Rapport de TP - SY26

TP03 - Quantification scalaire

Rémi Burtin

Cyril Fougeray

24 avril 2014

Université de Technologie de Compiègne

1 Introduction

L'objectif de ce TP est de programmer l'algorithme itératif de Lloyd-Max pour la conception d'un quantificateur scalaire optimal.

2 Quantification uniforme

2.1 Distorsion normalisée et rapport signal bruit

Ce premier script nous permet de calculer la distorsion normalisée (NMSE) ainsi que le rapport signal/bruit (Signal to Noise Ratio - SNR).

Pour cela, nous utilisons la fonction *lloyds* qui permet d'encoder un vecteur (ici une image en vecteur ligne) selon Lloyd-Max avec un nombre précis de partitions (correspondant aux nombre de valeurs de reconstruction). Cette fonction *lloyds* retourne la distorsion (non normalisée). Afin de calculer la distorsion normalisée, nous devons diviser cette distorsion D par la variance du vecteur passé à la fonction :

$$NMSE = \frac{D}{\sigma_X^2}$$

2.1.1 Résultats

Après avoir exécuté la fonction, nous nous rendons compte des résultats :

```
>> [nmse snr] = disto_1_1('lena.bmp', 2)
nmse = 0.3009
snr = 5.2157

>> [nmse snr] = disto_1_1('lena.bmp', 8)
nmse = 0.0190
snr = 17.2149

>> [nmse snr] = disto_1_1('lena.bmp', 256)
nmse = 0
snr = Inf
```

Lors de la première exécution, nous choisissons d'encoder l'image avec 2 valeurs différentes (correspondant à 2 niveaux de gris dans notre cas). La distorsion est donc importante, et maximale (il n'est pas possible et inutile d'encoder l'image avec 1 seule valeur car aucune information n'est conservée dans ce cas).

Évidemment, dans le cas où nous encodons l'image avec 256 niveaux de gris (telle que l'image originale), aucune distorsion n'est présente.

2.2 Quantification uniforme

Nous allons maintenant programmer une fonction permettant la quantification uniforme d'une image à partir d'un débit, correspondant aux nombre valeurs de reconstruction. Cette fonction renvoie l'image quantifiée ainsi que la distorsion avec l'image originale.

L'image originale utilisée sera ici en niveau de gris. Si l'image passée en paramètre de la fonction est une image en couleur, elle sera alors convertie via la fonction *rgb2gray*. Le signal X obtenu est alors borné entre la valeur minimum 0 et la valeur maximum 255 (domaine de variation).

Nous allons utiliser la fonction *quantiz* afin de quantifier uniformément l'image. Cette fonction permet de choisir les seuils de décision ainsi que les valeurs de reconstruction pour un signal donné.

Pour quantifier uniformément le signal, nous allons construire N niveaux de reconstruction ce qui nous permettra de définir nos seuils de décision. N est ici fonction du débit binaire R passé en paramètre de la fonction de quantification : $N = 2^R$. Ainsi, nous devons construire $N+1$ seuils de décision. Afin de partitionner uniformément le domaine de variation, nous utilisons la fonction *linspace* qui permet de créer un vecteur ayant des valeurs linéairement réparties entre une borne inférieure et une borne supérieure. La fonction *quantiz* nous impose la suppression de ces bornes dans le vecteur des seuils de décision, comme énoncé dans la documentation de la fonction :

```
 Elements of INDX = 0, 1, 2, ..., N-1 represent SIG in the range of
[-Inf, PARTITION(1)], [PARTITION(1), PARTITION(2)], [PARTITION(2),
PARTITION(3)], ..., [PARTITION(N-1), Inf].
```

Ainsi, les deux bornes extrêmes seront automatiquement $-\infty$ et $+\infty$. La fonction *linspace* a inclus les bornes dans le vecteur, nous devons donc supprimer les valeurs 0 et 255 de notre vecteur `PARTITION`.

