Institut Galilée
Télécom 3

Travaux pratiques de traitement d'images numériques
Deuxième séance
Institut Galilée
2010-2011

G. Dauphin & A. Beghdadi

Préambule

Les programmes nécessaires pour faire ce TP sont huff2norm.m, norm2huff.m, frequency.m qui sont des fonctions réalisées par Giuseppe Ridinò (http://www.mathworks.com/matlabcentral/fileexchange/4900). Les images sont disponibles dans Matlab

A. Codage sans perte : l'algorithme de Huffman

Introduction - Rappels

Le codage de Huffman est un algorithme de compression de données sans perte élaboré par David Albert Huffman et publié en 1952. Le code est déterminé à partir d'une estimation des probabilités d'apparition des symboles de source, un code court étant associé aux symboles de source les plus fréquents. Le principe du codage de Huffman repose sur la création d'un arbre binaire, c'est-à-dire un arbre où chaque nœud a deux fils, le fils de gauche est étiqueté par un zéro et le fils de droite est étiqueté par un 1. Chaque terminaison représente un des symboles à coder. Le code associé au symbole est une succession de 0 et de 1, la succession qui correspond au parcours depuis la racine jusqu'à la terminaison correspondant au symbole. Une implémentation du code de Huffman est donnée par les programmes *huff2norm.m* et *norm2huff.m*

Cet algorithme est en un certain sens optimal c'est-à-dire que la longueur statistiquement nécessaire pour coder un des symboles est presque égale à l'entropie

$$H(X) \le L(X) \le H(X)+1 \quad (1)$$

où $H(X) = \sum p_i \log_2\left(\frac{1}{p_i}\right)$ et $L(X) = \sum_i p_i C_i$ et p_i est la probabilité du symbole i et C_i est la longueur du code associé au symbole i.

1. Application sur des données synthétiques

> En utilisant la simulation d'un ensemble de données aléatoires ayant une distribution de probabilités choisie préalablement suivant une fonction déterministe ou de façon aléatoire, montrez que le codage puis le décodage de ces données se fait sans dégradations et que la relation (1) est respectée.

Instructions Matlab à tester pour réaliser le programme :
Simulation de données aléatoires suivant une loi uniforme à valeurs dans un alphabet à 256 éléments :
>>x=ceil(255*rand(1,2000)) ;
On mesure l'histogramme des données x en le mettant sous la forme d'un vecteur *x(:)* puis en appliquant la fonction Matlab *hist*.
>>histogrammeObserve=hist(x(:),0:255);
L'axe des abscisses est l'ensemble des éléments de l'alphabet et l'axe des ordonnées est le nombre d'occurrences.
>>figure(1) ; plot(0:255,histogrammeObserve);
Choix aléatoire d'une distribution de probabilité :

>>rand(1,256) ; histogrammeDemande=ans/sum(ans) ;
Distribution de probabilité choisie à partir d'une fonction :
>>sin(pi*(1:256)/257) ; histogrammeDemande=ans/sum(ans) ;
A partir de ces histogrammes demandés on peut construire les fonctions de répartitions associées que l'on note $F_B(b) = P(B \le b)$ qui est à valeurs dans *[0,1]*:
>>figure(1) ; plot(0:255,cumsum(histogrammeDemande));
Soit U une variable aléatoire qui suit une loi uniforme sur *[0,1]*. Alors $F_B^{-1}(U)$ est une variable aléatoire qui suit la loi de B. En effet :

