TIPE : Introduction à la théorie des ondelettes

Xavier Friederich et Gaétan Bahl

12 juin 2014

Table des matières

1 Introduction et problématisation **2**
1.1 Limites de l'analyse de Fourier . . . 2
1.2 Possibilité de perfectionnement ? . . . 2

2 La théorie des ondelettes - Définition et conditions d'existence **2**

3 Transformation en ondelettes **3**
3.1 Transformation en ondelettes continue . . . 3
3.2 Transformation en ondelettes discrète. . . . 5
3.3 Comparaison avec la transformation de Fourier . . . 5

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

5 Application à la compression des données **6**

6 Utilisation pratique de la transformation par ondelettes discrète **6**
6.1 Le choix des outils . . . 6
6.1.1 Le langage : Python . . . 6
6.1.2 La librairie de traitement d'image : PIL . . . 6
6.1.3 La librairie d'interface graphique : Tkinter . . . 6
6.2 L'algorithme matriciel . . . 6
6.3 L'application graphique . . . 7

7 Conclusion **8**

8 Annexes **9**
8.1 Sur la transformation de Fourier . . . 9
8.1.1 L'utilisation de la transformée de Fourier . . . 9
8.1.2 Définition de la transformation de Fourier . . . 9
8.1.3 Propriétés de la transformation de Fourier . . . 9
8.1.4 Transformée inverse . . . 9
8.1.5 Exemples de transformées de Fourier . . . 10
8.1.6 Applications. . . . 10
8.2 Transformation par les ondelettes. . . . 12
8.2.1 Exemples d'ondelettes mères : . . . 12
8.2.2 Comparaisons de méthodes de compression . . . 13
8.2.3 L'algorithme de Mallat : démonstration dans le cas des ondelettes de Haar. . . . 13
8.2.4 Schématisation de l'algorithme de Mallat . . . 16
8.2.5 Représentation matricielle de l'algorithme utilisant les ondelettes de Haar . . . 16
8.2.6 Exemples d'images traitées avec notre algorithme . . . 18
8.3 Code de l'algorithme en Python et application graphique . . . 20
8.3.1 Fichier ondelettes.py . . . 20
8.3.2 Fichier ondelettesGUI.py . . . 26

9 Bibliographie, Liens et Remerciements **30**

1 Introduction et problématisation

Le traitement du signal prend de plus en plus d'ampleur dans la vie de tous les jours. Il permet notamment de réduire la mémoire occupée par un signal (sonore, graphique...) enregistré sur un ordinateur par exemple. 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. L'analyse de Fourier constitue un outil efficace mais qui présente incontestablement des limites.

1.1 Limites de l'analyse de Fourier

On rappelle que si f est une fonction continue 2π-périodique de classe $\mathcal{C}^1$ par morceaux, f est somme de sa série de Fourier, ce qui se réécrit de la façon suivante :

$$\forall t \in \mathbb{R}, f(t) = \frac{a_0}{2} + \sum_{n=1}^{+\infty} (a_n(f)\cos(nt) + b_n(f)\sin(nt))$$

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

Bien évidemment, la plupart des signaux sont non-périodiques. Dans le cas général et dans la limite de la possibilité de le faire, on effectue une transformation de Fourier.
La transformation de Fourier d'une fonction f produit comme transformée une fonction $\hat{f}$ à valeurs complexes. On comprendra que la très grande majorité des signaux est numérique et que les définitions mathématiques avec les intégrales (domaine du continu) ne sont pas adaptées au domaine du discret.
La transformée de Fourier telle qu'elle est utilisée dans un ordinateur (transformée de Fourier discrète (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 sont identiques (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 (qui a pour but un calcul rapide de la TFD et de son inverse) nécessite que le nombre d'échantillons 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.

1.2 Possibilité de perfectionnement ?

Certes, il s'agit de réduire l'espace occupé par les fichiers mais il faut aussi conserver une bonne qualité du signal. Or, la recherche récente a montré que la transformation par les ondelettes permet de trouver le juste équilibre. La transformation par les ondelettes est une transformation des fonctions/signaux plus performante car elle est notamment capable de détecter les portions du signal qui varient plus rapidement que d'autres.

