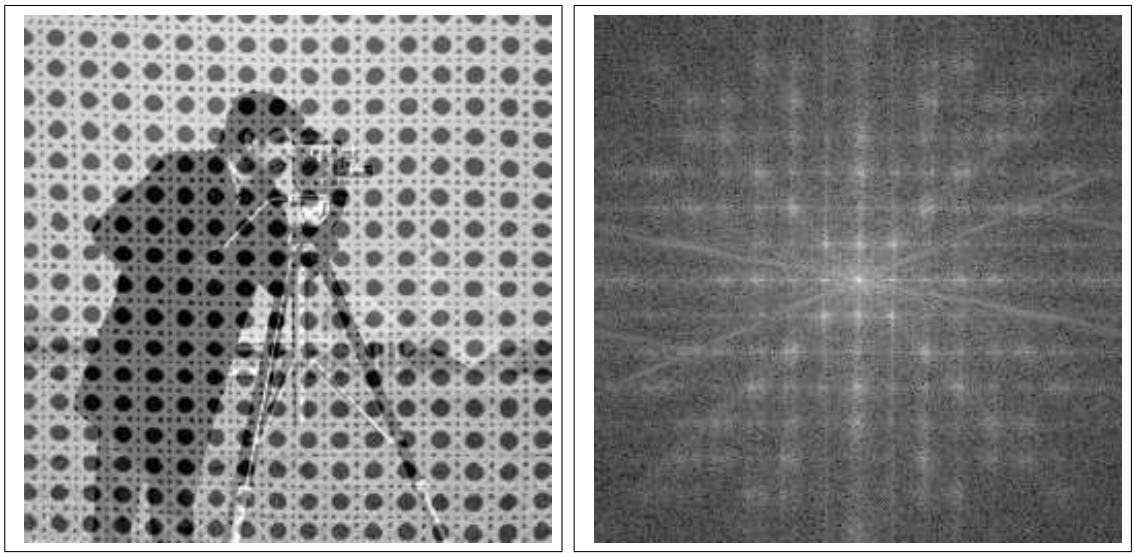
Télécom Physique de Strasbourg **UE EP012M70**

TRAITEMENT DU SIGNAL BIDIMENSIONNEL ET DES IMAGES



TRAVAUX PRATIQUES - V6

F. Heitz et M. Louys (Janvier 2024)

Remerciements et sources : Certains TP sont adaptés des sources suivantes :
• Gilles Burel : Introduction au traitement d'images : simulation sous MatLab, Lavoisier, Hermes Science, 2001.
• Gabriel Peyré : A numerical tour of signal processing :
`http://www.numerical-tours.com/matlab/`
• IPOL Journal - Image Processing On Line : `http://www.ipol.im/`
• EPFL Biomedical Imaging Group demos : `http://bigwww.epfl.ch/demo/`

Table des matières

Déroulement des séances

Planning indicatif

Séance	Questions	Titre du Thème
1	1-8	Analyse spectrale
2	9-13	Recalage par corrélation
3	14-20	Tomographie
4	21-32	Détection de contours et Débruitage par filtres
5	33-36	Débruitage moyenne non locales et échantillonnage
6	37-42	Interpolation
7	43-48	Ondelettes 1D
8	49-54	Ondelettes 2D : introduction

Environnement de travail et prérequis
Ces travaux pratiques sont à réaliser sous environnement MatLab (R14SP3 : MatLab 7.1, SP Toolbox 6.4, IP Toolbox 5.1). Les étudiants doivent gérer eux-même leur apprentissage des outils fondamentaux en traitement d'images :

- connaissance et compréhension d'un ensemble de notions et mots-clés ;
- utilisation des outils Matlab de la toolbox "Image Processing".
- consultation *préalable* de la documentation sur le site "MathWorks" thème MatLab notamment.

Les images et fichiers MatLab nécessaires au TP sont disponibles sur la page Moodle "TP-TS2D 2A TravauxPratiques" `https://moodle3.unistra.fr/course/view.php?id=21958`

Les séances de TP s'appuient sur les notions et techniques développées dans les cours, TD et TP suivants :

- 1A Outils numériques : initiation à MatLab (TP)
- 1A Introduction au traitement du signal : signaux déterministes analogiques (C, TD, *F. Heitz*)
- 1A Probabilités (C, TD, TP, *C. Heinrich*)
- 2A Statistiques (C, TD, TP, *F. Heitz et P. Charbonnier*).
- 2A Traitement numérique des signaux et signaux aléatoires. (C, TD, TP, *Y. Takakura et F. Heitz*)
- 2A Traitement du signal bidimensionnel et des images (C, TP, *F. Heitz*)

Modalités de contrôle Émargement obligatoire en début de séance, discussion scientifique durant la séance, **examen final sur machine (2H)**

1 Analyse spectrale 2D et corrélation

1.1 Rappel : affichage d'une image. Lecture, écriture des fichiers graphiques

Une image numérique est une matrice rectangulaire [1] dont les éléments (appelés pixels) sont à valeurs dans un ensemble fini de valeurs discrètes (quantification). Dans le cas d'une image monochrome, un codage sur q bits par pixel permet de représenter 2^q niveaux de gris différents. Une image binaire correspond à $q = 1$. Une image "en niveaux de gris" (souvent appelée "image d'intensité"' ou de "luminance"') est généralement codée sur $q = 8$ bits (parfois $q = 16$ bits). Une image en "vraies couleurs" (RVB ou RGB pour "Red Green Blue"en anglais) est représentée par la juxtaposition de 3 plans de couleurs (R, V, B), chacun codé sur q bits. Ceci permet de représenter 2^{3q} couleurs différentes (16 millions de couleurs pour $q = 8$).

Dans MatLab les pixels sont en pratique représentés par les types suivants :

- type logique pour les images binaires ;
- types uint8, uint16 (entier non signé sur 8 ou 16 bits), type int 16 (entier signé sur 16 bits) pour les images monochromes ou un plan d'une image couleur ;
- les types single ou double (réels sur 64 bits), sont également utilisés (les valeurs des pixels sont alors dans $[0, 1]$, où 0 est le noir, 1 est le blanc).

Une autre façon de représenter les images en couleurs est l'indexation par une table de couleur (colormap).

