

## TP n°4 - Compression d'images

L'objectif de ce TP est de mettre en oeuvre une compression d'images via une transformée en ondelettes (Haar). Les fichiers (matlab/python) nécessaires à ce TP sont disponibles à l'adresse <http://www.di.ens.fr/~andreux/teaching/TP4.zip>.

Le code ainsi que les réponses aux questions des parties 4 et 5 sont à envoyer à [mathieu.andreux@ens.fr](mailto:mathieu.andreux@ens.fr) pour le mardi **16 mai 2017** au plus tard.

### 1. Ondelettes de Haar

On suppose qu'un entier  $N$  est fixé et qu'il est de la forme  $N = 2^J$ , pour un certain  $J \in \mathbb{N}^*$ . Pour tout  $j \in \{1, \dots, J\}$  et pour tout  $n \in \{0, \dots, 2^{J-j} - 1\}$ , on définit des signaux  $\phi_{j,n}[k]$  et  $\psi_{j,n}[k]$  avec  $0 \leq k < N = 2^J$  par :

$$\phi_{j,n}[k] = \begin{cases} 2^{-j/2} & \text{si } 2^j n \leq k < 2^j(n+1), \\ 0 & \text{sinon.} \end{cases}$$

$$\psi_{j,n}[k] = \begin{cases} 2^{-j/2} & \text{si } 2^j n \leq k < 2^j n + 2^{j-1}, \\ -2^{-j/2} & \text{si } 2^j n + 2^{j-1} \leq k < 2^j(n+1), \\ 0 & \text{sinon.} \end{cases}$$

a) Calculer, pour tous  $j_1, j_2, n_1, n_2$  tels que  $j_1 \leq j_2$ ,  $\langle \psi_{j_1, n_1}, \psi_{j_2, n_2} \rangle$  et  $\langle \psi_{j_1, n_1}, \phi_{j_2, n_2} \rangle$ . Calculer également  $\langle \phi_{j, n_1}, \phi_{j, n_2} \rangle$  pour tous  $j$  et  $n_1, n_2$ .

On définit maintenant, pour tout  $j \in \{1, \dots, J\}$  et tous  $n_1, n_2 \in \{0, \dots, 2^{J-j} - 1\}$ , des signaux bidimensionnels  $\psi_{j, n_1, n_2}^1[k_1, k_2]$ ,  $\psi_{j, n_1, n_2}^2[k_1, k_2]$ ,  $\psi_{j, n_1, n_2}^3[k_1, k_2]$  avec  $0 \leq k_1, k_2 < N$  :

$$\begin{aligned} \psi_{j, n_1, n_2}^1[k_1, k_2] &= \phi_{j, n_1}[k_1] \psi_{j, n_2}[k_2] \\ \psi_{j, n_1, n_2}^2[k_1, k_2] &= \psi_{j, n_1}[k_1] \phi_{j, n_2}[k_2] \\ \psi_{j, n_1, n_2}^3[k_1, k_2] &= \psi_{j, n_1}[k_1] \psi_{j, n_2}[k_2] \end{aligned}$$

On définit également :

$$\phi[k_1, k_2] = \phi_{J,0}[k_1] \phi_{J,0}[k_2] = 2^{-J}$$

b) Montrer que la famille  $\mathcal{H} = \{\phi\} \cup \{\psi_{j, n_1, n_2}^1, \psi_{j, n_1, n_2}^2, \psi_{j, n_1, n_2}^3\}_{0 < j \leq J, 0 \leq n_1, n_2 < 2^{J-j}}$  est une base orthonormée de  $\mathbb{R}^{N \times N}$ .

### 2. Transformée en ondelettes de Haar

Puisque l'ensemble  $\mathcal{H}$  considéré à la question précédente est une base orthonormée, tout signal  $f \in \mathbb{R}^{N \times N}$  peut s'écrire sous la forme :

$$f = a_\phi[f] \phi + \sum_{j=1}^J \sum_{s=1,2,3} \sum_{n_1, n_2} a_{s,j,n_1,n_2}[f] \psi_{j,n_1,n_2}^s$$

où  $a_\phi[f] = \langle f, \phi \rangle$  et  $a_{s,j,n_1,n_2}[f] = \langle f, \psi_{j,n_1,n_2}^s \rangle$ .