En quoi consiste la transformation par les ondelettes ? Dans quelle mesure la compression par les ondelettes est-elle plus efficace que l'utilisation de la transformation de Fourier ?

A l'aide de la démonstration de l'algorithme de Mallat et de notre propre implémentation de l'algorithme avec visualisation graphique, on tentera de répondre à ces questions.

2 La théorie des ondelettes - Définition et conditions d'existence

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 continue ou discrète. Le travail suivant fera l'objet du cas particulier de la transformation en ondelettes unidimensionnelle.

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 $\mathcal{L}^1(\mathbb{R},\mathbb{C}) \cap \mathcal{L}^2(\mathbb{R},\mathbb{C})$ qui vérifie la condition suivante :

$\displaystyle\int_{\mathbb{R}} \frac{|\hat{\psi}(\omega)|^2}{|\omega|} d\omega < \infty$ où $\hat{\psi}$ est la transformée de Fourier de ψ, donnée par la formule $\hat{\psi}(\omega) = \displaystyle\int_{-\infty}^{+\infty} \psi(t).e^{-2i\pi\omega t} dt$.

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

ψ vérifie alors : $\displaystyle\int_{\mathbb{R}} \psi(t).dt = 0$.

Démonstration :

Il faut que $\omega \longmapsto \dfrac{|\hat{\psi}(\omega)|^2}{|\omega|}$ ait un prolongement continu en 0, donc il faut que $\hat{\psi}(0) = \displaystyle\int_{\mathbb{R}} \psi(t).dt = 0$.

En $+\infty$ ou $-\infty$, il n'y a pas de problème de convergence de l'intégrale définissant la condition d'admissibilité car par théorème, $\hat{\psi}$ est une fonction de carré intégrable sur $\mathbb{R}$ puisque ψ l'est.

Remarque :

Si $\hat{\psi}$ est continûment différentiable, alors on a l'équivalence : $\displaystyle\int_{\mathbb{R}} \frac{|\hat{\psi}(\omega)|^2}{|\omega|} d\omega < \infty \iff \hat{\psi}(0) = 0$.

(Il suffit d'écrire la formule de Taylor en 0 à l'ordre 1.)

Si l'ondelette -fonction analysante- est convenablement choisie, la transformation en ondelettes est inversible et la fonction peut être reconstruite après *analyse* lors d'une étape appelée *synthèse*.

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.

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 ondelettes continue

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

La transformée en ondelettes continue de la fonction f est définie à facteur constant près comme le produit scalaire (produit scalaire complexe usuel sur l'espace de fonctions, noté par la suite $\langle \cdot \mid \cdot \rangle$) de f et de ψ.

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

Les $\mathcal{W}_f(a,b)$ sont les *coefficients d'ondelettes*. 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 $a \neq 1$.

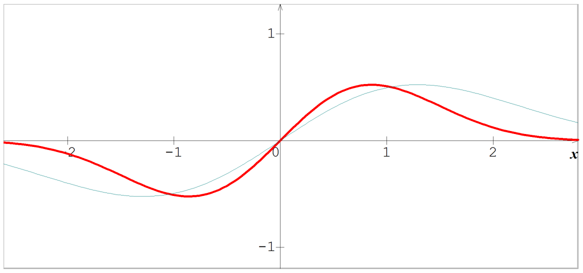

Figure 1 – Dilatation d'ondelette

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

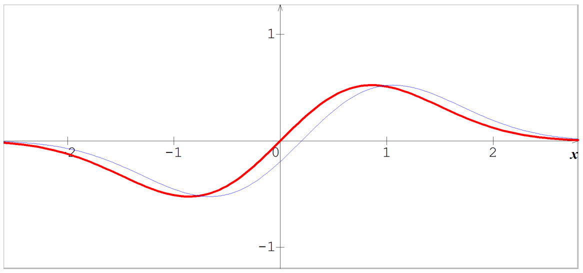

Figure 2 – Translation d'ondelette

Le graphique suivant (Fig.3), représentant l'application $\mathcal{W}$ appliquée en la fonction (ou signal en 1D) $f = \Pi$ (définie en annexe au paragraphe 8.1.5) et utilisant l'ondelette mère de Morlet (définie en 8.2.1), 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.

