1. un signal composé d'une sinusoïde à 100Hz pendant une seconde puis d'une sinusoïde à 200Hz pendant une seconde
2. un second composé d'une sinusoïde à 200Hz pendant une seconde puis d'une sinusoïde à 100Hz pendant la seconde suivante.

Leurs transformées de Fourier respectives seront identiques, ce qui est clair sur l'expression (1) (commutativité de l'addition).
Par conséquent, la TFD n'est applicable que sur des signaux dont l'on sait que l'information fréquentielle est la même partout.

En outre, l'algorithme FFT nécessite que N soit une puissance de 2 à cause de l'architecture récursive du programme. De plus, les algorithmes type FFT que l'on programme ne sont pas toujours efficaces au niveau de la mémoire et de la rapidité car on doit tenir compte des matrices et des nombres complexes que le logiciel de programmation ne connaît *a priori* pas.

On verra dans ce qui suit une transformation des fonctions/signaux plus performante, la transformation par les ondelettes qui est capable de détecter les portions du signal qui varient plus rapidement que d'autres.

2 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.

2.1 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 $\hat{\psi}(\omega) = \displaystyle\int_{-\infty}^{+\infty} \psi(t) \cdot e^{-2i\pi\omega t} dt$

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 :

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

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

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.

2.2 Exemples d'ondelettes mères :

2.2.1 Ondelette de Haar :

Il s'agit de la fonction $\begin{array}{rccl} \mathcal{H} : & [0,1] & \longrightarrow & \{-1;1\} \\ & x & \longmapsto & \begin{cases} 1 & \text{si } x \in [0; \frac{1}{2}] \\ -1 & \text{si } x \in]\frac{1}{2}; 1] \end{cases} \end{array}$

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

2.2.2 Ondelette "chapeau mexicain" :

On peut définir cette fonction par $\begin{array}{rccl} \psi : & \mathbb{R} & \longrightarrow & \mathbb{R} \\ & t & \longmapsto & \frac{2}{\sqrt{3}} \pi^{-\frac{1}{4}} (1 - t^2) e^{-\frac{t^2}{2}} \end{array}$

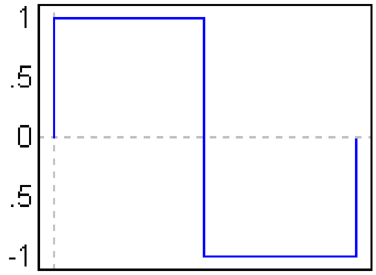

Figure 5 – Ondelette de Haar

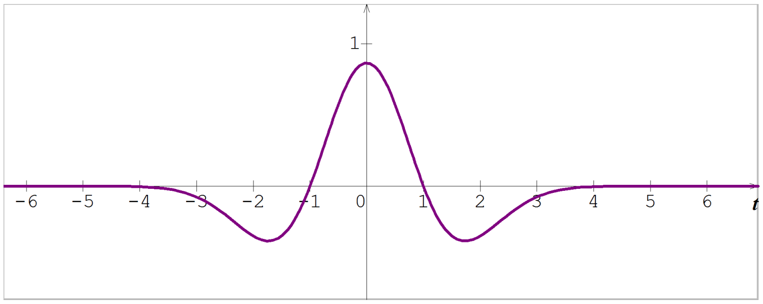

Figure 6 – Ondelette ”chapeau mexicain”

2.2.3 Ondelette de Morlet :

On peut définir cette fonction par $\begin{array}{rccl} \psi: & \mathbb{R} & \longrightarrow & \mathbb{R} \\ & t & \longmapsto & \cos(5t)e^{-\frac{t^2}{2}} \end{array}$

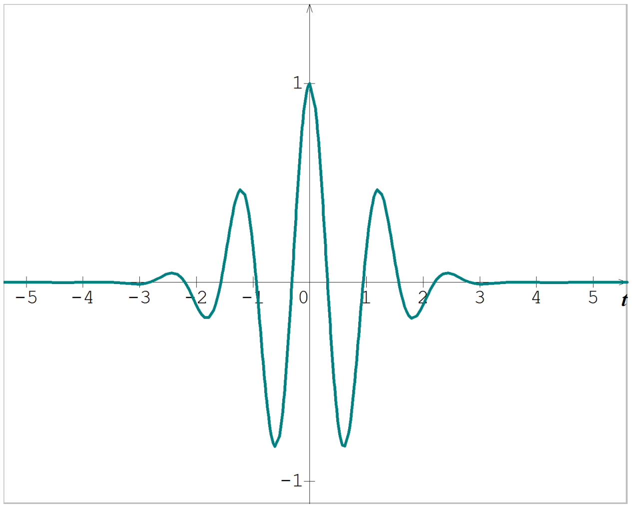

Figure 7 – Ondelette de Morlet

3 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 base qui varient en fréquence et en espace.

3.1 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.

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

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 $a \neq 1$.

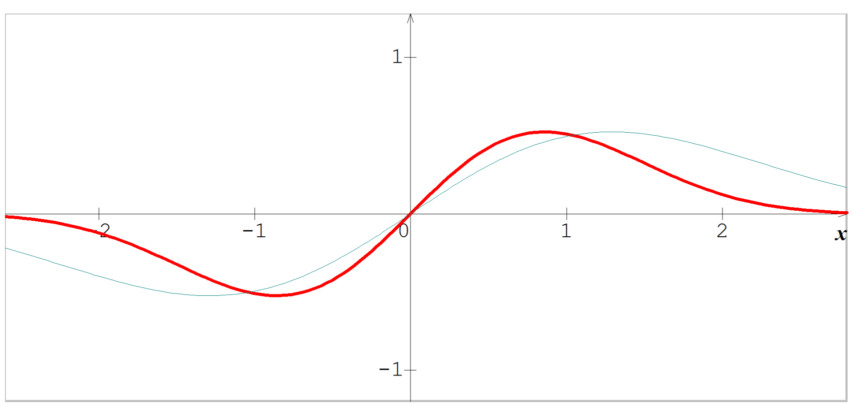

FIGURE 8 – Dilatation d'ondelette

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

3.2 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 :

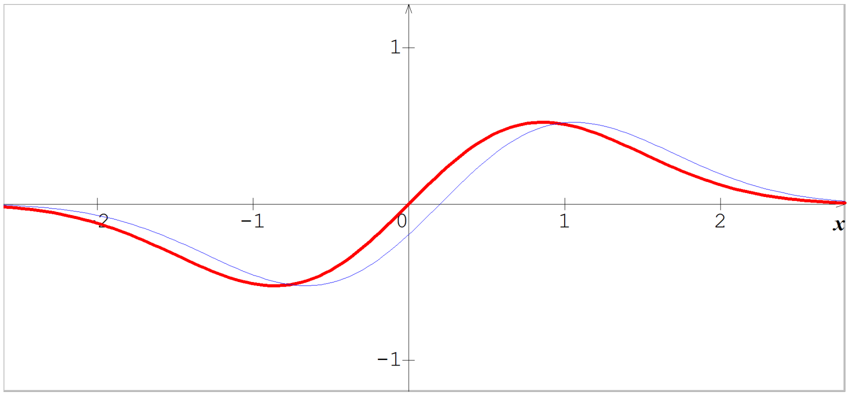

Figure 9 – Translation d'ondelette

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

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 $t_0 = k \cdot \Delta t$:

$f_{t_0,\Delta t}(t) = 2^{\frac{p}{2}} f(2^p \cdot x - k)$, 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

$f = \sum_{(k,p)\in\mathbb{Z}^2} f_{k,p}\psi_{k,p}$ où les $f_{k,p} = \langle f|\psi_{k,p}\rangle$ 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.

Le graphique suivant (Fig.10), représentant l'application $\mathcal{W}(a;b)$ appliquée en la fonction (ou signal en 1D) $f = \Pi$ (définie dans la première section de ce dossier) et utilisant l'ondelette mère de Morlet (définie plus haut), illustre la richesse de la transformation en ondelettes continue.

A partir d'une ondelette mère, on peut créer une pluralité d'ondelettes "filles" qui vont fournir, par rapport à la transformation classique de Fourier, une plus grande précision dans le traitement des signaux de fréquences constamment changeantes.

