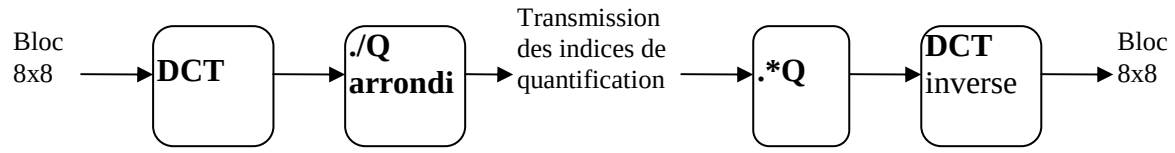
ST4-CN2 : Codage de Source Sujet remis par P.Jardin.
(plusieurs fonctions ont été gracieusement fournies par J.F.Bercher).

# TP COMPRESSION D'IMAGES

On se propose d'examiner lors de cette séance de travaux pratiques la mise en œuvre d'une méthode de compression d'images de type JPEG. Nous pourrons ici prendre certaines libertés par rapport à la stricte norme JPEG (processus de fond), de manière à simplifier l'implantation, ou mettre en évidence les caractéristiques de la méthode de compression. Il n'en reste pas moins que le but du TP est d'obtenir une véritable méthode de compression, c'est-à-dire, partant d'une image, construire un train binaire (un ensemble ordonné de bits), dont le décodage permet de retrouver une image proche de l'image initiale.
Le TP sera réalisé sous Matlab. On fournit un certain nombre de fonctions. D'autres devront être réalisées par vos soins. Des images, de différentes tailles, sont disponibles au format PGM (Portable GrayMap). Il s'agit d'images en niveaux de gris, sur 8 bits, soit 256 niveaux. La fonction ***getpgm*** permet de charger ces images sous la forme d'une matrice. La commande ***Img=getpgm('bird.pgm')*** ; permet par exemple de charger l'image bird.pgm dans la matrice Img. La fonction ***imagesc*** (image scaled) permet d'afficher une image en exploitant toute l'échelle fournie par la colormap. Dans notre cas, pour afficher en niveaux de gris, il faudra choisir ***colormap('gray').***
Le standard prévoit de centrer l'échelle des niveaux de gris [-128,127]. Vous appliquerez donc : ***Img=Img-128*** ;
Images disponibles : bird.pgm, boat.pgm, frog.pgm, lena.pgm, mandrill.pgm, peppers.pgm.

## Principale étape de compression

Le cœur de la compression JPEG consiste à effectuer sur chaque bloc LxL de l'image (L=8 pour la norme JPEG) les opérations représentées sur le schéma suivant :




Travail demandé :

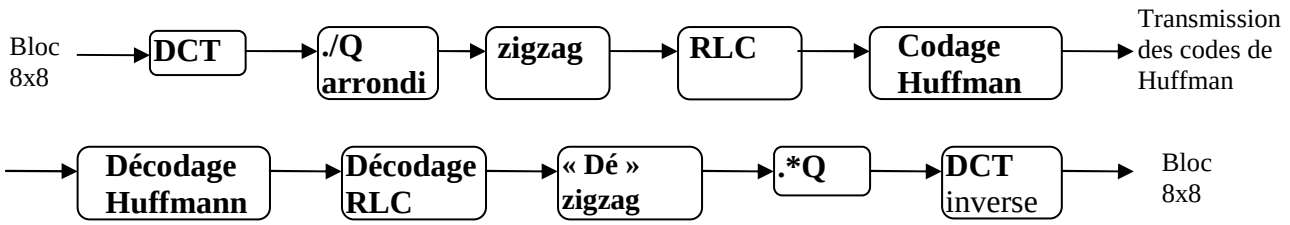
a) Écrire une fonction cod1.m qui calcule la DCT(***dct2.m***), sur des blocs de taille LxL d'une image Img, calcule les indices de quantification sur les blocs transformés (cette opération peut s'effectuer sur l'ensemble du bloc transformé par une *division de matrices point à point)* et retourne l'ensemble des blocs transformés indicés (dans une matrice de mêmes dimensions que l'image originale) La syntaxe d'appel de la fonction cod1 devra être : ***[ImgTr] = cod1(Img,L,Q) ;***
b) Ecrire la fonction de décodage : ***[ImgRec] =decod1(ImgTr,L,Q)*** qui, pour chaque bloc d'indices (pixels transformés), multiplie ceux-ci par les pas de quantification correspondants (*multiplication de matrices point à point*) pour obtenir le bloc transformé quantifié et opère la DCT inverse (***idct2.m*** ).
c) Ecrire la fonction ***[D]=distimage(Img,Imgrec)*** qui calcule la distorsion moyenne (écart quadratique moyen) entre l'image originale et l'image reconstituée.
d) Vous testerez ces fonctions sur une des images disponibles pour différentes matrices de quantification Q :
- avec une quantification optimale : ***Q=stdQ (***la fonction ***stdQ*** retourne la matrice de pas définie dans le standard JPEG).
- avec une quantification uniforme : ***Q= q*ones(L,L)*** où q est la valeur moyenne des pas de quantification de stdQ.
- avec des versions dégradées de ces matrices (c'est-à-dire avec des pas de quantification multipliés par un facteur $2^p$, à essayer pour p = 1 (réduction d'un bit par pixel a priori) ou p= 2 (réduction de 2 bits par pixel). Vous commenterez les résultats obtenus et étudierez en particulier la dépendance des distorsions objective (mesurée par D) et subjective (ce que vous voyez) en fonction du choix de la matrice de quantification.