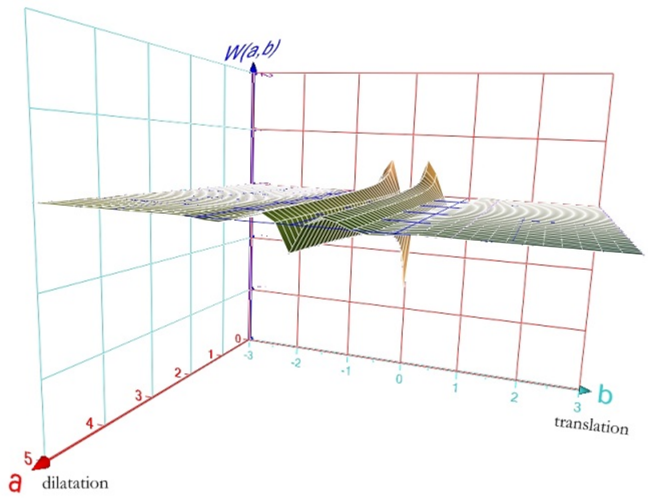

Figure 3 – Le graphe de la fonction $(a, b) \longmapsto \mathcal{W}_\Pi(a, b)$

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 :

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

Δt peut être choisi « géométriquement » ; les paramètres de translation 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 $\mathcal{L}^2(\mathbb{R})$ car 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.

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, ce que l'on observe en 8.2.2 où l'on compare les deux transformations).

Enfin, le coût ou la complexité 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 Algorithme de décomposition en ondelettes de Stéphane Mallat (1989)

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.

Nous avons fait une démonstration mathématique de cet algorithme présente en annexe ; pour simplifier, nous nous sommes limités au cas particulier de décomposition d'un signal par les ondelettes de Haar.

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 n'utilisant que des sommes ou des différences.

La démonstration au paragraphe 8.2.3 est faite pour la première partie de l'algorithme, l'*analyse*.

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

5 Application à la compression des données

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

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

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

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

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

6 Utilisation pratique de la transformation par ondelettes discrète

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

Le code complet de notre algorithme est disponible en annexe à la fin de ce dossier. Il se compose de deux parties (deux fichiers), l'une constituant la décomposition en tant que telle et l'autre permettant, via une application graphique, d'illustrer la décomposition sur une image 2D que l'utilisateur peut choisir.

6.1 Le choix des outils

Pour nos programmes, nous avons utilisé le langage Python et certaines librairies.

6.1.1 Le langage : Python

Nous avons choisi d'utiliser le langage Python pour plusieurs raisons. Celui-ci offre beaucoup de possibilités, est facile à utiliser et 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.

6.1.2 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.

6.1.3 La librairie d'interface graphique : Tkinter

Pour la partie «interface graphique utilisateur» (ou GUI), nous avons utilisé la librairie Tkinter, simple d'utilisation et convenant parfaitement à notre projet : concevoir une application simple d'utilisation et montrant la compression d'une image en utilisant notre algorithme.

6.2 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 4 est un schéma-bloc montrant l'algorithme. La partie «Transformation DWT» est explicitée par la figure 5