Dans les questions suivantes, on construit un algorithme qui calcule les coefficients  $a$ .

a) Montrer que, pour tous  $n_1, n_2 \in \{0, \dots, 2^J - 1\}$  :

$$\begin{aligned} a_{1,1,n_1,n_2}[f] &= \frac{1}{2} (f[2n_1, 2n_2] + f[2n_1 + 1, 2n_2] - f[2n_1, 2n_2 + 1] - f[2n_1 + 1, 2n_2 + 1]) \\ a_{2,1,n_1,n_2}[f] &= \frac{1}{2} (f[2n_1, 2n_2] - f[2n_1 + 1, 2n_2] + f[2n_1, 2n_2 + 1] - f[2n_1 + 1, 2n_2 + 1]) \\ a_{3,1,n_1,n_2}[f] &= \frac{1}{2} (f[2n_1, 2n_2] - f[2n_1 + 1, 2n_2] - f[2n_1, 2n_2 + 1] + f[2n_1 + 1, 2n_2 + 1]) \end{aligned}$$

b) Écrire une fonction **last\_scale**, qui prend en argument un signal  $f$ , de taille  $N \times N$ , où  $N$  est de la forme  $2^J$  avec  $J \geq 1$ , et renvoie les  $a_{s,1,n_1,n_2}[f]$ .

c) On définit :

$$g = a_\phi[f]\phi + \sum_{j=2}^J \sum_{s=1,2,3} \sum_{n_1,n_2} a_{s,j,n_1,n_2}[f] \psi_{j,n_1,n_2}^s$$

Attention à l'indexation sur  $j$ .

Montrer que  $g[2k_1, 2k_2] = g[2k_1, 2k_2 + 1] = g[2k_1 + 1, 2k_2] = g[2k_1 + 1, 2k_2 + 1]$  pour tous  $0 \leq k_1, k_2 < 2^{J-1}$ .

d) On note  $\tilde{f}$  le signal tel que  $\tilde{f}[k_1, k_2] = 2g[2k_1, 2k_2]$ , pour  $0 \leq k_1, k_2 < 2^{J-1}$ .

Montrer que, pour tous  $k_1, k_2$  :

$$\tilde{f}[k_1, k_2] = \frac{1}{2} (f[2k_1, 2k_2] + f[2k_1 + 1, 2k_2] + f[2k_1, 2k_2 + 1] + f[2k_1 + 1, 2k_2 + 1])$$

e) Modifier la fonction **last\_scale** pour qu'elle renvoie également  $\tilde{f}$ .

f) On note  $\{\tilde{\phi}\} \cup \{\tilde{\psi}_{j,n_1,n_2}^1, \tilde{\psi}_{j,n_1,n_2}^2, \tilde{\psi}_{j,n_1,n_2}^3\}_{0 \leq j \leq J-1, 0 \leq n_1, n_2 < 2^{J-1-j}}$  les ondelettes de Haar de dimension  $2^{J-1} \times 2^{J-1}$  (c'est-à-dire définies de la même façon que dans la première section, en remplaçant  $N$  par  $\tilde{N} = 2^{J-1}$ ). Il s'agit d'une base orthonormée de  $\mathbb{R}^{2^{J-1} \times 2^{J-1}}$ .

Montrer que :

$$\tilde{f} = a_\phi[f]\tilde{\phi} + \sum_{j=1}^{J-1} \sum_{s=1,2,3} \sum_{n_1,n_2} a_{s,j+1,n_1,n_2}[f] \tilde{\psi}_{j,n_1,n_2}^s$$

g) Exprimer la transformée en ondelettes de Haar de  $\tilde{f}$  en fonction de celle de  $f$ .

h) Écrire une fonction récursive, **wavelet\_transform**, qui prend en entrée un signal  $f$  et renvoie en sortie  $a_\phi[f]$  et les  $a_{s,j,n_1,n_2}[f]$ .