Nous devons maintenant choisir nos valeurs de reconstruction $r_i, i \in [1, N]$. Lors d'une quantification uniforme, ces valeurs de reconstruction sont au milieu des seuils de décisions, soit la moyenne des seuils de décisions consécutifs : $r_i = (d_i + d_{i+1})/2$. Une boucle permet de construire un vecteur contenant les valeurs de reconstruction à partir du vecteur des seuils de reconstruction (*cf* Annexe A.2). L'image quantifiée est donc retournée par la fonction *quantiz*, nous la reconstruisons ensuite afin d'obtenir une matrice, telle que l'image originale. Afin d'afficher l'image via la fonction *imshow*, il faut diviser les valeurs de la matrice par 255.

Nous exécutons la fonction dans MATLAB :

```
[image_quant dist] = quant_uni_1_2('lena.bmp', 2);
partition =
   63.7500  127.5000  191.2500
codebook =
   31.8750   95.6250  159.3750  223.1250
>> imshow(image_quant./255)
>> dist
dist = 331.5913
```

FIGURE 1 – Quantification uniforme, lena.bmp, débit = 2

Les valeurs de reconstruction sont au milieu de chaque intervalle. Ces intervalles sont répartis entre 0 et 255 ce qui implique qu'il est impossible de retrouver les valeurs de reconstruction 0 ou 255. La distorsion est donc dans tous les cas non nulle dans notre cas.

2.3 Tests sur les différentes images

	4 niveaux	8 niveaux	64 niveaux
Lena	331.59	83.38	1.28
Peppers	309.10	86.97	1.29
Harbour	282.88	113.16	1.21
Bridge	331.21	84.13	0.87
Boats	226.24	92.48	1.28
Airfield	405.28	92.13	1.37

TABLE 1 – Comparaison des distorsions

2.4 Huffman - Taux de compression des images quantifiées

	Uniforme 4 niveaux	Uniforme 16 niveaux	Sans
Lena	77.08 %	55.66 %	6.66 %
Peppers	77.05 %	53.82 %	4.77 %
Harbour	79.56 %	59.86 %	15.21 %
Bridge	77.27 %	53.14 %	28.35 %
Boats	80.6 %	60.80 %	11.74 %
Airfield	75.31 %	52.46 %	10.70 %

TABLE 2 – Taux de compression avec Huffman

3 Quantification scalaire optimale de Lloyd-Max

3.1 Mise en oeuvre

La programmation de la fonction de quantification scalaire optimale de Lloyd-Max se fait à l'aide de deux paramètres : le signal (dans notre cas une image), et le débit disponible (bits/pixel). Cette fonction renvoie l'image quantifiée.

Pour initialiser l'algorithme, nous utilisons une simple quantification uniforme. Ensuite, nous allons chercher à minimiser l'erreur quadratique à l'aide d'une boucle itérative. A chaque itération, il faut que la valeur de reconstruction se rapproche de la moyenne des échantillons dans un intervalle donné : application de la condition du centroïde. A noter que, comme indiqué dans l'énoncé, lorsque l'intervalle est vide, la valeur de reconstruction devient le centre de l'intervalle. Ensuite, nous modifions l'intervalle (valeurs de décision) en appliquant la règle du plus proche voisin. L'image est quantifiée à l'aide des valeurs de reconstruction et des seuils de décisions via la fonction *quantiz* (vue dans la partie précédente). Sa distorsion est recalculée et comparer à la précédente à chaque itération ce qui permet de sortir de la boucle lorsque cette distorsion ne fluctue plus beaucoup. La condition d'arrêt que nous avons fixé est une différence de distorsion inférieur à 0.1. Lorsque la distorsion se stabilise, nous reconstruisons l'image afin de pouvoir l'afficher ensuite. L'ensemble de l'algorithme est disponible en annexe A.3

FIGURE 2 – Quantification scalaire optimale de Lloyd-Max, lena.bmp, débit = 2

Contrairement à la quantification uniforme, il est possible via cet algorithme de retrouver une distorsion nulle. En effet, les valeurs de reconstruction (niveaux de gris) retrouvées pour un débit égal à 8 sont exactement les mêmes valeurs que pour l'image originale (résultat que nous observons paragraphe 2.1).