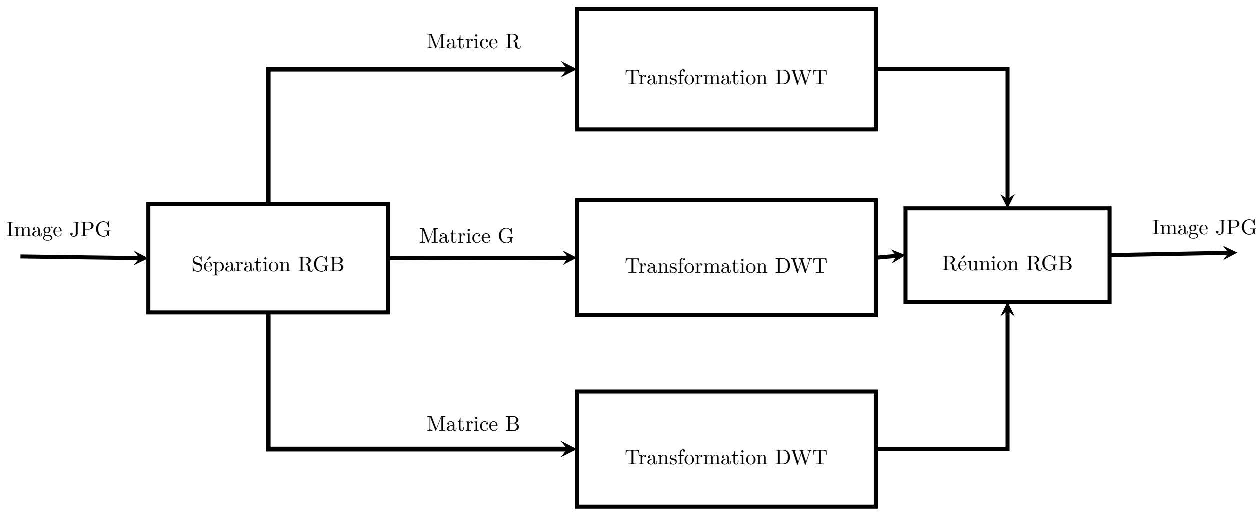

FIGURE 4 – L'algorithme matriciel

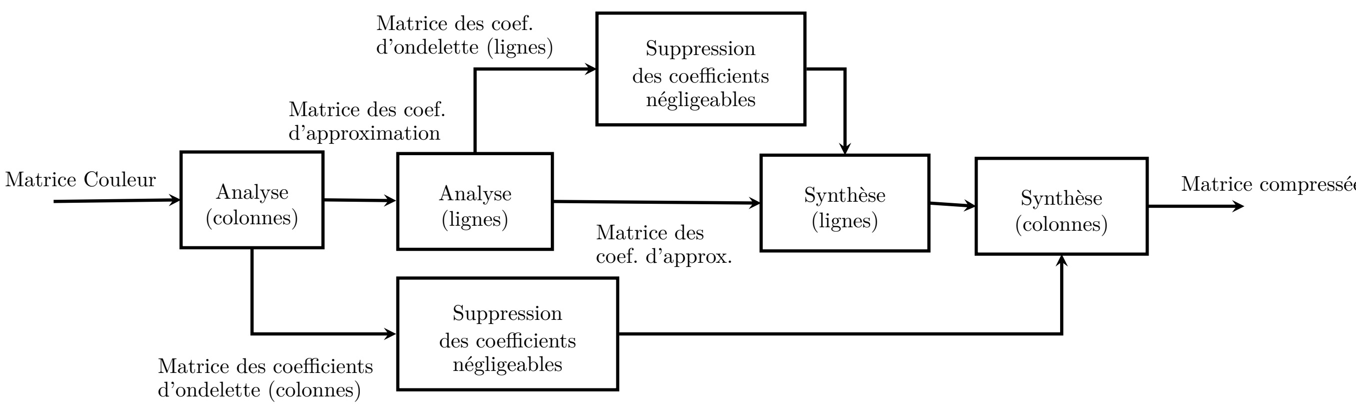

FIGURE 5 – La transformation appliquée aux matrices

6.3 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%

La figure 6 montre la fenêtre principale de l'application, avec une image en cours d'édition. L'interface est minimaliste, mais suffisante.

La figure 7 montre les menus de l'application, qui permettent de choisir entre toutes les actions décrites ci-dessus.

La figure 8 montre le sélecteur de seuil, qui apparaît lorsqu'on choisit «Compresser» ou «Compresser (new)» dans les menus.

FIGURE 6 – Fenêtre principale de l'application

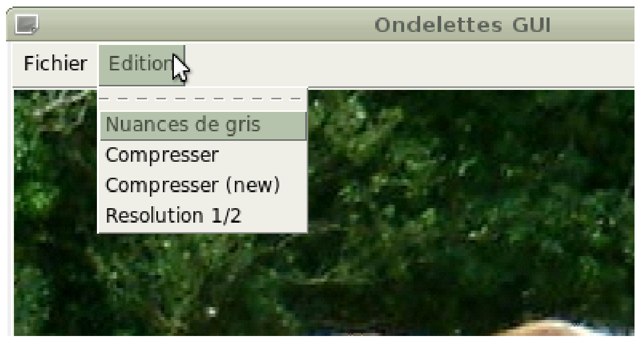

FIGURE 7 – Menus de l'application

FIGURE 8 – Le sélecteur de seuil de compression

7 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.