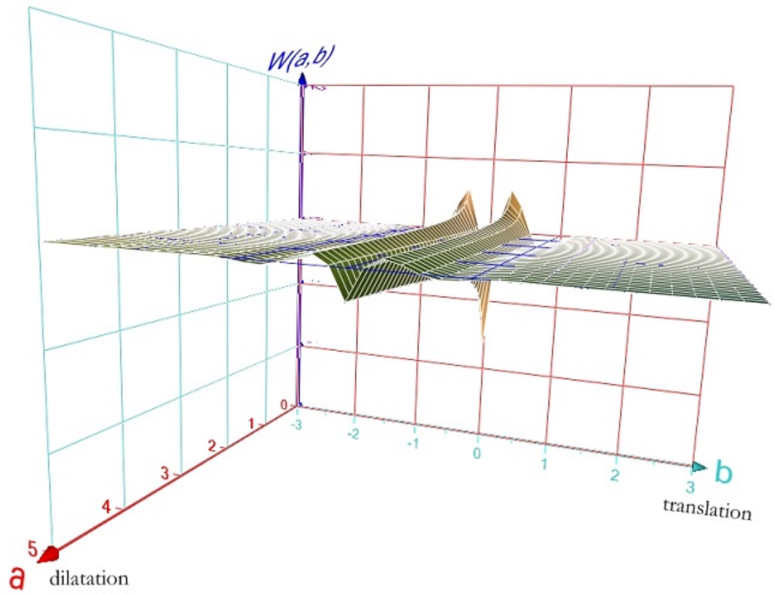

FIGURE 10 – Le graphe de la fonction $\mathcal{W}(a;b)(\Pi)$

3.3 Comparaison avec la transformation de Fourier

Un des avantages de la transformation par les ondelettes (en comparaison avec la transformation de Fourier), c'est que le fait de modifier ou de supprimer un des coefficients de la transformée d'un signal ne va en rien dégrader le signal.
En outre, les algorithmes de transformation en ondelettes 2D s'appliquent à la totalité de l'image et non pas à des blocs de pixels comme par exemple les algorithmes type FFT, ce qui permet d'éviter les carrés uniformes lorsque le taux de compression est relativement élevé.
Enfin, l'utilisation d'une ondelette réversible permet une compression sans perte de données.

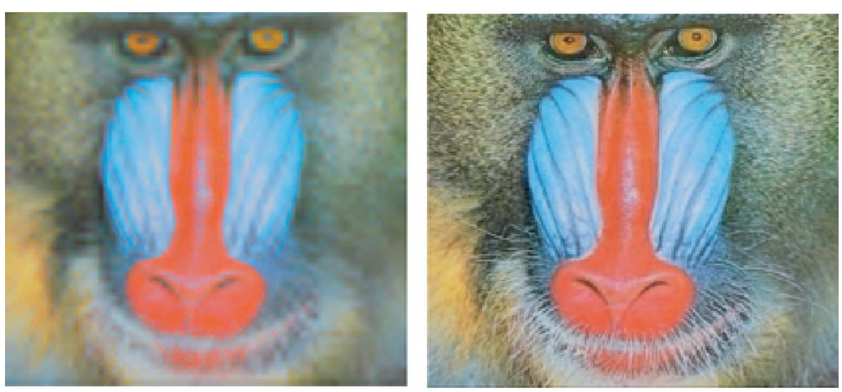

FIGURE 11 – Comparaison des méthodes de compression

La figure 11 montre les résultats de deux compressions de la même image de départ, l'une utilisant la transformation de Fourier (à gauche), l'autre utilisant la transformation en ondelettes (à droite).

Enfin, le coût d'un algorithme utilisant les ondelettes, c'est-à-dire le nombre d'opérations à effectuer, est en $O(N)$, ce qui est mieux que le coût des meilleurs algorithmes type FFT en $O(N \log N)$.

4 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.

4.1 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)
— $\forall k \in \mathbb{Z}, V_{k+1} \subset V_k$
— $\forall k \in \mathbb{Z}, f \in V_k \Leftrightarrow f(\frac{\cdot}{2}) \in V_{k+1}$
— $\lim\limits_{j\to\infty} V_k = \bigcap\limits_{k\in\mathbb{Z}} V_k = \emptyset$
— $\lim\limits_{j\to-\infty} V_k = \overline{\bigcap\limits_{k\in\mathbb{Z}} V_k} = \mathcal{L}^2(\mathbb{R})$ 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

4.2 Intéret 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 $\mathcal{L}^2(\mathbb{R}) = \overline{\bigoplus_{k\in\mathbb{Z}} W_k}$.

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 $\mathcal{L}^2(\mathbb{R}) = \overline{\bigoplus_{k\in\mathbb{Z}} W_k}$.

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

5.1 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.

5.2 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.

5.2.1 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 $\mathbb{K}_0 \subset \mathbb{K}_1 \subset .. \subset \mathbb{K}_p \subset \mathbb{K}_{p+1} \subset ...$ ce qui montre $\bigcup\limits_{p\in\mathbb{N}} \mathbb{K}_p$ 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

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

(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 :

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

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

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

5.2.2 Propriété mathématique : Orthogonalité dans les espaces euclidiens

Énoncé Soient E un espace euclidien de dimension $n \geq 1$, $\langle . | . \rangle$ 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 $x \in E$.
$\exists (\lambda_1, ..., \lambda_p) \in \mathbb{R}^p, x = \sum_{k=1}^{p} \lambda_k e_k + (x - \sum_{k=1}^{p} \lambda_k e_k)$
Or le premier vecteur de la somme est dans F. Donc le second entre parenthèses appartient à $F^{\perp}$ si et seulement si
$\forall k \in [|1, p|], \langle e_k | x - \sum_{k=1}^{p} \lambda_k e_k \rangle = 0$, 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.

5.2.3 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 :
$\mathbb{K}_{p+1} = \mathbb{S}_p \oplus \mathbb{S}_{p-1} \oplus \mathbb{S}_{p-2} \oplus ... \oplus \mathbb{S}_0 \oplus \mathbb{K}_0$, soit encore $\mathbb{K}_p = \mathbb{K}_0 \oplus_{i=0}^{p-1} \mathbb{S}_i$

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

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

Facilement, on montre que $(h_{p,1}, h_{p,0}, ..., h_{p,2^p-1}, H_{p,0}, H_{p,1}, ..., H_{p,2^p-1})$ forme une base de $\mathbb{K}_{p+1}$.
De plus, on a :
— Si $k \neq k'$

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

— Si $k = k'$

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

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

Alors le système $(H_{p,0}, H_{p,1}, ..., H_{p,2^p-1},)$ 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 :

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

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

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

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 :

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

5.2.4 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 $(h_{p,1}, h_{p,0}, ..., h_{p,2^p-1}, H_{p,0}, H_{p,1}, ..., H_{p,2^p-1})$ avec $\Psi_p = \Psi_{p-1}+d_{p-1}$ et les résultats précédents sur les produits scalaires amène à : $\boxed{\langle\Psi_p|h_{p-1,k}\rangle = \frac{\Psi_{p-1,k}}{2^{p-1}}}$ et $\boxed{\langle\Psi_p|H_{p-1,k}\rangle = \frac{d_{p-1,k}}{2^{p-1}}}$.

Démonstration On a

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

Et on a

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

Secondes égalités D'autre part, on peut montrer $\boxed{\langle \Psi_p | h_{p-1,k} \rangle = \frac{\Psi_{p,2k} + \Psi_{p,2k+1}}{2^p}}$ 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 :

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

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

—

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

—

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

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

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

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.

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

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

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

$$\begin{array}{ccccccccc} & & \text{Etape 1} & & \text{Etape 2} & & & & \text{Etape p} \\ \Psi_p & \rightarrow & \Psi_{p-1} & \rightarrow & \Psi_{p-2} & \rightarrow & ... & \rightarrow & \Psi_0 \\ & \searrow & d_{p-1} & \searrow & d_{p-2} & \searrow & d_{p-3} & \searrow & d_0 \end{array}$$

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.

5.4 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).