3.2 Tests sur les différentes images

FIGURE 3 – Comparaison visuelle : (de gauche à droite) image originale en niveaux de gris, image quantifiée uniformément, image quantifiée optimisée selon Lloyd-Max (débit = 4)

Nous observons visuellement les différences de qualité entre les deux quantifications mises en oeuvre dans ce TP (avec le même débit). La quantification uniforme apporte un contraste plus important qui vient biaiser les couleurs originales. La quantification scalaire optimale permet de retrouver des couleurs plus fidèles à l'image originale, avec des niveaux de gris moins saillants. En effet, les couleurs retrouvent plus facilement leur appartenance dans les classes qui ont été optimisées selon la densité, ce qui permet de retrouver des couleurs plus fidèles. Mathématiquement, cette qualité est démontrée par la minimisation de l'erreur quadratique que permet l'algorithme de Lloyd-Max :

	Lloyd-Max	Uniforme
Lena	162.73	331.59
Peppers	199.19	309.10
Harbour	152.30	282.88
Bridge	274.54	331.21
Boats	134.59	226.24
Airfield	285.65	405.28

TABLE 3 – Comparaison des distorsions - 4 niveaux

Des écarts importants sont observés entre les distorsions des deux algorithmes mis en œuvre, pouvant aller du simple au double lorsque les niveaux de reconstruction de la quantification uniforme sont très mal choisis.

3.3 Huffman - Taux de compression des images quantifiées

Ci-dessous, un tableau récapitulatif des différents taux de compression des images quantifiées puis compressées en utilisant l'algorithme de Huffman.

	Avec Lloyd-Max 4 niveaux	Avec Lloyd-Max 16 niveaux	Sans
Lena	75 %	52.36 %	6.66 %
Peppers	75 %	51.36 %	4.77 %
Harbour	78.15 %	58.81 %	15.21 %
Bridge	75.11 %	53.14 %	28.35 %
Boats	76 %	54.04 %	11.74 %
Airfield	75.97 %	52 %	10.70 %

TABLE 4 – Taux de compression avec Huffman

Évidemment, la compression d'images ayant peu de niveaux de gris est plus efficace, cependant, elle se fait au dépend de la qualité de l'image.

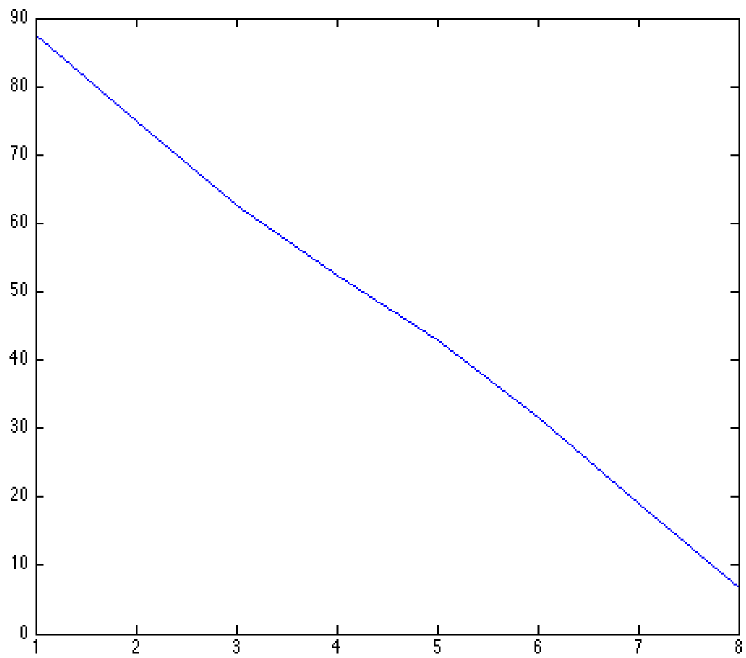

FIGURE 4 – Taux de compression de lena.bmp en fonction du nombre de bits

4 Conclusion

Ce TP nous a permis de se rendre compte de ce qu'implique la quantification. En effet, la distorsion du signal peut-être très importante et la qualité du signal est ainsi dégradée sévèrement. Cependant, nous avons vu qu'avec le même nombre de bits par pixel, il est possible de réduire significativement la distorsion, en appliquant une quantification intelligente qui permet de retrouver des niveaux de reconstruction en moyenne plus proches des valeurs réelles.