8 Annexes

8.1 Sur la transformation de Fourier

8.1.1 L'utilisation de la transformée de Fourier

Il est nécessaire dans le cas général de signaux non périodiques de passer d'une écriture discrète en une écriture en somme continue.
Le cadre le plus naturel pour définir les transformations de Fourier est celui des fonctions f intégrables.
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 $\mathbb{R}$.

8.1.2 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 $\forall\omega \in \mathbb{R}, \hat{f}(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(t)e^{-i\omega t}\,\mathrm{d}t$
$\hat{f}$ 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 $rad.s^{-1}$.
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 ; il est formé de traits verticaux
- Le spectre de phase : il s'agit du tracé de l'argument de $\hat{f}(\omega)$ en fonction de la pulsation ω.

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 :

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

8.1.3 Propriétés de la transformation de Fourier

- $\mathcal{F}$ est clairement linéaire.
- 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 :

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

8.1.4 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.

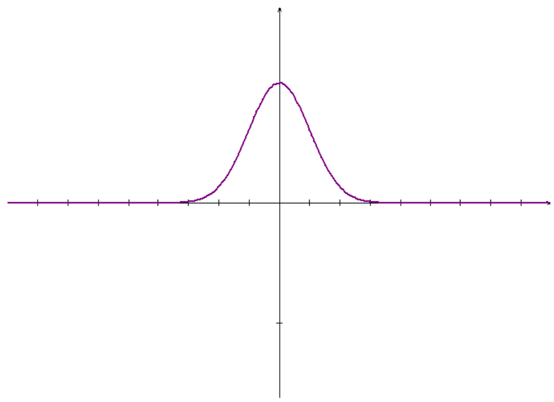

Figure 9 – Allure d'une gaussienne

8.1.5 Exemples de transformées de Fourier

Facilement, on peut montrer que la transformée de Fourier d'une gaussienne [1] 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 $\forall t \in \mathbb{R}\backslash \left[-\frac{1}{2}, \frac{1}{2}\right], \Pi(t) = 0$, on obtient directement par intégration de l'exponentielle complexe et en tenant compte de la relation $\sin x = \dfrac{e^{ix} - e^{-ix}}{2i}$, $\mathcal{F}(\Pi)(\omega) = \mathrm{sinc}(\frac{\omega}{2})$. [2]
La fonction étant réelle, son spectre de phase correspond à la fonction nulle et son spectre d'amplitude est le suivant :

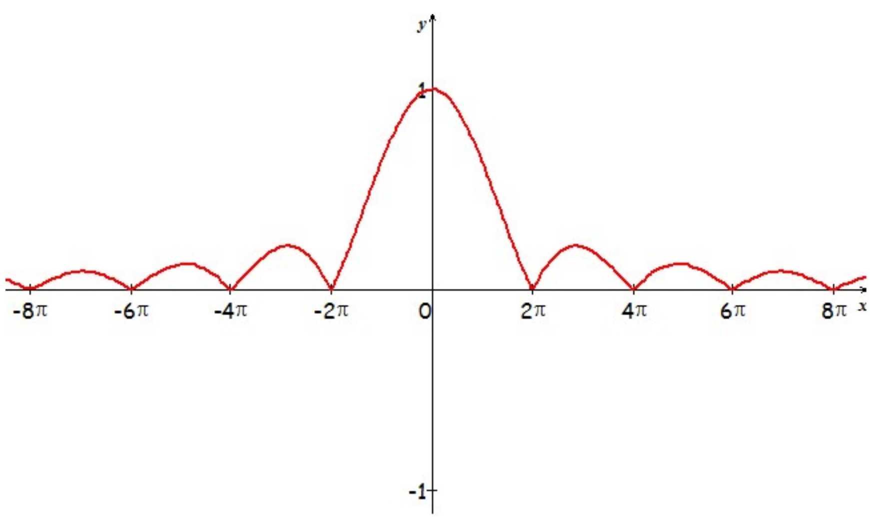

Figure 10 – Spectre d'amplitude de la fonction Π

8.1.6 Applications.

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

1. une fonction en $e^{\frac{-x^2}{2}}$.
2. la fonction sinc (sinus cardinal) est au premier sens mathématique la fonction définie sur $\mathbb{R}^*$ par $\mathrm{sinc}(x) = \frac{\sin x}{x}$.