Une des fonctions qui permet de visualiser une image sous MatLab est `imshow`. On pourra utiliser l'instruction `imshow(I, 'InitialMagnification','fit')` où `I` désigne l'image et les autres paramètres indiquent à MatLab d'adapter la taille de l'image à la taille de fenêtre de visualisation.

Une autre fonction, très utile pour la visualisation est `imagesc(I)`. Cette fonction permet d'afficher une image, *en occupant toute la dynamique disponible dans la table de couleur couramment spécifiée*. La fonction `colormap(gray)` permet de spécifier une table de couleur en niveaux de gris.

`imtool(I)` permet également de visualiser une image, en modifiant l'image visualisée de façon interactive.

Par ailleurs sur le disque dur de l'ordinateur, une image est contenue dans un fichier. Les principaux formats de fichiers d'images auxquels nous aurons affaire sont les formats graphiques standard : jpeg, tiff... Ces fichiers peuvent être créés par l'instruction `imwrite` et se lisent avec l'instruction `imread` :

`I=imread('frog.jpg')` lit l'image frog.jpg et la place dans la matrice I.

L'afficher par :

```
imshow(I ,'InitialMagnification','fit' ) ;
```

1. On notera que les indices de lignes et de colonnes commencent à 1 et non à 0 comme c'est le cas de certains langages de programmation, tels que le langage C et C++.

ou directement :
`imshow('frog.jpg','InitialMagnification','fit' );`

1.2 Transformation de Fourier : visualisation d'un spectre centré

Matlab calcule la transformée de Fourier discrète (TFD) 2D grâce à la fonction `fft2` (la transformée inverse est obtenue par `ifft2`).

1. Faire un programme visualisant une image I, puis le module de son spectre. L'appliquer à l'image `cameraman.tif` . Que remarque t-on ?

   Ajuster la dynamique du spectre avec `imtool`. Commenter.

2. En pratique, on utilise une échelle logarithmique pour visualiser correctement les modules des spectres : `imagesc(log(abs(fft2(I))+eps))` . Visualiser ainsi le spectre. Où se situe la fréquence nulle $(0, 0)$?

3. Comme on le constate la fréquence nulle n'est pas au centre de l'image. Pour une visualisation plus facile à interpréter, on utilise la fonction `fftshift` qui décale la fréquence nulle au centre de l'image, et les hautes fréquences près des bords :
   `imagesc(log(abs(fftshift(fft2(I)))+eps))`. L'appliquer à l'image. Quelles sont les propriétés du module du spectre ?

4. Visualiser la phase du spectre (on n'utilise pas d'échelle logarithmique dans ce cas).

5. Visualiser les spectres des textures naturelles suivantes : `D20.gif` , `D21.gif` , `D101.gif`. Commenter l'allure des spectres obtenus.

6. Visualiser le spectre de l'image `cameraman.tif` et filtrer l'image dans le plan de Fourier en ne conservant que le spectre $5\times$ 5 autour du centre. Visualiser l'image filtrée obtenue.

1.3 Transformation de Fourier : réduction du bruit

7. Une application typique du filtrage est la réduction de bruit. Le bruit est souvent localisé dans les hautes fréquences du spectre, aussi est-il possible de réduire ses effets en filtrant passe-bas l'image. On effectue ici un filtrage rudimentaire du bruit dans le domaine de Fourier, en éliminant dans le spectre les fréquences où le bruit est présent.

   Afficher les images `house.jpg` et `house_noise.jpg`, ainsi que leurs spectres. Dans quel domaine de fréquences se trouve le bruit venant dégrader l'image (préciser) ?

   Après avoir compris et complété le programme ci-dessous, on comparera `I, Ibruitee` et `Idebruitee` pour différentes valeurs du paramètre SIZE. A partir des spectres des images, déterminer la valeur optimale de ce paramètre. Interpréter ce paramètre. L'image débruitée est elle identique à l'image originale ? Pourquoi ?

```
SIZE=‘‘paramètre à déterminer" ;
I=imread('house.jpg') ;
I=rgb2gray(I) ;
I=double(I)/255.0 ;
imagesc(I);
colormap(gray);
Ibruitee=imread('house_noise.jpg');
Ibruitee=double(Ibruitee)/255.0 ;
figure;
```

```
imagesc(Ibruitee);
colormap(gray);
FF=fftshift(fft2(Ibruitee));
PBAS=zeros(256,256);
PBAS((129-SIZE/2):(128+SIZE/2),(129-SIZE/2):(128+SIZE/2))=ones(SIZE,SIZE);
%ici on filtre le bruit : ligne manquante à compléter
FF=ifftshift(FF);
Idebruitee=ifft2(FF);
figure;
imagesc(abs(Idebruitee));
colormap(gray);
```

8. Afficher de même l'image `cameraman-texture.tif` ainsi que son spectre. On souhaite retrouver l'image sous-jacente du cameraman. Proposer une méthode automatique (la décrire de façon détaillée, l'implantation Matlab n'est pas demandée).

1.4 Corrélation 2D. Application au recalage d'images

On considère deux images I et J, la seconde image étant simplement une version translatée de la première, par un vecteur (t_x, t_y). On souhaite recaler ces deux images en utilisant la méthode par corrélation. Recaler signifie en particulier déterminer le vecteur de translation (t_x, t_y) entre les deux images. Les calculs de corrélation seront effectués *dans le domaine de Fourier*, ce qui permettra d'utiliser la FFT pour une plus grande efficacité calculatoire.

9. Rappeler l'expression de la corrélation classique dans le domaine spatial et dans le domaine de Fourier.
10. *Recalage d'images par corrélation de phase.* Le recalage par corrélation de phase consiste à calculer l'expression suivante :

$$R_{IJ}^{phase}(m,n) = TFD^{-1}\left(\frac{\mathcal{I}(u,v)}{|\mathcal{I}(u,v)|}\right)\left(\frac{\mathcal{J}(u,v)}{|\mathcal{J}(u,v)|}\right)^*$$

et à chercher la position de son maximum.
Exprimer $\left(\frac{\mathcal{I}(u,v)}{|\mathcal{I}(u,v)|}\right)\left(\frac{\mathcal{J}(u,v)}{|\mathcal{J}(u,v)|}\right)^*$ en fonction des phases Φ_I et Φ_J de $\mathcal{I}$ et $\mathcal{J}$. Justifier l'appellation "corrélation de phase".