5.4.1 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

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

On obtient alors en première moitié verticale de la matrice le nouvel échantillon principal et en seconde moitié les coefficients représentants 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 $2^p * 2^p$ 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 quelque peu différente de la matrice M_p puisque certains coefficients sont devenus des 0.

6 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.

7 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).

7.1 Le choix des outils

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

7.1.1 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.

7.1.2 La librairie de traitement d'image : PIL

Pour que notre application puisse supporter plusieurs types de fichiers, nous avons eu recours à une librairie 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 librairie, bien entendu, puisqu'il s'agissait avant tout de concevoir notre propre algorithme de compression.

7.1.3 La librairie d'interface graphique : Tkinter

Pour la partie «interface graphique utilisateur» (ou GUI), nous avons utilisé la librairie 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 librairies 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.

7.1.4 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.

7.1.5 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.

7.1.6 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.

7.2 L'algorithme matriciel

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 12 est un schéma-bloc montrant l'algorithme. La partie «Transformation DWT» est explicitée par la figure 13

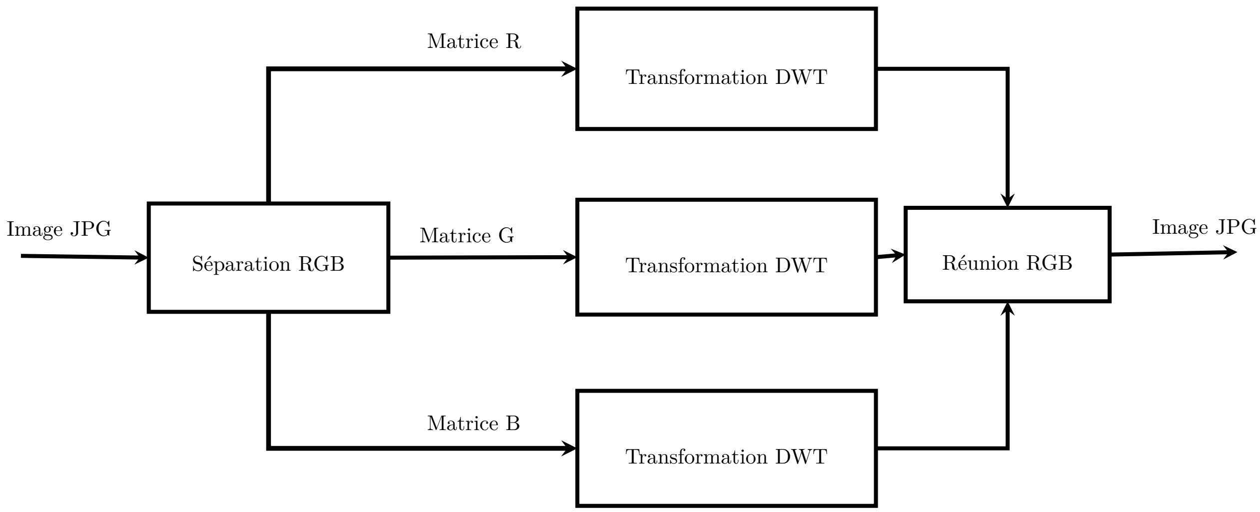

Figure 12 – L'algorithme matriciel

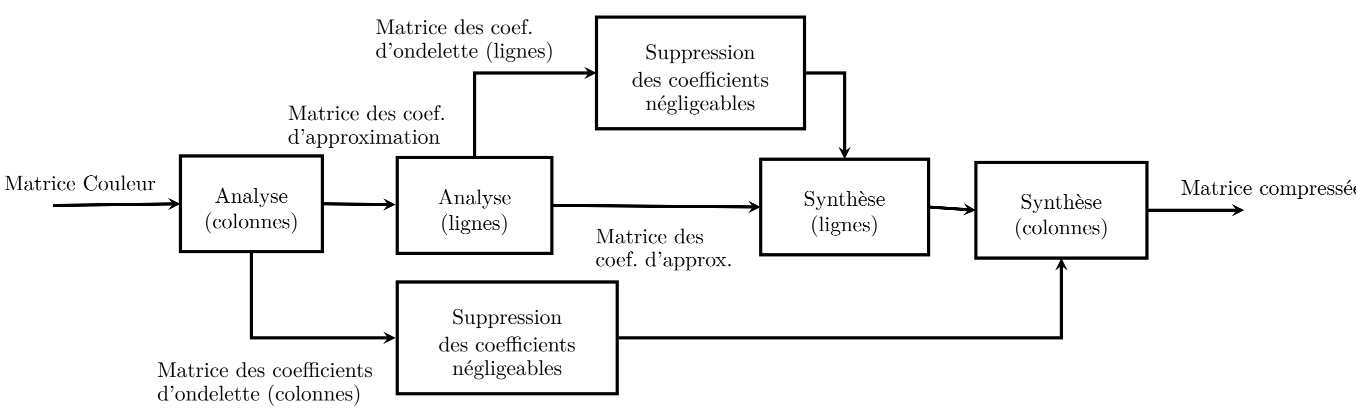

Figure 13 – La transformation appliquée aux matrices

7.3 Exemples d'images traitées avec notre algorithme

La figure 14 montre l'image à laquelle nous avons appliqué la compression. La figure 15 montre un détail de l'image.

FIGURE 14 – L'image de départ

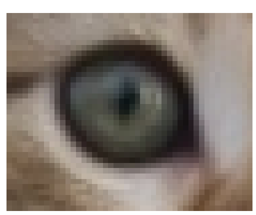

FIGURE 15 – Détail de l'image de départ

La figure 16 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 17, 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.

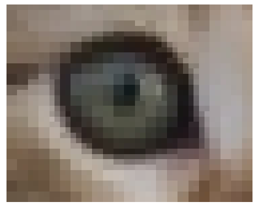

FIGURE 16 – Détail de l'image compressée

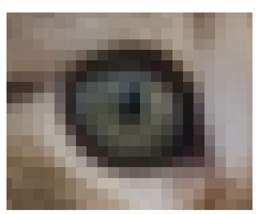

FIGURE 17 – Détail de l'image compressée à 100%

7.4 L'application graphique

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

— Ouverture d'une image
— Enregistrement d'une image
— 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 18 montre la fenêtre principale de l'application, avec une image en cours d'édition. L'interface est minimaliste, mais suffisante.

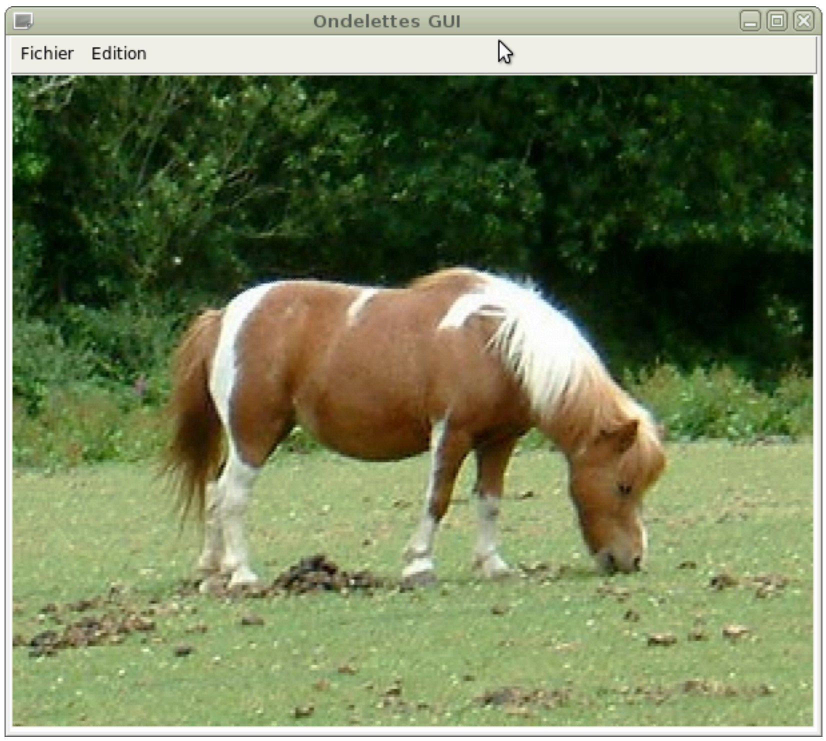

FIGURE 18 – Fenêtre principale de l'application

La figure 19 montre les menus de l'application, qui permettent de choisir entre toutes les actions décrites ci-dessus.