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 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.
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.

8.2 Transformation par les ondelettes.

8.2.1 Exemples d'ondelettes mères :

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

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

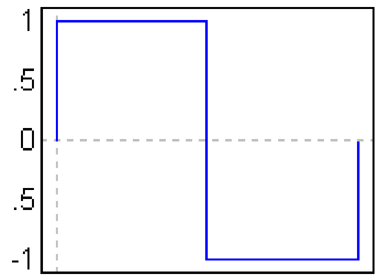

Figure 11 – Ondelette de Haar

L'ondelette "chapeau mexicain" : On peut définir cette fonction par
$$\psi : \begin{array}{ccl} \mathbb{R} & \longrightarrow & \mathbb{R} \\ t & \longmapsto & \frac{2}{\sqrt{3}}\pi^{-\frac{1}{4}}(1-t^2)e^{-\frac{t^2}{2}} \end{array}$$

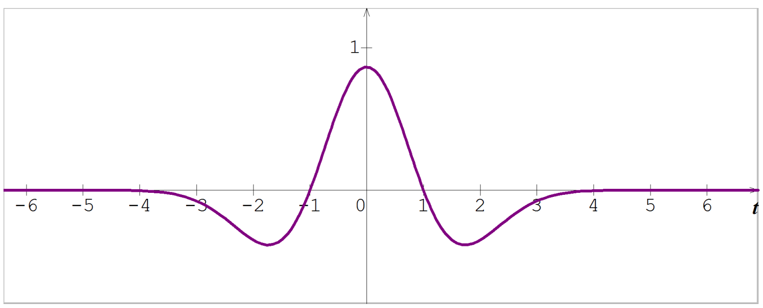

Figure 12 – Ondelette "chapeau mexicain"

L'ondelette de Morlet : On peut définir cette fonction par
$$\psi : \begin{array}{ccl} \mathbb{R} & \longrightarrow & \mathbb{R} \\ t & \longmapsto & \cos(5t)e^{-\frac{t^2}{2}} \end{array}$$

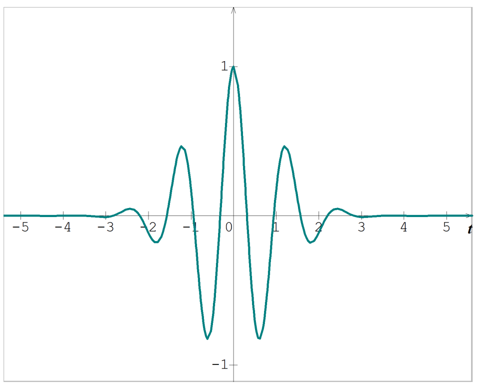

Figure 13 – Ondelette de Morlet

8.2.2 Comparaisons de méthodes de compression

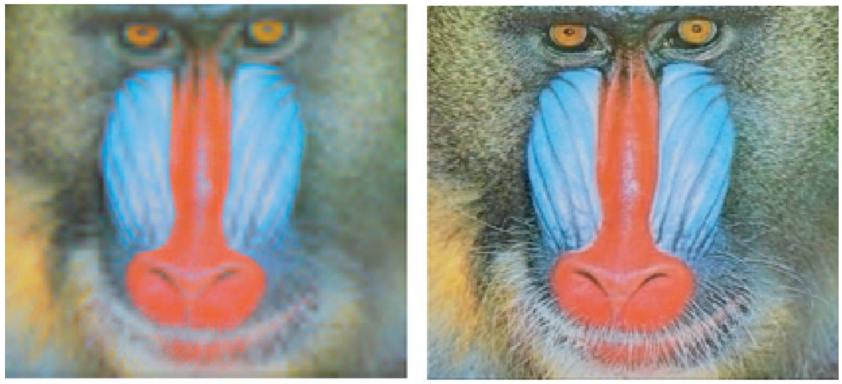

Figure 14 – Comparaison des méthodes de compression sur le Mandrill

La figure 14 (Mandrill) 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).