11. Montrer que si J est simplement une version translatée de I, alors $R_{IJ}^{phase}(m,n)$ présente un pic très marqué en (t_x, t_y).
La méthode par corrélation de phase donne généralement un pic beaucoup plus marqué que la méthode de corrélation classique. Elle est cependant beaucoup plus sensible aux distorsions entre les deux images, et peut être mise en échec alors que la méthode par corrélation classique donne toujours des résultats acceptables. La corrélation de phase peut être vue comme une méthode dans laquelle on réhausse les hautes fréquences des images à recaler. En effet, en général une image contient plus de basses fréquences que de hautes fréquences, aussi la division de $\mathcal{I}(u,v)$ par $|\mathcal{I}(u,v)|$ revient à amplifier les hautes fréquences par rapport aux basses fréquences (il en est de même pour $\mathcal{J}$). En première approximation, on peut dire que tout se passe comme si on appliquait la méthode de corrélation sur les images de contours et non pas sur les images sources. On doit alors distinguer deux cas :

- S'il y a peu de distorsions entre I et J, le fait de travailler sur les contours (donc sur des éléments fins) donne une meilleure précision pour le recalage (mathématiquement cela se traduit par un pic de corrélation plus marqué) ;
- en revanche, s'il y a distorsion entre les images, il devient impossible de mettre les contours en correspondance : ils sont en effet trop fins ; alors que si on utilise la corrélation classique (avec les images initiales), on arrive toujours à faire correspondre à peu près les différentes zones des deux images.

12. Le programme `TP1_correl.m` est un programme "à trous" qui code les deux fonctions de corrélation décrites précédemment (corrélation classique, corrélation de phase) et détermine leur maximum.

    Comprendre le fonctionnement de ce programme, le compléter et vérifier qu'il est bien conforme à la théorie.

13. Afin de simuler une distorsion de l'image, le programme `TP1_correl.m` permet d'appliquer une rotation à l'image, que les méthodes de recalage par corrélation décrites ci-dessus ne permettent pas de compenser (elles ne compensent que les translations) [2].

    En modifiant l'angle de rotation, déterminer à partir de quel angle de rotation la méthode de corrélation classique devient plus performante que la méthode de corrélation de phase. On remarquera qu'une faible distorsion suffit à rendre la corrélation de phase inefficace.

2. La rotation appliquée a pour centre le centre de l'image, afin de ne pas introduire de composante de translation supplémentaire

2 Tomographie

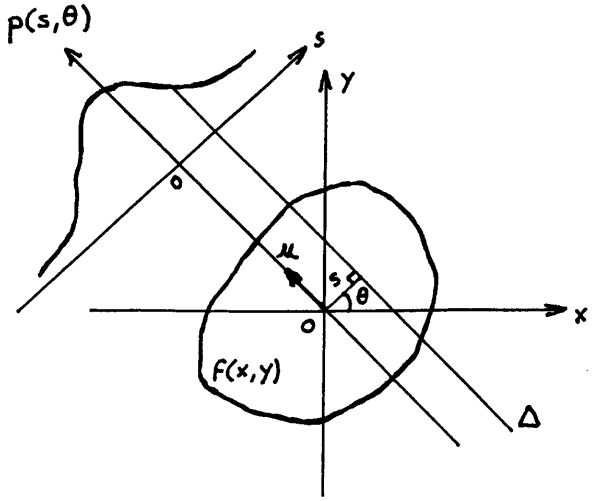


FIGURE 1 – Géométrie du système d'acquisition

On considère un système de tomographie 2D (de type scanner X) à projections parallèles, dont la géométrie est rappelée sur la figure 1.

14. Donner les équations de projection parallèle (transformée de Radon). Qu'appelle-t-on un sinogramme ?
15. Calculer le sinogramme d'un point (x_0, y_0). Montrer que c'est une sinusoïde (cf. exercice TD). Quel est le sinogramme du point origine $(x_0, y_0) = (0, 0)$?
16. Calculer le sinogramme de la droite d'équation $s_0 = x \cos\theta_0 + y \sin\theta_0$ (cf. exercice TD). Interpréter graphiquement.
17. Rappeler le principe de la reconstruction par rétroprojection filtrée (cf. cours et document : Bloch-Tomographie.pdf).
18. Simulation : code Matlab `myCT_reconstruction.m`

    Le script `myCT_reconstruction` est une implémentation de la transformée de Radon, et des rétroprojections simples et filtrées.

    Lancer les commandes suivantes :

    ```
    myCT_reconstruction
    myCT_reconstruction('disque2.jpg',4)
    myCT_reconstruction('disque2.jpg',16)
    myCT_reconstruction('disque2.jpg',256)
    ```

    Comprendre le fonctionnement du programme à partir du code source. Comment le filtrage est il réalisé dans la rétroprojection ? Tester sur d'autres images (disques, droites, lettres,

X-scan) se trouvant dans le même répertoire en faisant varier le nombre de projections. Interpréter.

19. Code à trous : `CT_reconstruction.m`

    Réaliser les mêmes fonctions que précédemment en complétant ce code en utilisant les fonctions radon et iradon de Matlab.

20. Plate-forme de simulation : `CT_demo V0.1`

    `http://fr.mathworks.com/matlabcentral/fileexchange/47382-ct-demo-v0-1`

    Cette plate-forme de simulation permet de simuler de façon plus réaliste (quoiqu'encore simpliste) un scanner X. Elle permet de sélectionner :
    - l'image à reconstruire ;
    - le courant du tube à rayons X qui va déterminer le niveau de bruit des images (valeurs $\in [0.01, 2000]$) ;
    - le nombre de projections (valeurs $\in [1, 1024]$) ;
    - le type de filtre utilisé dans la rétroprojection (voir : document Bloch-Tomographie.pdf).

    On affiche également la dose reçue par la patient en mS (indication non garantie par le constructeur !).

    Plus le courant du tube est élevé, meilleur est le rapport signal à bruit, mais plus forte sera la dose de rayons X reçue par le patient. Il en va de même pour le nombre de projections.

    A titre d'indication, les doses limites admises en France sont de : 1 mSv/12 mois glissants pour le public et de 20 mSv/12 mois glissants pour les travailleurs soumis à des radiations ionisantes.