e) Calculez et tracez l'histogramme des indices de quantification obtenus. Vous interpréterez cet histogramme :
- estimation du nombre de bits nécessaires (sans faire appel à une technique de codage sans pertes) pour transmettre à priori les indices de quantification (pixels transformés).
- remarque sur le nombre de valeurs nulles
- estimée de l'entropie de ces indices et ré-estimation du nombre de bits par pixel a priori nécessaire (en faisant appel à une technique de codage sans pertes). Pour cette dernière étape vous pouvez vous inspirer du script ***gene_code.m*** utilisé dans la suite.

Pour la suite du TP on considérera le cas Q=stdQ (mais il vous sera loisible de tester d'autres choix à vos moments perdus…)

## Autres étapes de la compression JPEG

Le schéma complet d'une chaine de codage –décodage JPEG peut être représenté comme suit :




**1) Lecture en zigzag et Run Length Coding**
Chacun des blocs DCT quantifiés obtenus est lu en zigzag, de manière à regrouper les coefficients non nuls. Un codage RLC modifié est ensuite effectué.
Les fonctions suivantes sont fournies. La fonction ***zz.m*** permet d'effectuer la lecture en zigzag. La fonction ***RLC_jpg.m*** retourne une matrice au format RLC de JPEG, à savoir [valeur non nulle , nombre de valeurs à zéro suivantes]. La fonction ***codQzzRLC.m*** réalise ces deux opérations sur les blocs transformés d'une image.

Travail demandé : Observer les résultats par étapes de cette fonction sur un bloc d'image (en passant en argument une portion (8*8) de l'image) et en retirant les « ***;*** » pour afficher les résultats intermédiaires.

**2) Codage sans pertes – Codage de Huffmann**
Le script ***gene_code.m*** applique la fonction ***codQzzRLC*** à l'ensemble des blocs de 4 images disponibles afin d'obtenir une base d'apprentissage de couples [valeur non nulle , nombre de valeurs à zéro suivantes] notés (vnz, nbz). L'auteur de ce texte a ensuite estimé pour chacune des grandeurs vnz (valeur non nulle) et nbz (nombre de valeurs à zéro suivantes) les distributions de probabilités de ces grandeurs à partir d'histogrammes. Ces histogrammes et l'entropie qui leur est associée sont affichés par le script. Enfin la connaissance des distributions de probabilité permet de construire (grâce à la fonction standard ***huffmandict***) un dictionnaire (ensemble de mots de code) pour coder chacune de ces deux grandeurs.
Remarque : dans la norme JPEG c'est un dictionnaire unique qui est construit pour des couples de valeurs (vnz,nbz).

Travail demandé : Interpréter la figure générée par ce script et déduire des différentes grandeurs fournies (entropie, nb de valeurs RLC pour l'ensemble des 4 images, nombre de pixels pour l'ensemble des 4 images) un taux de compression estimé pour la norme JPEG, ou un nombre de bits par pixel. Faire le lien avec la dernière question de la partie « Principale étape de compression ».

Les lignes suivantes donnent des exemples d'utilisation de ces dictionnaires pour coder et décoder des valeurs vnz,nbz de couples RLC à l'aide des fonctions standard ***huffmanenco.m et huffmandeco.m***

```
%exemple d'utilisation des codes
code_vnz=huffmanenco([1 5 -7 5],dico_huff_vnz) )%codage d'une séquence de valeurs vnz
vnz=huffmandeco(code_vnz,dico_huff_vnz) %décodage de la séquence codée
```