$$P\left(F_B^{-1}(U) \le b\right) = P\left(U \le F_B(b)\right) = F_B(b)$$

Il nous suffit donc de trouver une fonction qui inverse la fonction de répartition, c'est-à-dire une fonction qui retrouve l'indice d'un élément d'un vecteur à partir d'une valeur approchée de sa valeur et ce en supposant seulement que ce vecteur est bien ordonné. La fonction CptAsyn réalise ceci, t désigne les valeurs approchées recherchées et A désigne le vecteur bien ordonné. Dans cette longue instruction, ce qui apparaît comme des guillemets est en fait une juxtaposition de deux apostrophes et est interprété lors de l'exécution en Matlab comme une transposition de la matrice, ce qui apparaît comme une apostrophe est en fait une indication de début ou de fin de chaîne de caractères.
>>CptAsyn=inline('reshape(sum(ones(length(t(:).''),1)*(A(:).'')<=t(:)*ones(1,length(A(:))),2),size(t))','t','A');
Si t est un nombre (et non un vecteur) alors le résultat de cette fonction est le nombre d'éléments inférieurs ou égaux dans A à t. A priori A est rangé par ordre croissant, aussi le résultat est l'indice de t dans A qui donne une valeur inférieure ou égale à t. Cette instruction est alors presque synonyme à *t=2; A=[0 3]; find(t>=A,1,'last'),*
>>CptAsyn(1,[0 3])
>>CptAsyn(4,[0 3])
>>CptAsyn(-1,[0 3])
Si t est un vecteur ou une matrice alors le résultat est un vecteur ou une matrice de la même taille que t et dont les valeurs sont égales à l'application de CptAsyn sur chaque valeur de t.
>> CptAsyn([-1; 4; 1],[0 3])
La variable aléatoire peut donc être simulé par
>>x=CptAsyn(rand(1,20000),cumsum(histogrammeDemande));
On peut contrôler que la variable aléatoire simulée est bien conforme à la distribution de probabilité demandée :
>>histogrammeObserve= hist(x(:),0:255);
>>histogrammeObserve=histogrammeObserve/sum(histogrammeObserve);
>>figure(1) ; plot(0:255,histogrammeObserve,'b',0:255,histogrammeDemande,'r');
La fonction *norm2huff* fournit le code correspondent aux données dans x suppose à valeurs entières entre 0 et 255. *info* est l'ensemble des informations nécessaires pour décoder la séquence. Le code ainsi obtenu est une succession d'entiers entre 0 et 255, chacun code sur 8 bits.
>>[code,info]=norm2huff(uint8(x));
Longueur du code moyen par symbole
>>length(code)*8/length(x)
Dans la pratique on est oblige aussi de stocker *info*, mais alors le lien avec l'entropie est un peu modifié.
Pour calculer l'entropie, on peut utiliser la fonction Matlab *entropy* ou calculer l'histogramme observé *hO* et évaluer
>> H=sum(hO(hO~=0)/sum(hO).*log2(sum(hO)./hO(hO~=0)));

2. Application à des images numériques bruitées et filtrées

On se propose, dans ce qui suit, d'analyser les effets de quelques traitements sur le signal image en étudiant l'entropie associée à la distribution des niveaux de gris.

2.1. Choisir une image en niveau de gris. Cette image sera notée A. Mettre cette image au format double et à valeurs sur l'ensemble 0…255 (pour l'affichage avec *imshow*, il faudra au préalable diviser à 255).

a. Déterminez et tracez l'histogramme des niveaux de gris de cette image. En déduire l'entropie associée à cet histogramme.

b. Ajoutez un bruit blanc gaussien à l'image A. Soit B l'image ainsi obtenue. Déterminez et tracez l'histogramme de B. Déduisez l'entropie associée. Comparez à l'image A ; que peut-on en dire ?

c. On applique maintenant un filtre passe-bas à l'image A. Soit C l'image résultante. Calculez l'histogramme et l'entropie associée. Comparez aux deux cas précédents. Expliquez les résultats obtenus.
2.2. Dans chacun des trois cas (A,B et C) appliquez le code de Huffman. Comparez les taux de compression et l'efficacité.

B. Transformée en cosinus discrète sur une image entière

B1. Application de la DCT globale

La transformée en cosinus discrète est utilisée dans la compression jpeg parce qu'elle permet de concentrer l'information pertinente sur un petit nombre de coefficients.

> Choisissez une image entière en niveaux de gris, appliquez la transformée en cosinus discrète. Où se trouvent, au sens de la valeur absolue, les coefficients élevés et les coefficients faibles, si on voulait lire les coefficients du plus élevés au plus faible quel parcours parmi les coefficients faudrait-il faire ? Calculez le PSNR entre l'image originale et l'image reconstruite en ne conservant que les coefficients de la DCT dont les coordonnées vérifient i+j<=n. Représentez l'évolution du PSNR en fonction du nombre de composantes non-annulées.

Instructions Matlab à tester pour réaliser le programme :
Calcul des coefficients de la dct.

```
>>imDct=dct2(im) ;
```

On cherche à visualiser l'ordonnancement des valeurs des coefficients de la DCT. Pour cela on a recours aux instructions suivantes qui permettent d'afficher une image après égalisation d'histogramme, mais en l'occurrence il ne s'agit pas d'une image mais des composantes de la décomposition en cosinus discrète.

```
>>[val,ordre]=sort(abs(imDct(:)));
>>imDctOrdre(ordre)=1:prod(size(im));
>>imDctOrdre=reshape(imDctOrdre,size(im));
>>figure(1); imshow(imDctOrdre/max(max(imDctOrdre)));
```

Annulation des composantes dont la somme des coordonnées est supérieure à n

```
>>[J,I]=meshgrid(1:size(im,2),1:size(im,1));
>>imDctModifie=imDct;
>>imDctModifie(I+J>n)=0;
>>imDeformee=idct2(imDctModifie) ;
```

B2. Transformée en cosinus discrète locale

Dans la norme de JPEG, la transformée en cosinus discrète est en fait appliquée par bloc de 8x8. De fait ce choix diminue de façon importante la complexité des calculs.