    Observer le bruit (de Poisson) apparaissant pour les faibles valeurs de courant de tube et son effet sur la reconstruction. Faire également varier le nombre de projections.

    Modifier le filtre de reconstruction pour de faibles rapports signal et observer l'amélioration de la reconstruction. Qu'observe-t-on : sur le bruit, la résolution ? (voir : document Bloch-Tomographie.pdf).

3 Détection de contours

3.1 Détection de contours

Les contours jouent un rôle de premier plan en traitement d'image car ils permettent généralement de caractériser la forme des objets d'une image. Les méthodes les plus classiques de détection de contours consistent à estimer le gradient de l'image par des filtres puis à détecter les maxima du gradient.

Le jeu de filtres détecteurs de contours (Prewitt, Roberts ou Sobel) est constitué d'une paire de filtres linéaires qui détectent les contours dans deux directions orthogonales. Certains filtres sont prédéfinis sous Matlab grâce à la fonction `fspecial`. Voici un petit programme utilisant le filtre de Prewitt :

```
I=imread('frog.jpg');
I=rgb2gray(I);
I=double(I)/255.0;
seuil=0.8;
h=fspecial('prewitt');
v=-h';
Gh=filter2(h,I);
Gv=filter2(v,I);
G=sqrt(Gh.*Gh+Gv.*Gv);
Gs=(G>seuil);
```

21. Afficher les images et les interpréter
22. Comparer les contours fournis par les opérateurs de Sobel, Roberts et Prewitt en modifiant le cas échéant la valeur du seuil.
23. Quel est l'effet d'un changement de seuil ?

3.2 Détection de contours (passages par zéro de la dérivée seconde))

On peut localiser les contours par la recherche du maximum de la dérivée première (approche basée sur le calcul du gradient), mais on peut aussi rechercher le passage par zéro de la dérivée seconde (méthode du *zero crossing*). Dans le cas d'une image, il n'existe pas une dérivée seconde unique mais 4 dérivées partielles (selon x^2, y^2, xy et yx). En pratique, on utilise l'*opérateur Laplacien* qui fait la somme des deux dérivées partielles principales.

$$\nabla^2 I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

Marr et Hildreth ont proposé en 1980 un opérateur de détection de contours basé sur la détection des passages par zéro de la dérivée seconde (opérateur laplacien). L'image est

préalablement filtrée par un opérateur gaussien filtre passe-bas G de manière à réduire la sensibilité au bruit lors de la détection des passages par zéro de l'opérateur laplacien.

On obtient alors, grâce aux propriétés de l'opérateur produit de convolution :

$$\nabla^2 (I * G) = \nabla^2 (I) * G = I * \nabla^2 (G)$$

avec

$$G(r) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{\frac{-r^2}{2\sigma^2}} \text{ où } r = \sqrt{x^2 + y^2}$$

dont le laplacien est :

$$\nabla^2 G(r) = \frac{-1}{\pi\sigma^4} \left(1 - \frac{r^2}{2\sigma^2}\right) \exp^{-\frac{r^2}{2\sigma^2}}$$

Le paramètre σ (écart-type de la gaussienne) est fonction du niveau de lissage qu'on souhaite appliquer à l'image initiale, donc il sera fixé en fonction du niveau de bruit présent dans l'image. Ce paramètre a également une influence sur l'échelle à laquelle les contours sont détectés. Son choix est délicat car on ne connaît souvent pas le niveau du bruit, ni l'échelle à laquelle on souhaite détecter les contours.

En coordonnées cartésiennes le filtre LoG (Laplacian of Gaussian) $h(x, y)$ est caractérisé par la réponse impulsionnelle suivante :

$$h(x, y) = \nabla^2 g(x, y) = \frac{x^2 + y^2 - 2\sigma^2}{\sigma^4} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

Le filtre ainsi défini a la forme d'un chapeau mexicain, d'où son nom : "Mexican hat".

24. Ecrire le programme d'un filtre LoG en utilisant la commande

    `edge(Image,'log',sigma)% filtrage LoG`

    Tester ce détecteur de contour sur les images `cameraman.tif` et `lena256.tif`.

25. Ecrire le programme d'un opérateur détectant les passage par zéro d'une image filtrée par h en utilisant la commande

    `edge(Image,'zerocross',h) % methode du zero-crossing, filtre h`

    Que choisir comme masque h ? Tester ce détecteur de contour sur les images `cameraman.tif` et `lena256.tif`.

26. Comparer les 4 détecteurs de contour (*i.e.,* opérateurs NET-FLOU (net moins flou), DOG, zero-crossing et LoG) sur les images `cameraman.tif` et `lena256.tif`. Lequel sera le plus performant en présence de bruit ?

4 Application du filtrage : débruitage

4.1 Débruitage 1 : filtres moyenneur et filtre médian

Une des fonctions les plus importantes du traitement numérique de l'image est le *débruitage*, couramment utilisé en astronomie, en photographie en faible lumière, en imagerie médicale, etc.

On a vu qu'un filtre passe-bas (par exemple un simple filtre moyenneur) permet de réduire le bruit haute fréquence, mais au prix d'un lissage important des contours et des textures présents dans l'image, autrement dit, au prix de l'introduction d'un flou sur l'image. Il existe un « filtre » non linéaire, *i.e.,* qui ne peut être décrit par une opération de convolution, qui permet de réduire certains types de bruit en lissant très peu les contours : il s'agit du filtre médian. Ce filtre est particulièrement intéressant pour éliminer les bruits impulsionnels de type *sel et poivre*, c'est à dire un bruit affectant un faible nombre de pixels, mais en modifiant très fortement l'intensité. Ce filtre de taille $(2n+1) \times (2n+1)$ affecte au pixel central la valeur médiane des pixels couverts par le masque. Prenons un exemple pour bien comprendre. Soit un voisinage 3×3 pixels :

$$\begin{pmatrix} .7 & .5 & .3 \\ .4 & .2 & .7 \\ .6 & .5 & .2 \end{pmatrix}$$

Le filtre médian range par ordre croissant les intensités du voisinage :

$$.2 \quad .2 \quad .3 \quad .4 \quad \mathbf{.5} \quad .5 \quad .6 \quad .7 \quad .7$$

Il affecte au pixel central (qui valait initialement .2) l'intensité médiane, soit .5 sur cet exemple. Autrement dit, le filtre médian retient une valeur d'intensité telle que dans le voisinage, il y ait autant d'intensités supérieures que d'intensités inférieures à cette valeur.