La figure 20 montre le sélecteur de seuil, qui apparaît lorsqu'on choisit «Compresser» ou «Compresser (new)» dans les menus.

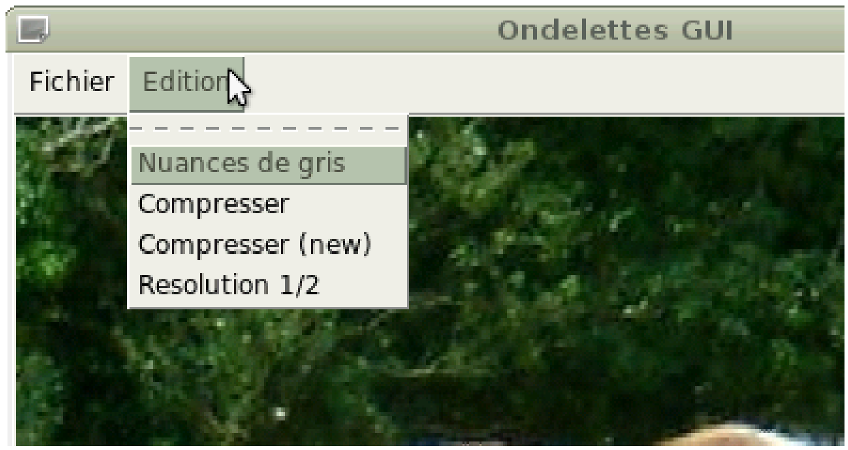

FIGURE 19 – Menus de l'application

FIGURE 20 – Le sélecteur de seuil de compression

7.5 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.

7.5.1 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 21

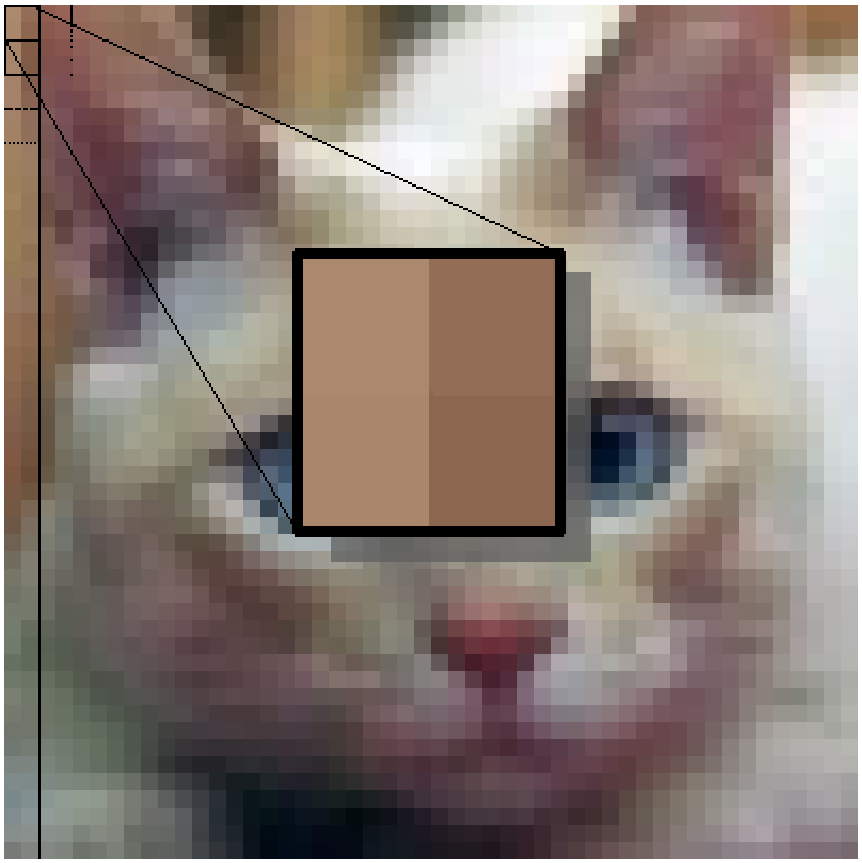

FIGURE 21 – Fonctionnement de l'algorithme

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 21. Le tableau créé est montré par la figure 22.

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

FIGURE 22 – 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 23. 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 23 – 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 24.

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 24 – 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 25.

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

Figure 25 – 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 26. 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 26 – Tableau après synthèse sur les colonnes

L'algorithme effectue ensuite les mêmes opérations sur les lignes. Le résultat est donné par la figure 27

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 27 – 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 28. La figure 29 montre une comparaison entre l'état des pixels avant traitement, et l'état des pixels après.

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.

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

FIGURE 28 – Tableau après synthèse sur les lignes

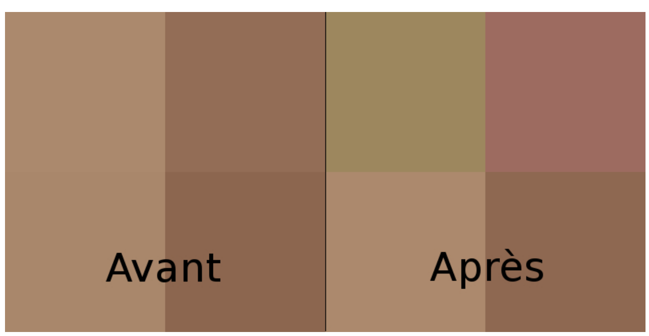

FIGURE 29 – Pixels avant et après traitement

7.5.2 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 30 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.

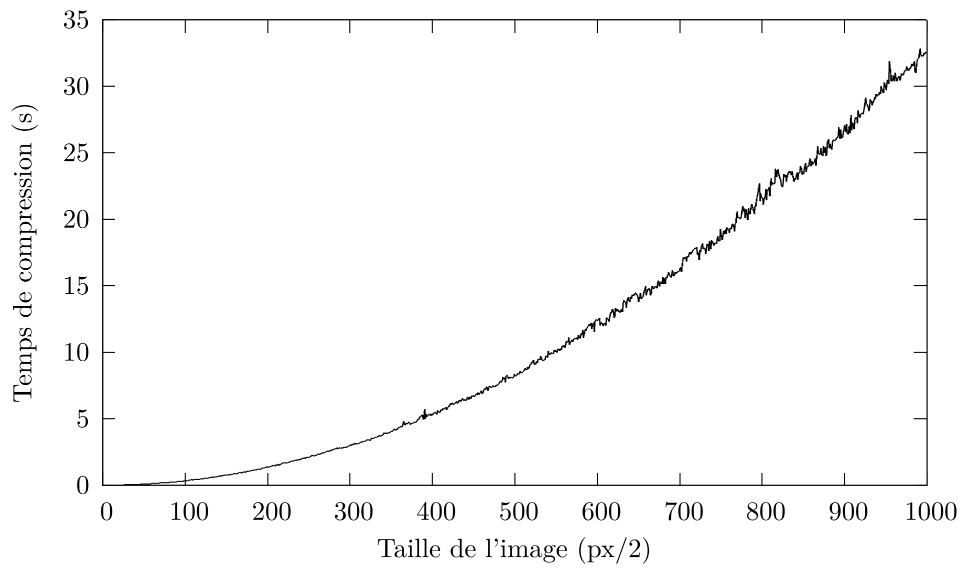

FIGURE 30 – Coût temporel de l'algorithme fasthaar en fonction de la taille de l'image

7.6 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.

7.6.1 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.

7.6.2 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 tranfert 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 31 montre le processus d'archivage.