> Choisissez une image en niveaux de gris dont le nombre de lignes et de le nombre de colonnes sont chacun des multiples de 8. Comparez visuellement l'image obtenue en annulant une certaine proportion de coefficients de sa DCT lorsque ces coefficients ont été calculés sur l'ensemble de l'image et lorsqu'ils ont été calculés par bloc de 8x8. Commentez les différences.

Instructions Matlab à tester pour réaliser le programme :
Rajout des lignes et des colonnes à une image de façon à obtenir une image dont le nombre de lignes et le nombre de colonnes sont chacun des multiples de 8 :

```
>>im=[im;zeros(8*ceil(size(im,1)/8)-size(im,1),size(im,2))];
>>im=[im zeros(size(im,1),8*ceil(size(im,2)/8)-size(im,2))];
```

Application de la dct par bloc de 8x8 :

```
>>imDct88=blkproc(im,[8 8],@dct2);
```

Annulation de coefficients par bloc

```
>>[jBloc,iBloc]=meshgrid(1 :8,1 :8);
>>iBloc=repmat(iBloc,size(im)/8);
>>jBloc=repmat(jBloc,size(im)/8);
```

```
>>imDct88Modifiee=imDct88;
>>imDct88Modifiee(iBloc+jBloc<n)=0;
```

C. Quantifications des coefficients de la DCT

L'apport de la transformée en cosinus discrète n'est pas seulement de concentrer ses valeurs sur un plus petit nombre de coefficients, c'est aussi que visuellement l'image est peu déformée lorsqu'on varie un tout petit peu ces coefficients. C'est cela qui permet de faire de la compression en quantifiant les coefficients de la DCT.

Pour des images en niveaux de gris, la norme JPEG consiste d'abord transformer les niveaux de gris en valeurs entre -128 et 127, ensuite à calculer par bloc de 8x8 les coefficients de la dct, et enfin à quantifier ces coefficients avec un pas de quantification qui dépend de la position du pixel à l'intérieur du bloc 8x8. Ces pas de quantifications sont donnés par une matrice.

```
m=[   16 11 10 16 24 40  51  61
      12 12 14 19 26 58  60  55
      14 13 16 24 40 57  69  56
      14 17 22 29 51 87  80  62
      18 22 37 56 68 109 103 77
      24 35 55 64 81 104 113 92
      49 64 78 87 103 121 120 101
      72 92 95 98 112 100 103 99]
```

Il est possible de modifier le compromis qualité de la compression/mémoire requise en divisant par un coefficient appelé qualite qui varie entre 0 et 1 : 1 pour la meilleur qualité et 0 pour la plus mauvaise. Ces coefficients sont ensuite quantifiés avec l'algorithme de huffman (au préalable on remet les coefficients entre 0 et 255).

Pour approcher le taux de compression, code est une succession de 0 et de 1 et donc ce vecteur requière en place mémoire un nombre de bits égale à sa longueur. info détient les informations nécessaires pour retrouver l'image de départ, il s'agit essentiellement de la signification des codes mémoires qui sont codés dans info.huffcodes sous la forme d'un vecteur nul sauf en un petit nombre de cases, les valeurs en ces cases peuvent être codés avec 8 bits et la position dans ce vecteur avec 16 bits.

> Implémentez sur une image en niveau de gris cet algorithme de compression (niveaux de gris centrés, calcul des coefficients de la dct par blocs de 8x8, quantification des composantes, codage huffman des composantes). Proposez une autre matrice m et montrez expérimentalement qu'à même taux de compression, l'autre matrice donne une image de qualité similaire ou plus mauvaise. Le coefficient qualite permet justement d'ajuster l'expérimentation de façon à obtenir le même taux de compression avec deux matrices m différentes.

Instructions Matlab à tester pour réaliser le programme :

Déclaration de la matrice m

```
>>m(1,1:8)=[  16 11 10 16 24 40  51  61];
>>m(2,:)=[     12 12 14 19 26 58  60  55];
>>m(3,:)=[     14 13 16 24 40 57  69  56];
>>m(4,:)=[     14 17 22 29 51 87  80  62];
>>m(5,:)=[     18 22 37 56 68 109 103 77];
>>m(6,:)=[     24 35 55 64 81 104 113 92];
>>m(7,:)=[     49 64 78 87 103 121 120 101];
>>m(8,:)=[     72 92 95 98 112 100 103 99];
>>qualite=1; m=m/qualite;
```

Quantification

```
>>imDctQuantifiee=blkproc(imDct,[8 8],'round(x./P1)',m);
```

Décodage

```
>> imDctNouvelle=imDctQuantifiee.*repmat(m,size(imDctQuantifiee)/8);
```

Taux de compression

```
>>bpp= (length(find(info.huffcodes~=0))*32+length(code))/prod(size(im));
```

D. Ordonnancement des coefficients de la DCT

Il est possible de gagner en compression en regroupant les coefficients nuls de la DCT. Pour cela on réordonne les coefficients de la dct suivant un ordre en zigzag, de cette façon les coefficients nuls sont à la fin. Et on remplace les derniers coefficients nuls de chaque bloc par un coefficient de fin de séquence noté EOB (End Of Block). En pratique EOB est une valeur égale à $1+C_{\max}$ où $C_{\max}$ est la composante la plus élevée.