27. Montrer la supériorité du filtre médian (utiliser la commande `medfilt2`) sur un filtre moyenneur dans le cas d'un bruit impulsionnel, en vous inspirant du programme suivant :

```
I=imread('house.jpg');
I=rgb2gray(I);
I=double(I)/255.0;
% creation de l image bruitee
Ibruitee=imnoise(I,'salt & pepper');
% debruitage par le filtre median
Imed = medfilt2(Ibruitee);
% débruitage par un filtre moyenneur
pb=ones(5,5)/25;
Ipb=filter2(pb,Ibruitee);
```

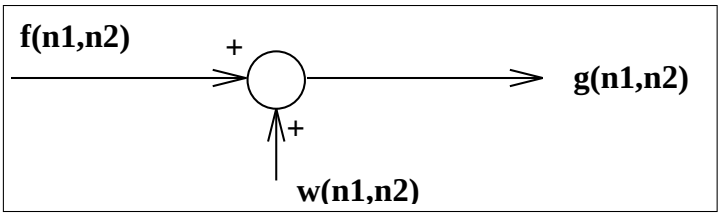
28. Même question avec une image bruitée par un bruit additif blanc gaussien
`bruit=imnoise(ones(size(I))/2,'gaussian',0,0.04);`

29. Calculer la valeur du rapport signal à bruit (Signal Noise Ratio) en identifiant la puissance du bruit avec sa variance :

$$SNR = \frac{P_{signal} = \frac{1}{Card(I)} \sum_{m,n} I^2(m,n)}{\sigma^2_{bruit}}$$

4.2 Débruitage 2 : Filtrage optimal - filtre de Wiener

Le filtre moyenneur est un filtre passe-bas non optimal (c'est à dire non adapté spécifiquement au signal et au bruit observé). Il existe un filtre optimal qui adapte la forme de sa réponse en fréquence au contenu fréquentiel du signal et du bruit : le filtre de Wiener. On considère un bruit additif :



$$\mathbf{g(n_1, n_2)} \;=\; \mathbf{f(n_1, n_2)} \;+\; \mathbf{w(n_1, n_2)}$$

où : f est l'image (idéale) avant dégradation, g est l'image dégradée (observée), w est le bruit additif.

- Hypothèses :
  — f et w sont des processus aléatoires stationnaires, non corrélés ;
  — les densités spectrales de puissance (d.s.p.) $S_f(\nu_1, \nu_2)$ et $S_w(\nu_1, \nu_2)$ de f et w sont supposées *connues*.
- Le filtre de Wiener est le système linéaire (le filtre) de réponse impulsionnelle h_{wiener} qui minimise l'erreur quadratique moyenne (EQM) :

$$\hat{f}(n_1, n_2) \;=\; g(n_1, n_2) * h_{wiener}$$
$$\mathbb{E}[(f(n_1, n_2) - \hat{f}(n_1, n_2))^2] \text{ minimum}$$

- La réponse en fréquence du filtre de Wiener est donnée par :

$$H_{wiener}(\nu_1, \nu_2) \;=\; \frac{1}{1 \;+\; \frac{S_w(\nu_1, \nu_2)}{S_f(\nu_1, \nu_2)}}$$

On suppose que le bruit est un bruit blanc gaussien discret de variance σ^2.
Le programme MatLab `gauss-wiener` compare les résultats d'un filtrage par un filtre passe-bas gaussien avec un filtre de Wiener optimal. Un signal 1D (piecewise-linear) et une image 2D (lena), sont filtrées. On considère ici un bruit blanc additif gaussien discret de variance σ^2. On suppose que la densité spectrale de puissance $S_f(\nu_1, \nu_2)$ du signal non bruitée *est connue*. Notons que ceci n'est généralement pas le cas.

30. Lancer le programme, comprendre son fonctionnement. Noter et interpréter les différents résultats (σ^2, PSNR, SNR). Les images originales et débruitées se trouvent dans `./images/gauss-wiener`. Quelle est la densité spectrale de puissance du bruit ? Donner sa valeur. Commenter les résultats et les différents rapports signal à bruit mesurés.

31. Identifier dans le code Matlab la partie correspondant au filtrage de Wiener (reproduire les lignes correspondantes).

32. On suppose maintenant que la densité spectrale de puissance du signal $S_f(\nu_1, \nu_2)$ *n'est pas connue*. On la remplace par la d.s.p. d'une autre image non bruitée, de la même « classe » d'images. Insérer dans le code Matlab un tel filtrage de Wiener, en utilisant pour modèle pour l'image non bruitée l'image `mandrill.png`. Observer l'image filtrée et comparer avec le filtrage de Wiener idéal. Quels sont les rapports signal à bruit obtenus dans ce cas. Conclusion ?

4.3 Débruitage 3 : Méthode par ondelettes

Cette section est abordée plus loin comme application de la transformée en Ondelettes 2D. cf 6.3

4.4 Débruitage 4 : Méthode des moyennes non locales

La méthode des moyennes locales (MNL) (*Non Local Means*, Buades 2005) est actuellement considérée comme une des méthodes les plus efficaces de débruitage d'images. Contrairement au filtre moyenneur qui calcule une moyenne locale dans une fenêtre centrée sur le pixel courant, la méthode des moyennes non locales calcule une moyenne intégrant des fenêtres de toute l'image, qui présentent une similarité avec la fenêtre locale. Ceci permet de lisser davantage le bruit. La méthode est présentée succinctement sur `http://en.wikipedia.org/wiki/Non-local_means_filter`. Une implantation en ligne est par ailleurs proposée sur le site de *Image Processing On Line* (IPOL) : `http://www.ipol.im/pub/art/2011/bcm_nlm/`.

33. Extraire une zone 128x128 de l'image `lena` sur le site d'IPOL. Lui ajouter un bruit blanc de même variance que précédemment. Comparer le résultat du filtrage (PSNR et visuellement) avec les différents résultats précédents. Conclusion ? NB : les temps de calculs sur le site sont aléatoires et plutôt long . Inutile de charger une grande image.