8.2.3 L'algorithme de Mallat : démonstration dans 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.

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 $\mathbb{R} \setminus [0,1]$.

$\mathcal{F}(\mathbb{R},\mathbb{R})$, 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 $\mathcal{F}(\mathbb{R},\mathbb{R})$.

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 $\mathcal{F}(\mathbb{R},\mathbb{R})$.

À partir de la fonction de Haar $\mathcal{H}$, on définit la fonction $H_{p,k}$ par $H_{p,k}(x) = \mathcal{H}(2^p x - k)$ avec $(p,k) \in \mathbb{N}^2$.
[*Pour alléger l'écriture et les calculs, on peut comme ici choisir d'omettre le facteur* $2^{\frac{p}{2}}$ devant $\mathcal{H}(2^p x - k)$.]

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 $\mathcal{F}(\mathbb{R},\mathbb{R})$ muni du produit scalaire canonique réel, on a pour $k \neq k'$:
$\langle h_{p,k}|h_{p,k'}\rangle = \int_0^1 h_{p,k}(x)h_{p,k'}(x).dx = \int_0^1 0.dx = 0$

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.

Décomposition orthogonale en somme directe. On peut donc prendre $\mathbb{S}_p$ le supplémentaire orthogonal de $\mathbb{K}_p$ dans $\mathbb{K}_{p+1}$. On a ainsi $\mathbb{K}_{p+1} = \mathbb{S}_p \oplus \mathbb{K}_p$.
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 $(h_{p,0}, h_{p,1}, ..., h_{p,2^p-1}, H_{p,0}, H_{p,1}, ..., H_{p,2^p-1})$ forme une base de $\mathbb{K}_{p+1}$.
De plus, on a pour $k \neq k'$:

$$\begin{aligned} \langle h_{p,k}|H_{p,k'}\rangle &= \int_0^1 h_{p,k}(x) \cdot H_{p,k'}(x) \cdot dx \\ &= 0 \text{ facilement par découpage avec Chasles} \end{aligned}$$

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 $\mathbb{K}_{p+1}$.

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

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 $(h_{p,0}, h_{p,1}, ..., h_{p,2^p-1}, H_{p,0}, H_{p,1}, ..., H_{p,2^p-1})$ 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 calcule :
$\langle H_{p,k}|H_{p,k}\rangle = \int_0^1 (H_{p,k}(x))^2.dx = \int_{x_k}^{\frac{x_k+x_{k+1}}{2}} 1^2.dx + \int_{\frac{x_k+x_{k+1}}{2}}^{x_{k+1}} (-1)^2.dx = \frac{1}{2^p}$
$\langle h_{p,k}|h_{p,k}\rangle = \int_0^1 h_{p,k}(x)^2.dx = \int_{x_k}^{x_k+1} 1^2.dx = \frac{1}{2^p}$

$$\langle h_{p,k}|H_{p,k}\rangle = \int_{x_k}^{\frac{x_k+x_{k+1}}{2}} 1.1.dx + \int_{\frac{x_k+x_{k+1}}{2}}^{x_{k+1}} 1.(-1).dx = 0$$

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

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 :

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

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 :

–

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

–

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

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

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

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 :

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

8.2.4 Schématisation de l'algorithme de Mallat

Pour la compression d'un signal Ψ_p par des ondelettes, on obtient le schéma suivant :
$$\begin{array}{cccc} & & \text{Etape 1} & \\ \Psi_p & \rightarrow & \Psi_{p-1} & \\ & \searrow & d_{p-1} & \text{(détails)} \end{array}$$

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

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

8.2.5 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 $2^n \times 2^n$ 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).

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

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

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

On obtient

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

On obtient alors en première moitié verticale de la matrice le nouvel échantillon principal et en seconde moitié les coefficients 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 $M_{p-1}, M_{p-2}, ...M_1, M_0$ avec la relation de récurrence $M_{k-1} = M_k \times H_p(k \in \{p, p-1, ..., 1\}$ et H_p désigne la matrice carrée $2^p \times 2^p$ spécifique à l'algorithme de Haar, choisie de telle sorte que son nombre p de colonnes et de lignes soit celui des colonnes et lignes de la matrice initiale de pixels).