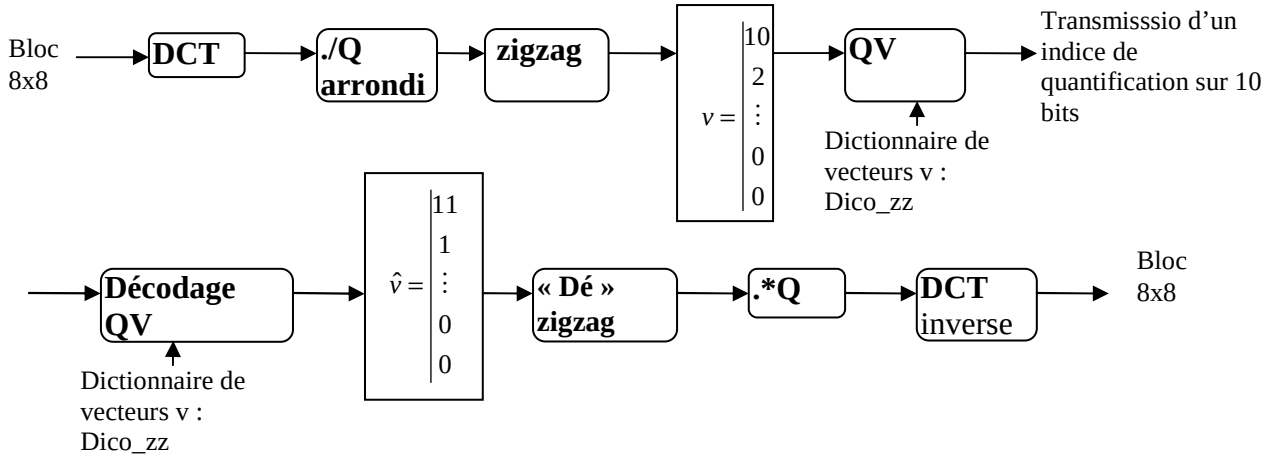
Travail demandé : modifier (compléter) la fonction ***codQzzRLC.m*** (que vous renommerez ***codJPEG.m***) pour générer les deux trains binaires correspondant aux deux suites de valeurs vnz et nbz codées par codage de Huffman. Vous calculerez par ailleurs le taux effectif de compression obtenu pour l'image traitée.
La syntaxe de cette fonction sera :
***[codebin_vnz,codebin_nbz,taux]=codJPEG(Img,L,Q,dico_huff_nbz,dico_huff_vnz);***

**3) Décodage**
L'auteur (très aimable) de ce texte fournit une fonction de décodage pour vérifier votre fonction :
***[imgrec]=decodJPEG(codebin_vnz,codebin_nbz,L,Q,NB,dico_huff_nbz,dico_huff_vnz)***
(NB=size(img)/L doit être passé en argument de la fonction pour donner le nombre de blocs à décoder).
L'auteur, décidemment sympathique, fournit aussi le script du programme principal (pas vraiment compliqué) qui appelle les deux fonctions tout à tour. C'est ***TP_JPEG.m***
Vous vérifierez en particulier que vous n'avez pas introduit de distorsion supplémentaire par cette étape de codage sans pertes.

## Codage très bas débit – utilisation de la Quantification Vectorielle

Cette partie complète l'illustration des méthodes de compression vues en cours en vous proposant de mettre en œuvre une étape de Quantification Vectorielle (QV) sur les vecteurs d'indices des blocs transformés lus en zigzag.

L'idée de cette partie est de remplacer les étapes suivantes : codage RLC et Huffman par une étape de quantification vectorielle des vecteurs v sur 10 bits, c'est-à-dire avec 10/64 bit par pixel
Le schéma pour la chaine de codage décodage devient alors :




Il faut pour mettre en œuvre ce codage créer tout d'abord le dictionnaire de QV c'est-à-dire l'ensemble des $2^{10}$=1024 meilleurs représentants des vecteurs v. Ensuite on peut réaliser le codage décodage suivant le schéma de principe précédent.

**1) Création du dictionnaire**

Suivant le même principe que celui utilisé dans ***gene_code***, le script ***gene_dico_QV*** construit un dictionnaire «optimal» noté ***dico_zz*** à partir d'une base d'apprentissage constituée des vecteurs v correspondant à l'analyse (DCT, ./Q et lecture zigzag ) des blocs de 4 images disponibles. Ce script met en œuvre l'algorithme LBG itératif vu en TD. Remarque : ici le nombre de vecteurs obtenus (N≈4000) n'est pas très grand devant le nombre de représentants du dictionnaire.

Travail demandé : Observer les figures générées par ce script qui montrent dans le plan des 2 premières composantes de v l'évolution des représentants ainsi que l'évolution de la distorsion au cours de l'algorithme.

**2) Codage utilisant la QV**

Il s'agit de mettre en œuvre le début de la chaine représentée ci-dessus qui délivre la suite des indices des plus proches représentants des vecteurs v issus de l'analyse des blocs d'une image

Travail demandé : modifier (compléter) la fonction ***codQzz.m*** (que vous renommerez ***codQzz_QV.m***) pour générer la suite d'indices des plus proches représentants des vecteurs v (vous utiliserez la fonction ***ppv*** pour trouver ces plus proches représentants).

La syntaxe de cette fonction sera : ***[indice]=codQzz_QV(Img,L,Q,dico_zz);***

**3) Décodage**

Il s'agit de restituer une image décodée à partir de la suite d'indices (mise en œuvre de la fin de la chaine représentée).

L'auteur de ce texte fournit la fonction de décodage : ***[imgrec]=decodQzz_QV(indice,L,Q,dico_zz,NB)***

ainsi que le script du programme principal qui appelle les deux fonctions tout à tour (***TP_QV.m***)

Travail demandé : Commentez le résultat obtenu en termes de débit et distorsion par rapport aux techniques vues dans la première partie.