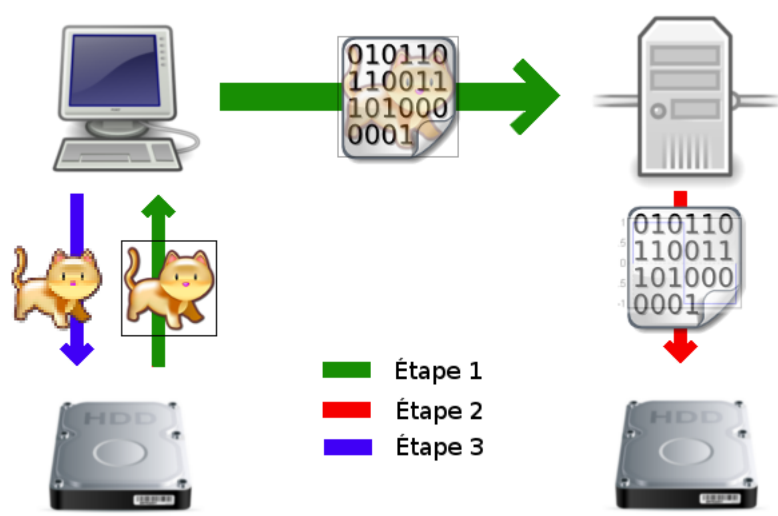

FIGURE 31 – Processus d'archivage

7.6.3 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.

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 32 illustre ce processus

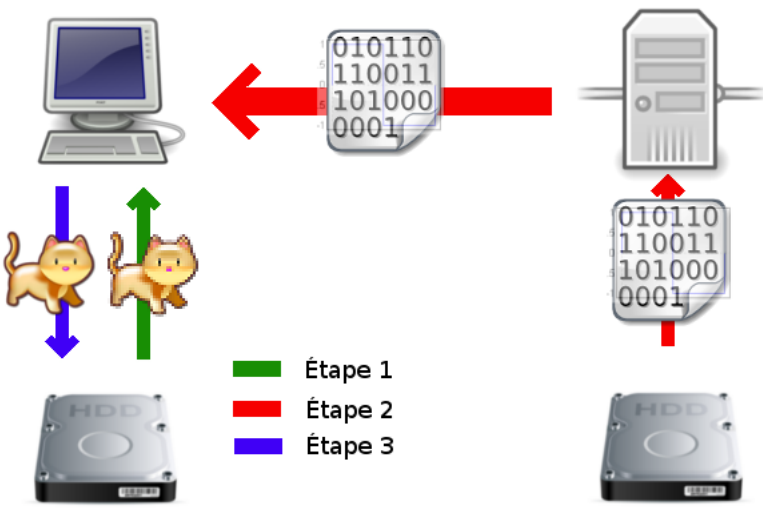

Figure 32 – Processus de restauration

8 Description du code

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

8.1 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).

8.1.1 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 replacée 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 transorme 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 auxilliaire.

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

8.1.2 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 synthesecolonnes 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 résulats restent tout de même impressionnants puisqu'elle peut faire gagner jusqu'à 45% d'espace en seulement quelques secondes.

8.2 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.

8.2.1 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 boite 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.

8.2.2 La classe DialogScale

C'est la classe qui définit la boîte de dialogue de compression (Fig.20). 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 réglette qui permet de choisir la compression, et le bouton «Ok».

La fonction onScale Elle est appelée lorsque l'utilisateur modifie la position de la réglette. 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 réglette, puis ferme la boîte de dialogue.

8.2.3 La fonction main

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

8.3 Le fichier bench.py

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

8.3.1 La fonction image_gen

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

8.3.2 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.

8.3.3 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.

8.3.4 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.

8.3.5 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).

8.4 Le fichier serveur.py

Ce fichier est celui qui est exécuté en permanence sur le serveur, comme décrit à la page 33.

8.4.1 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.

8.4.2 La fonction recvImage

Cette fonction est exécutée quand le client envoie "sendimg" 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.

8.4.3 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.

8.4.4 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.

8.5 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]`

8.5.1 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 définit 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.

8.5.2 La fonction sendImage

Cette fonction est appelée lors de l'utilisation de la commande `s`, elle envoie au serveur `"sendimg"` 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.

8.5.3 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.

8.5.4 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.

8.5.5 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.

9 Conclusion

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, comme l'aura très nettement montré notre implémentation informatique. D'ailleurs, nous aurons nous-mêmes pu exploiter la richesse des ondelettes dans le domaine de transferts-échanges de fichiers en proposant un service qui permet la compression d'images et leur affichage grâce à la connexion à un ordinateur distant.

10 Bibliographie, Liens et Remerciements

— http ://www.cmi.univ-mrs.fr/~melot/Master2/TPsignal_PS.html
— 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 Creative Commons BY-SA :
http ://creativecommons.org/licenses/by-sa/3.0/
— Remerciements à : M. Petitjean

11 Annexes

Tout le code de ce projet est sous licence Creatice Commons BY-SA (voir p.40).

