Initiation au traitement du signal et applications
Séance 4

École des Mines de Nancy

Frédéric Sur
http://www.loria.fr/~sur/enseignement/signal/
sur@loria.fr

Consigne : Il vous est demandé de **rédiger les travaux pratiques**. Ouvrez en parallèle un document Word dans lequel vous allez coller les différents résultats obtenus, les commandes qui vous ont permis de les obtenir, et les réflexions que cela vous inspire. L'évaluation de ce cours sera basée sur ces comptes rendus.

1 Remarque préliminaire

Les fonctions Matlab `dct.m` et `idct.m` fournies permettent respectivement de calculer les transformées en cosinus DCT et DCT-inverse, en utilisant la FFT comme dans la formule vue en cours. Ces fonctions agissent colonne par colonne lorsqu'on leur donne une matrice en entrée. De la même manière que pour la Transformée de Fourier Discrète, calculer la DCT d'une image revient à calculer une première DCT sur les colonnes, puis une seconde DCT sur les lignes (*idem* pour la DCT inverse). Vous trouverez un exemple dans la section 4.

2 Une compression brutale du son

Voici un algorithme de compression du son assez brutal : pour un signal de son à compresser, on calcule sa DCT et on met à zéro tous les coefficients dont la valeur absolue est en dessous d'un certain seuil.

La décompression consiste simplement à calculer la DCT inverse du signal ainsi compressé.

Écrivez une fonction Matlab qui implante ce schéma de compression.

Le taux de compression sera ici défini comme le rapport entre le nombre de coefficients mis à zéro et le nombre total de coefficients :

```
length(find(abs(dctsignal)<seuil))/length(signal)
```

Faites des essais sur le fichier `foryoublue.wav`. À partir de quel taux de compression le rendu se dégrade-t-il ? Parvenez-vous à entendre l'effet de « pré-écho » ? Quel est le lien avec l'effet de Gibbs ?

3 Étude expérimentale de la compression JPEG

Compressez au format JPEG les images `lettres.tif` et `boat.tif` à des taux de plus en plus forts. Vous utiliserez la commande Matlab :

```
imwrite(uint8(Image),'Imagecompressee.jpg','quality',100)
```

en diminuant le paramètre "quality", ici à 100%

Examinez la taille du fichier créé. Jusque quel taux de compression jugez-vous que la dégradation n'est pas visible ?

Sur quelles zones de l'image commencez-vous à observer les dégradations ? Au contraire, quelles sont les zones supportant les plus gros taux de compression ?

Identifiez les deux types d'artefacts *block effect* et *ringing*[1]. Interprétez-les à l'aide du cours.

À votre avis, pourquoi JPEG est basé sur des blocs de taille 8x8 et pas plus gros ?

Faites la même expérience sur l'image couleur `GrandTeton.jpg`. Visualisez l'effet de bloc et le phénomène de *ringing*, ainsi qu'une dérive des couleurs à fort taux de compression. À quoi peut-elle être due ?

4 Suréchantillonnage par *zero padding*

Le suréchantillonnage d'une image consiste à augmenter sa taille. Il s'agit d'une opération de plus en plus utilisée pour (par exemple) adapter les anciennes vidéos à la diffusion dite « haute définition ».

On considère des images de taille $N \times N$, et on veut les interpoler en des images de taille $kN \times kN$ (avec k valant 2,4, 8).

Voici trois méthodes possibles.

– duplication des pixels. On remplace chaque pixel de l'image originale par un bloc $k \times k$ obtenu par duplication.
– on calcule la transformée de Fourier de l'image originale de taille $N \times N$, on la complète par des zéros pour former une matrice de taille $kN \times kN$, puis on prend la transformée de Fourier inverse.
– même raisonnement, mais avec la transformée en cosinus.

Les deux dernières méthodes sont dites par *zero padding* (remplissage par des zéros).

Implantez les trois méthodes, et utilisez-les sur `boat.tif` et `lena.tif`. On commencera par le cas $k = 2$, et on ne dépassera pas $k = 4$ pour l'image `boat`.

Indication pour la méthode par duplication : `Image(1:2:512,1:2:512)` désigne la sous-matrice de `Image` (ici de taille 512x512) formée des lignes et colonnes d'indices impairs.

Indication pour les méthodes par zero padding : voici le code à essayer pour `lena`. Expliquez les manipulations opérées[2], et adaptez-le pour `boat`.

```
>> Image=double(imread('lena.tif'));
>> fftIma=fftshift(fft2(Image));
>> dctIma=dct(dct(Image)');
>> fftIma2=zeros(2*size(Image));
>> dctIma2=zeros(2*size(Image));
>> fftIma2(129:129+255,129:129+255)=fftIma;
>> dctIma2(1:256,1:256)=dctIma;
>> Ima2dct=idct(idct(dctIma2)');
>> Ima2fft=ifft2(ifftshift(fftIma2));
>> figure,colormap(gray),imagesc(Ima2dct)
>> figure,colormap(gray),imagesc(real(Ima2fft))
```

Comparez les méthodes, en particulier le comportement des deux dernières près des contours des images (utilisez la loupe). Quel est l'avantage de la méthode basée sur la DCT par rapport à celle basée sur la DFT ?

Question bonus : même expérience sur une image couleur dont on traite les canaux indépendamment.

[1] Le phénomène de *ringing* est le nom généralement donné à l'effet de Gibbs pour les images

[2] En particulier, pourquoi ces différences de traitement entre `dct` et `fft` ?