34. Une variante complexe de la méthode MNL est l'algorithme BM3D qui a été reconnu récemment comme l'algorithme de débruitage le plus efficace, dans un benchmark le comparant aux autres méthodes classiques (Wiener, Ondelettes, MNL, etc.). On en trouvera une description et une implantation sur IPOL : `http://www.ipol.im/pub/art/2012/l-bm3d/`.

    Comme précédemment, charger l'image `lena` sur le site d'IPOL. Lui ajouter un bruit blanc de même variance que précédemment. Comparer le résultat du filtrage avec les différents résultats précédents. Conclusion ?

5 Echantillonnage, interpolation et transformations spatiales

5.1 Echantillonnage : illustration de l'effet de repliement spectral

Le problème de repliement spectral apparaît lors de la discrétisation d'un signal continu (cf question 35) ou lors du sous-échantillonnage d'un signal discret (cf question 36).

35. Considérons la fonction cosinus radiale, qui est à symétrie circulaire : à un point du plan (x, y), elle associe la valeur $cos(2\pi f_0 \sqrt{x^2 + y^2})$. Sachant que sa transformée de Fourier, également à symétrie circulaire, vaut 0 partout sauf à l'intérieur du disque de rayon f_0, en déduire comment discrétiser ce signal. Effectuer des tests avec le fichier `TP2_discretisation.m`. Observer les problèmes de repliement spectral lorsque la fréquence d'échantillonnage est trop faible.
36. Supposons maintenant que cette fonction ait été convenablement échantillonnée (c'est-à-dire en respectant les conditions de Shannon). Cependant, la taille de l'image étant trop importante, on désire sous-échantillonner cette image. Quelle problème cela peut-il poser si aucune précaution n'est prise ? Analyser le fichier `TP2_sousechantillonnage.m`. Quelle solution est proposée ?

5.2 Interpolation et transformations spatiales

Lorsque l'on réalise une transformation spatiale simple sur une image, telle qu'une translation (de pas non entier) ou une rotation (d'angle différent de $k\frac{\pi}{2}$) , il est nécessaire de rééchantillonner l'image pour produire l'image transformée, car les points transformés ne tombent en général pas sur la grille. Une transformation géométrique appliquée à une image discrète nécessite donc une interpolation des niveaux de gris, afin de calculer les valeurs du signal entre les points d'échantillonnage.
Une transformation géométrique dans le plan peut s'exprimer sous la forme générale :

$$\begin{aligned} x' &= a(x, y) \\ y' &= b(x, y) \end{aligned}$$

On notera :

$$\begin{aligned} x &= c(x', y') \\ y &= d(x', y') \end{aligned}$$

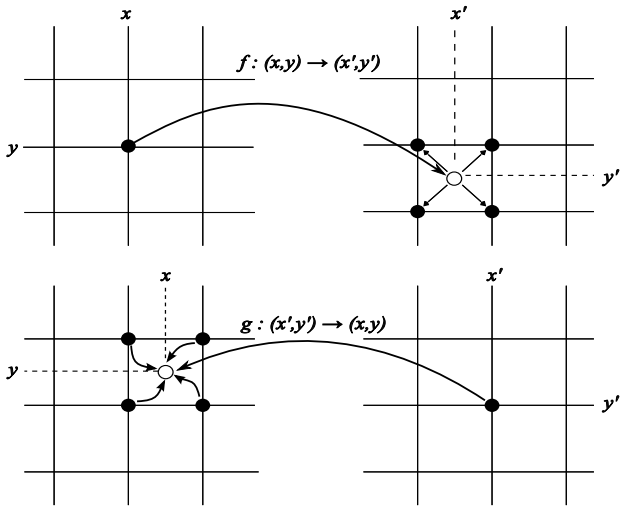
Si g est l'image transformée de f, on notera :

$$g(x', y') = F(c(x', y'), d(x', y'))$$

la transformation inverse (lorsqu'elle existe).
Il existe deux approches pour appliquer une transformation spatiale sur une image :

- Utilisation de la transformation directe. On parcourt l'espace de départ (points (x, y)) sur la grille d'échantillonnage (valeurs entières). Le point (x', y') n'appartient en général pas à la grille d'échantillonnage (valeurs non entières). Il faut alors distribuer la valeur $g(x', y') = f(x, y)$ sur les pixels de l'espace d'arrivée. Cette opération n'est pas simple (certains points de l'espace d'arrivée peuvent ne pas être atteints : présence de trous dans l'image résultante, certains points se voient affectées plusieurs valeurs différentes, etc.)
- Utilisation de la transformée inverse. La méthode la plus souvent utilisée est de parcourir l'espace d'arrivée sur la grille d'échantillonnage et d'utiliser la transformée inverse : le point (x', y') est donné sur la grille (valeurs entières), on calcule le point (x, y) dont est issu (x', y') en utilisant la transformée inverse.
  On a alors : $g(x', y') = F(c(x', y'), d(x', y'))$
  En général $(c(x', y'), d(x', y'))$ n'est pas sur la grille d'échantillonnage (valeurs non entières). On calcule alors $f(c(x', y'), d(x', y'))$ par interpolation par rapport à ses plus proches voisins.

La figure suivante représentant $f(x, y)$ et $g(x', y')$ dans les deux situations :



5.2.1 Interpolation au plus proche voisin

La méthode d'interpolation la plus simple est l'interpolation au plus proche voisin : on affecte à $f(c(x', y'), d(x', y'))$ la valeur du pixel le plus proche de $(c(x', y'), d(x', y'))$.
Le script MatLab suivant réalise une rotation d'angle θ de l'image `I` par rapport à son centre, avec une interpolation au plus proche voisin :

```
1 theta = pi/5;
2 [l,c] = size(I);
3 [Y,X] = meshgrid(1:c,1:l);
4 ctre = (1 + [l c]) ./ 2;
5 Jcplx = X-ctre(1) + j * (Y-ctre(2));
6 Jcplx = Jcplx .* exp(-j * theta);
7 Jx = real(Jcplx) + ctre(1);
8 Jy = imag(Jcplx) + ctre(2);
9 Ji = round(Jx);
10 Jj = round(Jy);
11 mask = (Ji >0) & (Ji < l) & (Jj>0) & (Jj<c);
```

```
12 indices = sub2ind(size(I),Ji(mask),Jj(mask));
13 J = zeros(size(I));
14 J(mask) = I(indices);
```

37. A l'aide de la documentation sur la fonction `meshgrid`, expliquer à quoi correspondent les variables `X` et `Y`.

    Le script utilise la rotation dans le plan complexe $f : z \to f(z)$, avec $z = x + iy$.
    A quelle(s) ligne(s) a-t-elle lieu ? Donner l'expression analytique de la transformation utilisée.