11.1 Fichier ondelettes.py

```
#-------------------------------------------------------------------------------
# Name:          ondelettes.py
# Purpose:       definition de classes de traitement d'image
#
# Author:        Gaetan
#
# Created:       10/07/2013
# Copyright:     (c) Gaetan Bahl/Xavier Friederich 2013
# Licence:       CC-BY-SA
#-------------------------------------------------------------------------------

import Image, math
import time


class Matrice:

    def __init__(self,x,y,what):
        self.tableau = []
        self.x, self.y = x,y

        for i in range(self.x):
            self.tableau.append([])
            for j in range(self.y):
                self.tableau[i].append(what)

    def transpose(self):
        self.tableau = [[row[i] for row in self.tableau] for i in range(self.y)]
        self.x,self.y = self.y,self.x

    def add(self,matrice):

        for i in range(n):
            for j in range(p):
                self.tableau[i][j] += matrice.tableau[i][j]

    def multiply(self,matrice):

        for i in range(self.x):
            for j in range(self.y):
                somme = 0
                for k in range(self.y):
                    somme += self.tableau[i][k]*matrice.tableau[k][j]

    def copy(self,matrice):

        for i in range(self.x):
            for j in range(self.y):
                self.tableau[i][j] = matrice.tableau[i][j]

    def update(self):
        self.x = len(self.tableau)
        self.y = len(self.tableau[0])

    def save(self):
        self.tableau2 = list(self.tableau)

    def save1(self):
        self.mat_orig = list(self.tableau)

    def restore(self):
```

```
        self.tableau = list(self.mat_orig)

class MatriceImage():

    def __init__(self,lienimage):
        self.lienimage = lienimage
        self.image = Image.open(lienimage)
        self.pix = self.image.load()
        self.sizex, self.sizey = self.image.size
        self.matrice = Matrice(self.sizex,self.sizey,self.pix[0,0])
        self.fill()

    def fill(self):
        for i in range(self.sizex):
            for j in range(self.sizey):
                self.matrice.tableau[i][j] = self.pix[i,j]

    def grayscalemean(self):
        for i in range(self.sizex):
            for j in range(self.sizey):
                mean = (self.pix[i,j][0] + self.pix[i,j][1] + self.pix[i,j][2])/3
                self.pix[i,j] = (mean,mean,mean)

    def grayscalemeanmatrix(self):
        self.matrixgray = Matrice(self.sizex,self.sizey,0)
        for i in range(self.sizex):
            for j in range(self.sizey):
                mean = (self.pix[i,j][0] + self.pix[i,j][1] + self.pix[i,j][2])/3
                self.matrixgray.tableau[i][j] = mean

    def getmatrixred(self):
        self.matrixred = Matrice(self.sizex,self.sizey,self.pix[0,0][0])
        for i in range(self.sizex):
            for j in range(self.sizey):
                self.matrixred.tableau[i][j] = self.matrice.tableau[i][j][0]

    def getmatrixblue(self):
        self.matrixblue = Matrice(self.sizex,self.sizey,self.pix[0,0][2])
        for i in range(self.sizex):
            for j in range(self.sizey):
                self.matrixblue.tableau[i][j] = self.matrice.tableau[i][j][2]

    def getmatrixgreen(self):
        self.matrixgreen = Matrice(self.sizex,self.sizey,self.pix[0,0][1])
        for i in range(self.sizex):
            for j in range(self.sizey):
                self.matrixgreen.tableau[i][j] = self.matrice.tableau[i][j][1]

    def save(self,nom):
        self.image.save(nom)

    def ondelette_haar(self,tableau_valeurs,ordre):
        longueur = len(tableau_valeurs)
        coeff_approx, coeff_ondelettes = [],[]
        for i in range(longueur/2):

            coeff_approx.append((tableau_valeurs[2*i] + tableau_valeurs[2*i+1])/2)
            coeff_ondelettes.append((tableau_valeurs[2*i] - tableau_valeurs[2*i+1])/2)

        if ordre == 1:
            return coeff_approx,coeff_ondelettes
        else:
            return ondelette_haar(coeff_approx,ordre -1),coeff_ondelettes

    def getcolonne(self,tab,num):
        col = []
        for i in range(len(tab)):
            col.append(tab[i][num])
```

```
            return col

    def setcolonne(self, matrice, tab, num):
        for i in range(len(tab)):
            matrice[num][i] = tab[i]

    def getligne(self, tab, num):
        return tab[num]

    def apply_haar_lig(self, matrice, chiffre):
        for i in range(len(matrice)):
            matrice[i] = self.ondelette_haar(matrice[i],1)[chiffre]

    def apply_haar_col(self, matrice):
        for i in range(len(matrice[0])):
            col = self.getcolonne(matrice, i)
            col = self.ondelette_haar(col,1)[0]
            self.setcolonne(matrice, col, i)

    def makeimagegray(self, matriceimage):
        matriceimage.update()
        im = Image.new("RGB", (matriceimage.x, matriceimage.y), "white")
        pix = im.load()

        for i in range(matriceimage.x):
            for j in range(matriceimage.y):
                pix[i,j] = (int(matriceimage.tableau[i][j]),int(matriceimage.tableau[i][j]),
                    int(matriceimage.tableau[i][j]))
        return im

    def haar_grayscale(self):
        self.grayscalemeanmatrix()

        self.apply_haar_lig(self.matrixgray.tableau,0)
        self.matrixgray.update()
        self.matrixgray.transpose()
        self.matrixgray.update()
        self.apply_haar_lig(self.matrixgray.tableau,0)
        self.matrixgray.update()
        self.matrixgray.transpose()
        self.imagehaargray = self.makeimagegray(self.matrixgray)

    def update(self):
        self.sizex = matrice.x
        self.sizey = matrice.y

    def create_coef_matrix(self):
        self.matrixcoefr = Matrice(self.matrixred.x, self.matrixred.y,0)
        self.matrixcoefg = Matrice(self.matrixred.x, self.matrixred.y,0)
        self.matrixcoefb = Matrice(self.matrixred.x, self.matrixred.y,0)
        self.matrixcoefr.copy(self.matrixred)
        self.matrixcoefg.copy(self.matrixgreen)
        self.matrixcoefb.copy(self.matrixblue)


    def haar(self):

        for i in [self.matrixred, self.matrixgreen, self.matrixblue]:
            self.apply_haar_lig(i.tableau,0)
            i.update()
            i.save()
            i.transpose()
            i.update()
            self.apply_haar_lig(i.tableau,0)
            i.update()
            i.transpose()

        for i in [self.matrixcoefr, self.matrixcoefg, self.matrixcoefb]:
```

```
            i.save1()
            self.apply_haar_lig(i.tableau,1)
            i.update()
            i.save()
            i.update()
            i.restore()
            self.apply_haar_lig(i.tableau,0)
            i.update()
            i.transpose()
            i.update()
            self.apply_haar_lig(i.tableau,1)
            i.update()
            i.transpose()



    def makeimage(self):
        im = Image.new("RGB", (self.matrixred.x, self.matrixred.y), "white")
        pix = im.load()

        for i in range(self.matrixred.x):
            for j in range(self.matrixred.y):
                pix[i,j] = (int(self.matrixred.tableau[i][j]),int(self.matrixgreen.tableau[i][j]),int(self.matrixblue.tableau[i][j]))
        return im

    def compression(self,epsilon):
        for tab in [self.matrixcoefr.tableau,self.matrixcoefg.tableau,self.matrixcoefb.tableau]:
            for i in tab:
                for j in range(len(i)):
                    if abs(i[j]) < epsilon:
                        i[j] = 0

        for tab in [self.matrixcoefr.tableau2,self.matrixcoefg.tableau2,self.matrixcoefb.tableau2]:
            for i in tab:
                for j in range(len(i)):
                    if abs(i[j]) < epsilon:
                        i[j] = 0

    def syntheselignes(self):
        for i in range(self.sizex/2):
            for j in range(self.sizey/2):
                self.pix[2*i,2*j] = (int(self.matrixred.tableau[i][j] + self.matrixcoefr.tableau[i][j]), int(self.matrixgreen.tableau[i][j] + self.matrixcoefg.tableau[i][j]) , int(self.matrixblue.tableau[i][j] + self.matrixcoefb.tableau[i][j]))
                self.pix[2*i+1,2*j] = (int(self.matrixred.tableau[i][j] - self.matrixcoefr.tableau[i][j]), int(self.matrixgreen.tableau[i][j] - self.matrixcoefg.tableau[i][j]) , int(self.matrixblue.tableau[i][j] - self.matrixcoefb.tableau[i][j]))

    def synthesecolonnes(self):

        for y in range(len(self.matrixcoefr.tableau2[0])):

            for x in range(len(self.matrixcoefr.tableau2)):

                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]))

    def clearimage(self):
        for i in range(self.sizex):
```

```
            for j in range(self.sizey):
                self.pix[i,j] = (255,0,0)

    def fasthaar(self,epsilon,xa,xb,ya,yb):

        for x in range(xa/2,xb/2):

            for y in range(ya/2,yb/2):
                #copier les 4 pixels dans un carre

                carres = []

                for i in range(3):
                    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]]])

                ##ANALYSE
                ondlhaut = [(carres[i][0][0] - carres[i][1][0])/2 for i in range(3)]
                ondlbas  = [(carres[i][0][1] - carres[i][1][1])/2 for i in range(3)]

                for i in range(3):
                    carres[i][0][0] = (carres[i][0][0] + carres[i][1][0])/2
                    carres[i][0][1] = (carres[i][0][1] + carres[i][1][1])/2

                ondlmix = [(carres[i][0][0] - carres[i][0][1])/2 for i in range(3)]

                for i in range(3):
                    carres[i][0][0] = (carres[i][0][0] + carres[i][0][1])/2

                ##SYNTHESE
                for i in range(3):
                    if abs(ondlmix[i]) < epsilon:
                        carres[i][0][1] = carres[i][0][0]
                    else:
                        carres[i][0][1] = carres[i][0][0] - ondlmix[i]
                        carres[i][0][0] = carres[i][0][0] + ondlmix[i]

                    if abs(ondlhaut[i]) < epsilon:
                        carres[i][1][0] = carres[i][0][0]
                    else:
                        carres[i][1][0] = carres[i][0][0] - ondlhaut[i]
                        carres[i][0][0] = carres[i][0][0] + ondlhaut[i]

                    if abs(ondlbas[i]) < epsilon:
                        carres[i][1][1] = carres[i][0][1]
                    else:
                        carres[i][1][1] = carres[i][0][1] - ondlbas[i]
                        carres[i][0][1] = carres[i][0][1] + ondlbas[i]

                for i in [2*x,2*x+1]:
                    for j in [2*y,2*y+1]:
                        self.pix[i,j] = (carres[0][i-2*x][j-2*y],carres[1][i-2*x][j-2*y],carres[2][i-2*x][j-2*y])



def main():

    image = MatriceImage("images/chat.jpg")
    start = time.time()

    image.fasthaar(2,10,image.sizex,0,image.sizey)

    end = time.time()
    print end - start
    image.image.save("chat.jpg",'JPEG',quality = 100)

if __name__ == '__main__':
```

```
    main()
```