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

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 $M'_1, M'_2, ..., M'_p$ par la relation de récurrence suivante :

$M'_{k+1} = M'_k \times (H_p)^{-1}$, avec $k \in \{0, 1, \ldots, p-1\}$, $(H_p)^{-1}$ 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 est quelque peu différente de la matrice M_p puisque certains coefficients sont devenus des 0.

8.2.6 Exemples d'images traitées avec notre algorithme

La figure 15 montre l'image à laquelle nous avons appliqué la compression. La figure 16 montre un détail de l'image.

FIGURE 15 – L'image de départ

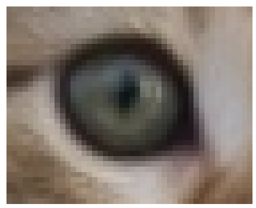

FIGURE 16 – Détail de l'image de départ

La figure 17 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. Cela montre que la compression par ondelettes est plutôt efficace et que notre algorithme est fonctionnel.

FIGURE 17 – Détail de l'image compressée

Sur la figure 18, 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 18 – Détail de l'image compressée à 100%

8.3 Code de l'algorithme en Python et application graphique

Tout le code de ce projet est sous licence Creatice Commons BY-SA (voir p.30).

8.3.1 Fichier ondelettes.py

```
import Image, math
import time
from threading import Thread
from multiprocessing import Process
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 haarimage(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()


    def haardetails(self):


        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 haarmulti(self):
        t1 = Thread(target=self.haarimage, args=())
        t2 = Thread(target=self.haardetails, args=())
```

```


        t1.start()
        t2.start()
        t1.join()
        t2.join()


    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):
            start = time.time()
            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)]
                #print ondlbas
                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 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 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 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])
            end = time.time()
            print end - start




    def fasthaar_thread(self,epsilon):
        t1 = Process(target=self.fasthaar, args=(epsilon,0,self.sizex/2,0,self.sizey))
        t2 = Process(target=self.fasthaar, args=(epsilon,self.sizex/2,self.sizex,0,self.sizey))


        t1.start()
```

```
        t2.start()
        t1.join()
        t2.join()


def main():


    image = MatriceImage("piano_bleu.jpg")
    start = time.time()
    image.getmatrixblue()
    image.getmatrixgreen()
    image.getmatrixred()
    image.create_coef_matrix()


    #image.grayscalemeanmatrix()
    #image.makeimagegray(image.matrixgray).save("piano_gris.jpg")
    #start = time.time()
    image.haar()
    #end = time.time()


    #image.makeimage().save("1px.jpg",'JPEG', quality = 100)
    image.compression(10)
    image.clearimage()
    image.syntheselignes()
    image.synthesecolonnes()
    #start = time.time()
    #image.fasthaar(15,0,image.sizex,0,image.sizey)


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


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

8.3.2 Fichier ondelettesGUI.py

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




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()

class Appli(Frame):

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

        self.parent = parent
        self.initUI()

    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)
        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.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.syntheselignes()
        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()


```

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


    def onExit(self):
        self.quit()




def main():


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




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

9 Bibliographie, Liens et Remerciements

– http ://www.cmi.univ-mrs.fr/~melot/Master2/TPsignal_PS.html
– http ://www.math-info.univ-paris5.fr/ gk/MasterTunis/cours_ond.pdf
– http ://math.univ-lyon1.fr/ gannaz/2010-01-19_ondelettes.pdf
– http ://math.umons.ac.be/preprints/src/LucasOndelettes.pdf
– La licence Creative Commons BY-SA :
http ://creativecommons.org/licenses/by-sa/3.0/
– Remerciements : Je tiens à remercier mon professeur de mathématiques de spéciale, M. Patrick Génaux, qui m'a soutenu et encadré tout au long de l'année dans la préparation de mon travail mais également mon professeur de physique de première année, M. Jérôme Petitjean, qui m'a suggéré le thème de la transformation des signaux.