    A quelle(s) ligne(s) a lieu l'interpolation au plus proche voisin ? Quelle est la fonction utilisée ?

    Les points (x', y') qui ont des antécédents (x, y) qui sortent de l'image de départ sont mis à la valeur 0. A cette fin, quelle variable est utilisée ?

38. Appliquer successivement 10 rotations d'angle $\frac{\pi}{5}$ à l'image Lena (lena-gray.jpg). Comparer l'image résultante avec l'image initiale. Commenter.

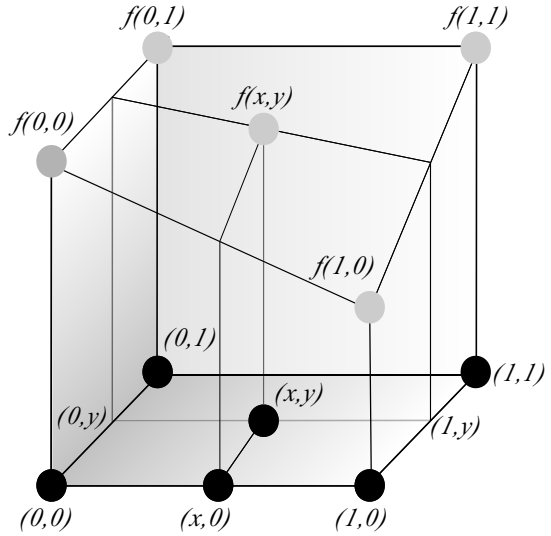
5.2.2 Interpolation bilinéaire

L'interpolation bilinéaire est une généralisation en deux dimensions de l'interpolation linéaire, bien connue pour les signaux 1D. Soit $f(x, y)$ une fonction à deux variables. On suppose que l'on connaît la fonction f aux quatre coins du carré $(0, 0)$, $(0, 1)$, $(1, 0)$, $(1, 1)$. Pour des valeurs non entières de (x, y) dans $[0, 1] \times [0, 1]$ on interpole $f(x, y)$ par la surface règlée :

$$f(x, y) = \alpha x \ + \ \beta y \ + \ \gamma x\, y \ + \ \delta$$

Montrer que l'on a :

$$\begin{aligned} \alpha &= f(1,0) - f(0,0) \\ \beta &= f(0,1) - f(0,0) \\ \gamma &= f(1,1) + f(0,0) - f(0,1) - f(1,0) \\ \delta &= f(0,0) \end{aligned}$$



39. Comment peut-on généraliser cette formule d'interpolation en un point (x, y) quelconque du plan ?

    A l'aide du script de la question précédente et des figures ci-dessus, modifier et compléter le programme MatLab `rotation_bilineaire.m`.

    Ecrire un programme qui applique successivement 10 rotations d'angle $\frac{\pi}{5}$, dans le cas de l'interpolation bilinéaire.

Comparer l'image résultante avec l'image initiale. Commenter et comparer avec l'interpolation aux plus proches voisins.
Proposer une méthode pour évaluer quantitativement la différence entre deux images, par exemple sur un carré de 100 pixels de coté situé au centre de l'image.

5.2.3 Interpolation bicubique

Cette méthode utilise des polynômes de la forme :

$$f(x,y) = \sum_{i=1}^{3} \sum_{j=1}^{3} \alpha_{ij} x^i \, y^j$$

Les coefficients α_{ij} sont calculés à partir des 16 plus proches voisins (on ne détaillera pas ici ce calcul). L'interpolation obtenue est de très bonne qualité.

40. Utiliser la fonction MatLab `imrotate` pour appliquer successivement 10 rotations d'angle $\frac{\pi}{5}$, dans le cas de l'interpolation bicubique. Comparer l'image résultante avec l'image initiale. Comparer avec les autres méthodes.

5.3 Etude des interpolateurs à base de B-splines

Les meilleurs interpolateurs actuels sont les interpolateurs à base de B-splines introduits par Thévenaz, Blu et Unser (EPFL, Lausanne) en 2000. Ces interpolateurs ont la particularité de ne pas être "interpolants"', c'est-à-dire que la fonction interpolée ne coïncide pas exactement avec la fonction de départ sur la grille d'échantillonnage. L'abandon de cette contrainte (imposée dans les interpolateurs classiques), introduit de nouveaux degrés de liberté qui permettent un gain substantiel en qualité. Leur implantation est assez complexe et ne sera pas étudiée ici. Pour les détails on se référera à l'article :
P. Thévenaz, T. Blu, M. Unser, Interpolation Revisited, IEEE Trans Medical Imaging, Vol. 19, No. 7, July 2000. Cet article figure sur le site du TP.

Dans le cadre de ce TP on utilisera une applet disponible sur le site de l'EPFL (Biomedical Imaging Group) (permettant de comparer différents interpolateurs).
`http://bigwww.epfl.ch/demo/jaffine/index.html`

41. Lancer l'applet Java. Appliquer à nouveau 10 rotations d'angle $\frac{\pi}{5}$ à l'image Lena. L'applet donne les valeurs du "rapport signal à bruit" pour les différents interpolateurs. Comment se définit le rapport signal à bruit ? (A quoi correspond le bruit ?)
Estimer, grâce à l'applet, le rapport signal à bruit pour les différents interpolateurs.
Comparer les interpolateurs à base de B-splines avec les autres. Conclusion ?