11.2 Fichier ondelettesGUI.py

```
#-------------------------------------------------------------------------------
# Name:        ondelettesGUI.py
# Purpose:     interface graphique du TIPE sur les ondelettes
#
# Author:      Gaetan Bahl
#
# Created:     08/08/2013
# Copyright:   (c) Gaetan Bahl 2013
# Licence:     CC-BY-SA
#-------------------------------------------------------------------------------

from Tkinter import *
import tkFileDialog, ImageTk
from ondelettes import *
from ttk import Frame, Style
from Tkconstants import *
from subprocess import call


class Appli(Frame):

    def __init__(self, parent):
        Frame.__init__(self, parent)

        self.parent = parent
        self.initUI()

    def cli(self):
        call(["./client.py", "sr", self.filename])


    def initUI(self):

        self.parent.title("Ondelettes GUI")
        self.pack(fill=BOTH, expand=1)
        menubar = Menu(self.parent)
        self.parent.config(menu=menubar)

        fileMenu = Menu(menubar)
        fileMenu.add_command(label="Ouvrir", command=self.askopenfilename)
        fileMenu.add_command(label="Enregistrer", command=self.asksaveasfilename)
        fileMenu.add_command(label="Exit", command=self.onExit)
        fileMenu.add_command(label="Send to Server", command=self.cli)
        menubar.add_cascade(label="Fichier", menu=fileMenu)

        editMenu = Menu(menubar)
        editMenu.add_command(label="Nuances de gris", command=self.grayscale)
        editMenu.add_command(label="Compresser", command=self.askcompression)
        editMenu.add_command(label="Compresser (new)", command=self.askcompression2)
        editMenu.add_command(label="Resolution 1/2", command=self.onExit)
        menubar.add_cascade(label="Edition", menu=editMenu)

        Style().configure("TFrame", background="#FFF")

    def askopenfilename(self):

        filename = tkFileDialog.askopenfilename(defaultextension = "jpg")
        self.filename = filename
        self.matriceimage = MatriceImage(filename)
        self.image = self.matriceimage.image
        self.imgtk = ImageTk.PhotoImage(self.image)
        self.labelimg = Label(self,image=self.imgtk)
```

```
        self.labelimg.image = self.imgtk
        self.labelimg.place(x = 0,y=0)
        self.labelimg.pack()


        self.parent.geometry(str(self.matriceimage.sizex+5)+"x"+str(self.matriceimage.sizey+5)+"+100+300")

    def asksaveasfilename(self):

        filename = tkFileDialog.asksaveasfilename(defaultextension = "jpg")

        self.image.save(filename,'JPEG',quality = 100)

    def askcompression(self):
        global compress
        fen = Tk()
        fen.geometry("300x100+300+300")
        box = DialogScale(fen)
        fen.mainloop()
        fen.destroy()
        self.compression()

    def askcompression2(self):
        global compress
        fen = Tk()
        fen.geometry("300x100+300+300")
        box = DialogScale(fen)
        fen.mainloop()
        fen.destroy()
        self.compression2()

    def compression(self):

        self.matriceimage.getmatrixblue()
        self.matriceimage.getmatrixgreen()
        self.matriceimage.getmatrixred()
        self.matriceimage.create_coef_matrix()
        self.matriceimage.haar()
        self.matriceimage.compression(compress)
        self.matriceimage.synthselignes()
        self.matriceimage.synthesecolonnes()
        self.labelimg.destroy()

        self.displayimage()

    def compression2(self):


        self.matriceimage.fasthaar(compress, 0, self.matriceimage.sizex, 0, self.matriceimage.sizey)

        self.labelimg.destroy()

        self.displayimage()

    def grayscale(self):
        self.matriceimage.grayscalemeanmatrix()
        self.image = self.matriceimage.makeimagegray(self.matriceimage.matrixgray)
        self.matriceimage.image = self.matriceimage.makeimagegray(self.matriceimage.matrixgray)
        self.labelimg.destroy()

        self.displayimage()

    def displayimage(self):
        self.imgtk = ImageTk.PhotoImage(self.matriceimage.image)
        self.labelimg = Label(self,image=self.imgtk)
```

```python
        self.labelimg.image = self.imgtk
        self.labelimg.place(x = 0,y=0)
        self.labelimg.pack()
        self.update()



    def onExit(self):
        self.quit()


class DialogScale(Frame):
    def __init__(self, parent):
        Frame.__init__(self, parent)

        self.parent = parent
        self.initUI()

    def initUI(self):

        self.parent.title("Compression")
        self.style = Style()
        self.style.theme_use("default")

        self.pack(fill=BOTH, expand=1)

        scale = Scale(self, from_=0, to=255,command=self.onScale, orient= HORIZONTAL)
        scale.place(x=90, y=20)


        self.label2 = Label(self, text="Choisissez un niveau de compression")
        self.label2.place(x=52, y=0)
        self.quitButton = Button(self, text="    Ok    ",command=self.ok)
        self.quitButton.place(x=120, y=65)

    def onScale(self, val):

        self.variable = int(val)

    def ok(self):
        global compress
        compress = self.variable
        self.quit()


def main():

    root = Tk()
    root.geometry("250x250+300+300")
    app = Appli(root)
    root.mainloop()


if __name__ == '__main__':
    main()
```

11.3 Fichier bench.py

```python
#-------------------------------------------------------------------------------
# Name:        bench.py
# Purpose:     benchmarking d'algorithmes
#
# Author:      Gaetan Bahl
#
# Created:     19/08/2013
# Copyright:   (c) Gaetan Bahl 2013
```

```
# Licence:      CC-BY-SA
#--------------------------------------------------------------------


import Image
import time
import random
from ondelettes import *


def image_gen(x):
    im = Image.new("RGB", (x, x), "white")
    return im


def randomize(pix, x):
    for i in range(x):
        for j in range(x):
            pix[i, j] = (random.randrange(255), random.randrange(255),
             random.randrange(255))


def bench_square(ran):

    for i in ran:
        image = image_gen(2 * i)
        pix = image.load()
        randomize(pix, 2 * i)
        start = time.time()
        fasthaar(pix, 0, 0, 2 * i, 0, 2 * i)
        end = time.time()
        print str(i) + " " + str(end - start)


def fasthaar(pix, epsilon, xa, xb, ya, yb):

    for x in range(xa / 2, xb / 2):

        for y in range(ya / 2, yb / 2):

            carres = []

            for i in range(3):
                carres.append([[pix[2 * x, 2 * y][i], pix[2 * x, 2 * y + 1][i]],
                     [pix[2 * x + 1, 2 * y][i], pix[2 * x + 1, 2 * y + 1][i]]])

            ondlhaut = [(carres[i][0][0] - carres[i][1][0]) / 2
             for i in range(3)]
            ondlbas = [(carres[i][0][1] - carres[i][1][1]) / 2
             for i in range(3)]

            for i in range(3):
                carres[i][0][0] = (carres[i][0][0] + carres[i][1][0]) / 2
                carres[i][0][1] = (carres[i][0][1] + carres[i][1][1]) / 2

            ondlmix = [(carres[i][0][0] - carres[i][0][1]) / 2
              for i in range(3)]

            for i in range(3):
                carres[i][0][0] = (carres[i][0][0] + carres[i][0][1]) / 2

            for i in range(3):
                if abs(ondlmix[i]) < epsilon:
                    carres[i][0][1] = carres[i][0][0]
                else:
                    carres[i][0][1] = carres[i][0][0] - ondlmix[i]
                    carres[i][0][0] = carres[i][0][0] + ondlmix[i]

```

```
                if abs(ondlhaut[i]) < epsilon:
                    carres[i][1][0] = carres[i][0][0]
                else:
                    carres[i][1][0] = carres[i][0][0] - ondlhaut[i]
                    carres[i][0][0] = carres[i][0][0] + ondlhaut[i]

                if abs(ondlbas[i]) < epsilon:
                    carres[i][1][1] = carres[i][0][1]
                else:
                    carres[i][1][1] = carres[i][0][1] - ondlbas[i]
                    carres[i][0][1] = carres[i][0][1] + ondlbas[i]

            for i in [2 * x, 2 * x + 1]:
                for j in [2 * y, 2 * y + 1]:
                    pix[i, j] = (carres[0][i - 2 * x][j - 2 * y],
                     carres[1][i - 2 * x][j - 2 * y],
                     carres[2][i - 2 * x][j - 2 * y])


def main():
    bench_square(range(1,1000))


if __name__ == '__main__':
    main()
```