Le vecteur ligne suivant indique l'ordre de lecture au sein d'un bloc, les composantes indiquées désignent les cases, elles sont numérotées de 1 à 64 par incrémentation le long des colonnes (9 désigne la case positionnée sur la première ligne et la deuxième colonne).

```
ordre=[1 9 2 3 10 17 25 18 11 4 5 12 19 26 33…
       41 34 27 20 13 6 7 14 21 28 35 42 49 57 50
       43 36 29 22 15 8 16 23 30 37 44 51 58 59 52
       45 38 31 24 32 39 46 53 60 61 54 47 40 48 55
       62 63 56 64];
```

Calculez la place mémoire ainsi économisée pour l'image choisie. Calculez le rapport de compression en bpp (bit par pixel).

Instructions Matlab à tester pour réaliser le programme :

Rangement des coefficients de la DCT en une succession de colonnes de longueur 64.

```
>>imDctQuantifieeCol=im2col(imDctQuantifiee,[8 8],'distinct');
```

Nouvel ordonnancement :

```
>>ordre=[1 9 2 3 10 17 25 18 11 4 5 12 19 26 33];
>>ordre=[ordre 41 34 27 20 13 6 7 14 21 28 35 42 49 57 50];
>>ordre=[ordre 43 36 29 22 15 8 16 23 30 37 44 51 58 59 52];
>>ordre=[ordre 45 38 31 24 32 39 46 53 60 61 54 47 40 48 55];
>>ordre=[ordre 62 63 56 64];
>>imDctQuantifieeColOrdonnee=imDctQuantifieeCol(ordre,:);
```

Visualisation de l'ordonnancement :

```
>>figure(1) ; imshow(abs(imDctQuantifieeCol)/5) ;
>>figure(2) ; imshow(abs(imDctQuantifieeColOrdonnee/5) ;
```

Supprimer les zéros à la fin d'un vecteur colonne

```
>>u=[0 1 1 0 1 0 0 0 0 ] .';
>>v=u(1 :find(u~=0,1,'last')) ;
```

Retrouver une matrice de taille fixe à partir d'un vecteur et des marqueurs EOB

```
>>v=[1 7 2 3 3 7 4 8 7] ;
>>EOB=max(v);
>>indiceFin=find(v==EOB)-1 ;
>>indiceDebut=[1 indiceFin(1:end-1)+2];
>>longueur=indiceFin-indiceDebut+1;
>>w=zeros(64,length(indiceFin));
>>for numBloc=1:length(indiceFin)
>>  w(1:longueur(numBloc),numBloc)=v(indiceDebut(numBloc):indiceFin(numBloc));
>>end;
```

Trouver l'ordre inverse (cet *ordreInverse* s'applique exactement de la même façon que *ordre*):

```
>>[a,ordreInverse]=sort(ordre);
```

Recompose rune image à partir des blocs 8x8 rangés en une succession de colonnes de 64 lignes :

```
>>im=col2im(im,[8 8],[256 256],'distinct');
```

E. Sous-échantillonnage de la chrominance

Dans la norme JPEG, une image couleur est d'abord décomposée en chrominance et en luminance, la chrominance est ensuite sous-échantillonnnée (après filtrage par un passe-bas) et ensuite chaque composante colorimétrique est compressée séparément. Il n'y a pas besoin de centrer les composantes de chrominance, elles sont en effet déjà en parti centrées.

Indiquez le niveau de compression ainsi atteint.

Instructions Matlab à tester pour réaliser le programme :
Pour convertir en *ycbcr*, il faut que im soit une image couleur en format doublee et à valeurs entre 0 et 1.

```
>>imycbcr=rgb2ycbcr(im) ;
```

Pour sous-échantillonner la chrominance

```
>> imycbcrSsEchY=imycbcr(:,:,1);
>>imycbcrSsEchCb=blkproc(imycbcr(:,:,2),[1 size(imycbcr,2)],@decimate,2);
>>imycbcrSsEchCr=blkproc(imycbcr(:,:,3),[1 size(imycbcr,2)],@decimate,2);
```