[Indication : il y a plusieurs possibilités pour le format de la sortie. Un choix judicieux serait de regrouper les coefficients sous la forme d'un tableau de mêmes dimensions que l'image d'entrée, sous la forme générique suivante :

$$\begin{pmatrix} a[\tilde{f}] & a_{2,1,\dots}[f] \\ a_{1,1,\dots}[f] & a_{3,1,\dots}[f] \end{pmatrix}$$

]

### 3. Reconstruction à partir de la transformée en ondelettes

Dans ce paragraphe, on écrit l'inverse de la fonction `wavelet_transform` : on suppose les  $a_{s,j,n_1,n_2}[f]$  et  $a_\phi[f]$  connus et on souhaite reconstruire  $f$ .

a) Si  $N = 1$ , exprimer  $f$  en fonction de  $a_\phi[f]$ .

b) On suppose maintenant que  $f$  est de taille  $N = 2^J$  avec  $N \geq 1$ . On définit  $g, \tilde{f}$  comme dans la question 2. et on pose :

$$h = f - g = \sum_{s=1,2,3} \sum_{n_1,n_2} a_{s,1,n_1,n_2}[f] \psi_{1,n_1,n_2}^s$$

Montrer que, pour tous  $k_1, k_2 \in \{0, \dots, 2^{J-1} - 1\}$  :

$$\begin{aligned} h[2k_1, 2k_2] &= \frac{1}{2}(a_{1,1,k_1,k_2}[f] + a_{2,1,k_1,k_2}[f] + a_{3,1,k_1,k_2}[f]) \\ h[2k_1 + 1, 2k_2] &= \frac{1}{2}(a_{1,1,k_1,k_2}[f] - a_{2,1,k_1,k_2}[f] - a_{3,1,k_1,k_2}[f]) \\ h[2k_1, 2k_2 + 1] &= \frac{1}{2}(-a_{1,1,k_1,k_2}[f] + a_{2,1,k_1,k_2}[f] - a_{3,1,k_1,k_2}[f]) \\ h[2k_1 + 1, 2k_2 + 1] &= \frac{1}{2}(-a_{1,1,k_1,k_2}[f] - a_{2,1,k_1,k_2}[f] + a_{3,1,k_1,k_2}[f]) \end{aligned}$$

c) Ecrire une fonction `reconstruct_high_freq` prenant en arguments des matrices contenant  $\{a_{1,1,k_1,k_2}[f]\}_{k_1,k_2}$ ,  $\{a_{2,1,k_1,k_2}[f]\}_{k_1,k_2}$  et  $\{a_{3,1,k_1,k_2}[f]\}_{k_1,k_2}$  et renvoyant la matrice  $h$  correspondante.

d) Ecrire une fonction `oversample` prenant en entrée un signal  $\tilde{f}$  et renvoyant le signal  $g$  correspondant (avec les notations de la partie 2).

[Indication : A ce stade, on pourra tester les deux fonctions précédentes sur la sortie de la fonction `last_scale`.]

e) Écrire une fonction récursive `inverse_wavelet_transform`, qui prend en entrée des coefficients  $a_{s,j,n_1,n_2}[f]$  et  $a_\phi[f]$  et renvoie  $f$ .

f) Tester les fonctions `wavelet_transform` et `inverse_wavelet_transform` : générer un signal  $f$  aléatoire de taille  $N \times N$  avec  $N = 512$  (par exemple un bruit blanc gaussien). Calculer sa transformée en ondelettes puis la reconstruction  $f_{rec}$  de  $f$  à partir de la transformée en ondelettes. Afficher la norme de  $f_{rec} - f$  :

$$\sqrt{\sum_{k_1,k_2} |f_{rec}[k_1, k_2] - f[k_1, k_2]|^2}$$

[Pour un bruit blanc gaussien de variance 1, vous devriez trouver des valeurs de l'ordre de  $10^{-13}$ . Si vous trouvez une valeur significativement supérieure, il y a une erreur quelque part.]

### 4. Compression d'images

Une fonction d'encodage et une fonction de décodage sont disponibles à l'adresse <http://www.di.ens.fr/~andreux/teaching/TP4.zip>, ainsi qu'une image de test. Des instructions de chargement de cette image sont données dans la partie 5.