A Codes source MATLAB

A.1 NMSE et SNR

```
function [nmse snr] = disto(image, nbval)
    img = imread(image);

    % Conversion image en gris
    composantes = size(img, 3);
    if (composantes == 3)
        img = rgb2gray(img);
    end

    % Conversion image en liste
    dim = size(img,1)*size(img,2);
    list=reshape(double(img), 1, dim);

    % Codage arithmetique via quantification de Lloyd-Max
    [part, codebook, dist] = lloyds(list, nbval);

    % Distorsion normalisee
    nmse = dist/var(list);
    % Rapport signal bruit
    snr = -10*log10(nmse);

    return;
end
```

A.2 Quantification uniforme

```
function [image_quant disto] = quant_uni(image, debit)
    img = imread(image);
    % Conversion en gris si couleur
    composantes = size(img, 3);
    if (composantes == 3)
        img = rgb2gray(img);
    end
    % Conversion image en vecteur ligne
    dim = size(img,1)*size(img,2);
    list=reshape(double(img), 1, dim);

    % Construction des partitions
    partition = linspace(0,255, 2^debit+1);

    % Creation des valeurs de reconstruction
    for i=1:(2^debit),
        codebook(i) = (partition(i) + partition(i+1)) / 2 ;
    end

    % Quantiz utilise -Inf et +Inf pour borner la partition.
    partition = partition(2:2^debit);

    % Quantification
    [~, image_quant] = quantiz(list, partition, codebook);

    %partition
    %codebook

    % Calcul de la distorsion
    disto = mean((list-image_quant).^2);

    % Reconstruction de l'image en vecteur
    image_quant = reshape(image_quant, size(img,1), size(img,2));
end
```

A.3 Quantification scalaire optimale de Lloyd-Max

```
function [image_quant disto] = quant_lloyd(image, debit)
    img = imread(image);
    % Conversion en gris si couleur
    composantes = size(img, 3);
    if (composantes == 3)
        img = rgb2gray(img);
    end
    % Conversion image en vecteur ligne
    dim = size(img,1)*size(img,2);
    list=reshape(double(img), 1, dim);

    % Construction des partitions
    partition = linspace(0,255, 2^debit+1);
    % Creation des valeurs de reconstruction
    for i=1:(2^debit),
        codebook(i) = (partition(i) + partition(i+1)) / 2 ;
    end

    % Initialisation condition d'arret
    oldDisto = 9999; % doit etre ¿¿ a la premiere disto calculee
    diffDisto = 9999; % Doit etre ¿ a 0.1 pour entrer dans la boucle

    % Condition de convergence
    while diffDisto > 0,1;

        for j=1:(2^debit) %calcul des ri
            % v = index des elements appartenant a l'intervalle j dans list
            v = find(list >= partition(j) & list <= partition(j+1));
            if (isempty(v)) % intervalle vide
                codebook(j) = (partition(j) + partition(j + 1)) / 2;
            else
                codebook(j) = mean(list(v));
            end
        end

        for j=2:(2^debit) %calcul des di
            partition(j) = (codebook(j) + codebook(j - 1))/2;
        end

        % Quantification
        [~, image_quant] = quantiz(list, partition(2:2^debit), codebook);
        % Calcul de la distorsion
        disto = mean((list-image_quant).^2);

        %Mise a jour condition d'arret
        diffDisto = oldDisto - disto;
        oldDisto = disto;
    end

    % Reconstruction de l'image en vecteur
    image_quant = reshape(image_quant, size(img,1), size(img,2));
end
```