11.4 Fichier serveur.py

```
#!/usr/bin/python2
#-------------------------------------------------------------------------------
# Name:        serveur.py
# Purpose:     serveur du TIPE sur les ondelettes
#
# Author:      Gaetan Bahl
#
# Created:     20/08/2013
# Copyright:   (c) Gaetan Bahl 2013
# Licence:     CC-BY-SA
#-------------------------------------------------------------------------------


import socket
import Image
import time
import struct



def recvImage(conn,msg):
    title = msg.split(' ')
    sizex,sizey = int(title[1]),int(title[2])
    image = Image.new("RGB", (sizex,sizey), "white" )
    pix = image.load()
    conn.send(b"ok")
    msg, img = "", ""
    unpacker = struct.Struct("BBB")
    while msg.endswith("end") == False:
        img += msg
        msg = conn.recv(3)


    print "pixels recus"

    for i in range(sizex):
        for j in range(sizey):
```

```python
                pixel = unpacker.unpack(img[(i*sizey + j)*unpacker.size:(i*sizey + j + 1)*
                    unpacker.size])
                pix[i, j] = (int(pixel[0]), int(pixel[1]), int(pixel[2]))

    print "image created"

    image.save("srv" + title[3], 'JPEG', quality = 100)
    print "image enregistree"
    conn.send(b'saved image')


def fasthaar_srv(socket, msg, epsilon=0):
    t = msg.split(' ')
    image = Image.open(t[1])
    pix = image.load()

    fichier = open("coef/" + t[1][:-4] + ".odl", 'wb')
    sizex,sizey = image.size


    print "compression en cours"
    for x in range(sizex/2):
        for y in range(sizey/2):

            carres = []

            for i in range(3):
                carres.append([[pix[2 * x, 2 * y][i], pix[2 * x, 2 * y + 1][i]],
                     [pix[2 * x + 1, 2 * y][i], pix[2 * x + 1, 2 * y + 1][i]]])

            ondlhaut = [(carres[i][0][0] - carres[i][1][0]) / 2
             for i in range(3)]
            ondlbas = [(carres[i][0][1] - carres[i][1][1]) / 2
             for i in range(3)]

            for i in range(3):
                carres[i][0][0] = (carres[i][0][0] + carres[i][1][0]) / 2
                carres[i][0][1] = (carres[i][0][1] + carres[i][1][1]) / 2

            ondlmix = [(carres[i][0][0] - carres[i][0][1]) / 2
              for i in range(3)]


            for i in [ondlmix,ondlhaut,ondlbas]:
                fichier.write(struct.pack("bbb",i[0],i[1],i[2]))


    fichier.close()
    print 'compression OK'
    socket.send(b"OK")


def sendCoef(sock, msg):
    banana = msg.split(" ")
    fichier = open("coef/srv" + banana[1][:-4] + ".odl", 'rb')
    print "envoi des coefficients"
    txt = fichier.readlines()
    for i in txt:
        sock.send(i)
    print "coefficients envoyes"
    sock.send(b'end')
    ms = sock.recv(2)
    print ms.decode()
    fichier.close()



def main():
```

```

    hote = ''
    port = 13337

    connexion = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    connexion.bind((hote, port))
    while 1:
        connexion.listen(5)

        msg_recu = b""
        print "attente d'un client"
        connexion_client, infos_connexion = connexion.accept()

        print "client connected"

        while msg_recu != b"stop":
            time.sleep(0.1)
            msg = msg_recu.decode()

            if msg.startswith('sendimg'):
                print "image en cours de reception"
                recvImage(connexion_client, msg)
            if msg.startswith('compress'):
                fasthaar_srv(connexion_client, msg)
            if msg.startswith('coef'):
                sendCoef(connexion_client, msg)


            msg_recu = connexion_client.recv(1024)




    print("Exiting")
    connexion_client.close()
    connexion.close()

if __name__ == "__main__":
    main()
```

11.5 Fichier client.py

```
#!/usr/bin/python2
#-------------------------------------------------------------------------------
# Name:         client.py
# Purpose:      client du TIPE sur les ondelettes
#
# Author:       Gaetan Bahl
#
# Created:      20/08/2013
# Copyright:    (c) Gaetan Bahl 2013
# Licence:      CC-BY-SA
#-------------------------------------------------------------------------------

import socket
import Image
import time
import struct
import sys

def sendImage(socket,link):
    image = Image.open(link)
    pix = image.load()
    x,y = image.size
    string = "sendimg " + str(x) + " " + str(y) + " " + link
    socket.send(string.encode('ascii'))
```

```
    reply = socket.recv(32)
    if reply == b"ok":
        t = ""
        for i in range(x):

            for j in range(y):
                r,g,b = pix[i,j]
                packer = struct.Struct("BBB")
                pixel = packer.pack(r, g, b)
                socket.send(pixel)


    socket.send(b"end")
    reply = socket.recv(11)

    print  "server says : " + reply

def askcompress(connexion, nom):

    mess = "compress " + nom
    connexion.send(mess.encode('ascii'))
    reply = connexion.recv(2)
    print "server says : " + reply
    return reply

def storeImage(link):
    image = Image.open(link)
    pix = image.load()
    x,y = image.size

    im = Image.new("RGB", (x / 2, y / 2), "white")
    pi = im.load()

    for i in range(x / 2):
        for j in range(y / 2):
            r = (((pix[2 * i, 2 * j][0] + pix[2 * i + 1, 2 * j][0]) / 2) + ((pix[2 * i, 2 * j + 1][0] + pix[2 * i + 1, 2 * j + 1][0]) / 2)) / 2
            g = (((pix[2 * i, 2 * j][1] + pix[2 * i + 1, 2 * j][1]) / 2) + ((pix[2 * i, 2 * j + 1][1] + pix[2 * i + 1, 2 * j + 1][1]) / 2)) / 2
            b = (((pix[2 * i, 2 * j][2] + pix[2 * i + 1, 2 * j][2]) / 2) + ((pix[2 * i, 2 * j + 1][2] + pix[2 * i + 1, 2 * j + 1][2]) / 2)) / 2
            pi[i, j] = (r,g,b)

    im.save("stor" + link, 'JPEG', quality = 100)



def askcoeff(conn, nom):

    unpacker = struct.Struct("bbbbbbbbb")

    print 'demande en cours'
    mess = "coef " + nom
    conn.send(mess.encode("ascii"))

    print "reception coefs"
    coefs = ''
    msg = ''
    while msg.endswith(b'end') == False:
        coefs += msg
        msg = conn.recv(9)
    print str(len(coefs)) + " octets recus"
    print "reconstitution image"
    image = Image.open("stor" + nom)
    pix = image.load()
    dim = image.size
    im =  Image.new("RGB", (dim[0] * 2, dim[1] * 2), "white" )
    px = im.load()
```

```
    for i in range(dim[0]):
        for j in range(dim[1]):
            tab = unpacker.unpack(coefs[(i*dim[1] + j)* unpacker.size : (i*dim[1] + j +1)* unpacker.size])

            px[2*i,2*j] = ( pix[i,j][0] + tab[0] + tab[3], pix[i,j][1] + tab[1] + tab[4], pix[i,j][2] + tab[2] + tab[5])
            px[2*i+1,2*j] = ( pix[i,j][0] + tab[0] - tab[3], pix[i,j][1] + tab[1] - tab[4], pix[i,j][2] + tab[2] - tab[5])
            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])
            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])


    print "enregistrement"
    im.save('2' + nom, 'JPEG', quality = 100)
    conn.send(b'ok')


def main(arg):

    connexion = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    if 'r' in arg[1]:
        try :
            connexion.connect((argv[3], 13337))
        except:
            connexion.connect(('srv.ordiclic.eu', 13337))
    else:
        connexion.connect(('localhost', 13337))

    if 's' in arg[1]:
        sendImage(connexion, arg[2])
        askcompress(connexion, 'srv' + arg[2])
        storeImage(arg[2])
    if 'd' in arg[1]:
        askcoeff(connexion, arg[2])

    connexion.send(b"stop")
    connexion.close()


if __name__ == "__main__":
    main(sys.argv)
```