La fonction **encode** prend en entrée une suite d'entiers (dont certains peuvent éventuellement être négatifs) et renvoie un code associé à cette suite, sous la forme d'une chaîne de caractères ne contenant que des 0 et des 1.

La fonction **decode** prend en entrée un code calculé par la fonction **encode** et renvoie la suite d'entiers dont provient le code. Sur de longues entrées, elle peut avoir un temps d'exécution assez long ( $\sim$  trente secondes).

L'algorithme d'encodage utilisé est l'algorithme de compression de Huffman.

Dans cette partie, on définit une fonction **compress**, qui prend en entrée un paramètre  $\delta > 0$  et une image **im**, de taille  $2^J \times 2^J$  pour un certain  $J \in \mathbb{N}$ .

- a) Calculer la transformée en ondelettes de Haar de **im**.
- b) Concaténer les  $2^{2J}$  coefficients de la transformée en un vecteur **v** à une seule dimension.
- c) Calculer **v\_quant**, le vecteur contenant les coefficients de **v** arrondis au multiple de  $\delta$  le plus proche.

d) Calculer **im\_comp**, le code associé à **v\_quant**/ $\delta$  par la fonction **encode**.

e) On souhaite que **compress** renvoie également l'entropie de la suite **v\_quant**/ $\delta$ .

Calculer un tableau qui contient le nombre d'occurrences de chaque entier dans **v\_quant**/ $\delta$ .

[Indication : vous pouvez utiliser la commande `n_occ = hist(v_quant/delta,xbins)`, pour un choix approprié de **xbins**.]

En déduire l'entropie **H** et la faire renvoyer à **compress** avec **im\_comp**.

## 5. Décompression

a) Écrire une fonction **decompress** qui prend en entrée un  $\delta > 0$  et une chaîne de caractères **im\_comp** (calculée par la fonction **compress**) et renvoie une approximation **im\_rec** de l'image dont provient **im\_comp**.

b) Charger l'image **batiment.png** dans une variable **im**. La convertir en un tableau de flottants à deux dimensions, de valeurs comprises entre 0 et 255, à l'aide des instructions suivantes : `imread('batiment.png'); im = sum(im,3); im = im/max(im(:))*255;`

c) Afficher sur trois figures différentes l'image **im** et les deux **im\_rec** obtenues pour  $\delta = 100$  et  $\delta = 30$ .

Utiliser `imagesc(image,[0,255])` et `colormap('gray')` pour avoir la bonne échelle de couleurs et `axis equal` pour éviter que les images ne soient déformées.

d) Effectuer le calcul suivant pour  $\delta = 2, 5, 10, 20, 50, 100, 200, 500$  :

- Calculer la chaîne **im\_comp** et l'entropie **H** renvoyées par **compress**.
- Calculer le nombre moyen de bits par pixel dans **im\_comp**, c'est-à-dire :

$$R = \frac{\text{nombre de caractères de } \mathbf{im\_comp}}{\text{nombre de pixels de } \mathbf{im}}$$

- À partir de **im\_comp**, calculer **im\_rec** et en déduire l'erreur quadratique sur **im**, c'est-à-dire :

$$D = \sum_{k_1, k_2} |\mathbf{im\_rec}[k_1, k_2] - \mathbf{im}[k_1, k_2]|^2$$

e) Afficher sur une quatrième figure les graphes de  $\log_2(D)$  en fonction de  $R$  et de  $\log_2(D)$  en fonction de  $H$ .

D'où vient la différence entre les deux courbes que vous venez de tracer ?

f) Vous devez observer que la variation de  $\log_2(D)$  en fonction de  $H$  est à peu près affine pour les plus petites valeurs de  $D$ . À quel résultat du cours est-ce dû ? Pourquoi ce résultat ne s'applique-t-il plus pour de grandes valeurs de  $D$  ?

g) À partir de quelle valeur de  $R$  la différence entre `im` et `im_rec` devient-elle difficilement visible ?

h) Si on utilise la fonction `encode` directement sur l'image `im`, combien de bits obtient-on en moyenne par pixel ?