5.4 References complémentaires

IPOL fournit également d'intéressants contenus, articles and demos sur les methodes d'interpolation : `http://www.ipol.im/pub/art/2011/g_lmii/`

6 Ondelettes, représentations multirésolutions et compression

Cette partie, en anglais, est directement adaptée du très recommandable site « Numerical Tours » de Gabriel Peyré :
`http://www.numerical-tours.com/`
`https://github.com/gpeyre/numerical-tours/tree/gh-pages`
`http://www.gpeyre.com`
qui a pour objectif l'exploration, en ligne, des méthodes avancées de traitement du signal et des images, en s'appuyant sur Matlab et Python.
Les toolboxes `toolbox_general` et `toolbox_signal` doivent avoir préalablement été installées et incluses dans les chemins Matlab. Inclusion des toolboxes dans le PATH :

```
addpath('toolbox_signal');
addpath('toolbox_general');
```

6.1 1-D Daubechies Wavelets

Ce premier tour explore la transformée en ondelettes 1D, à travers les ondelettes (classiques) de la mathématicienne I. Daubechies. Ces ondelettes ont été calculées de façon à :
- être orthogonales ;
- de support compact ;
- avoir k moments nuls.

Le support (compact) de l'ondelette est le plus petit possible pour un nombre donné k de moments nuls. Un nombre élevé de moments nuls permet de représenter un signal régulier (polynomial) avec un faible nombre de coefficients (donc de mieux le compresser).
`https://en.wikipedia.org/wiki/Daubechies_wavelet`

Toutes les ondelettes classiques peuvent être explorées sur le "wavelet browser" :
`http://wavelets.pybytes.com/`
Observer l'allure des ondelettes de Daubechies (d'ordre $k = 1, \ldots, 20$) : ondelettes, fonctions d'échelles et filtres numériques h et g (filtres miroirs en quadrature) permettant de calculer la transformée en ondelettes. On remarquera que la taille des filtres $p = 2k$. On remarquera également que Daubechies pour $k = 1$ correspond à l'ondelette de Haar, pour laquelle les approximations multirésolutions sont des fonctions constantes par morceaux.

Inclusion des solutions des exercices dans le PATH :
`addpath('solutions/wavelet_3_daubechies1d/');`

42. **Wavelet filters**
The 1-D wavelet transform of a continuous signal $f(x)$ computes the set of inner products

$$d_j[n] = \langle f, \psi_{j,n} \rangle$$

for scales $j \in \mathbb{Z}$ and position $n \in \mathbb{Z}$. The wavelet atoms are defined by scaling and translating a mother atom $\psi(x)$:

$$\psi_{j,n}(x) = \frac{1}{2^j}\psi\left(\frac{x - 2^j n}{2^j}\right).$$

Associated to this oscillating (high pass) wavelet function ψ is a non-oscillating (low pass) scaling function ϕ. The fast wavelet transform algorithm does not make use of the wavelet and scaling functions, but of the filters h and g that characterize their interaction :

$$g[n] = \frac{1}{\sqrt{2}}\langle\psi(x/2), \phi(x-n)\rangle \quad \text{and} \quad h[n] = \frac{1}{\sqrt{2}}\langle\phi(x/2), \phi(x-n)\rangle.$$

The simplest filters are the Haar filters

$$h = [1, 1]/\sqrt{2} \quad \text{and} \quad g = [-1, 1]/\sqrt{2}.$$

Daubechies wavelets extends the Haar wavelets by using longer filters, that produce smoother scaling functions and wavelets. Furthermore, the larger the size $p = 2k$ of the filter, the higher is the number k of vanishing moments (NDLR : moments nuls cités ci-dessus).

A high number of vanishing moments allows to better compress regular parts of the signal. However, increasing the number of vanishing moments also increases the size of the support of the wavelets, which can be problematic in part where the signal is singular (for instance discontinuous).

Choosing the best wavelet, and thus choosing k, that is adapted to a given class of signals, thus corresponds to a tradeoff between efficiency in regular and singular parts.

- The filter with $k = 1$ vanishing moments corresponds to the Haar filter ($p = 2$).
- The filter with $k = 2$ vanishing moments corresponds to the famous D4 wavelet, which compresses perfectly linear signals ($p = 4$).
- The filter with $k = 3$ vanishing moments compresses perfectly quadratic signals ($p = 6$).

Set the support size. To begin, we select the D4 filter. (*Try other values for p afterwards*).

```
p = 4;
```

Create the low pass filter h and the high pass g.

```
[h,g] = compute_wavelet_filter('Daubechies',p);
```

Note that the high pass filter g is computed directly from the low pass filter as :

$$g[n] = (-1)^{1-n}h[1-n]$$

Display.

```
disp(['h filter = [' num2str(h) ']']);
disp(['g filter = [' num2str(g) ']']);
```

h filter = [0 0.48296 0.83652 0.22414 -0.12941]

g filter = [0 -0.12941 -0.22414 0.83652 -0.48296]

43. **Up and Down Filtering**

The basic wavelet operation is low/high filtering, followed by down sampling.

Starting from some 1-D signal $f \in \mathbb{R}^N$, one thus computes the low pass signal $a \in \mathbb{R}^{N/2}$ and the high pass signal $d \in \mathbb{R}^{N/2}$ as

$$a = (f * h) \downarrow 2 \quad \text{and} \quad d = (f * g) \downarrow 2$$

where the sub-sampling is defined as

$$(u \downarrow 2)[k] = u[2k].$$

Create a random signal $f \in \mathbb{R}^N$.

```
N = 256;
f = rand(N,1);
```

Low/High pass filtering followed by sub-sampling.

```
a = subsampling( cconvol(f,h) );
d = subsampling( cconvol(f,g) );
```

For orthogonal filters, the reverse of this process is its dual (aka its transpose), which is upsampling followed by low/high pass filtering with the reversed filters and summing :

$$(a \uparrow 2) * \tilde{h} + (d \uparrow 2) * \tilde{g} = f$$

where $\tilde{h}[n] = h[-n]$ (computed modulo N) and $(u \uparrow 2)[2n] = u[n]$ and $(u \uparrow 2)[2n+1] = 0$.

Up-sampling followed by filtering :

```
f1 =  cconvol(upsampling(a),reverse(h)) + cconvol(upsampling(d),reverse(g));
```

Check that we really recover the same signal.

```
disp(strcat(['Error |f-f1|/|f| = ' num2str(norm(f-f1)/norm(f))]));
```

Error |f-f1|/|f| = 5.4681e-13

44. **Forward Wavelet Transform**

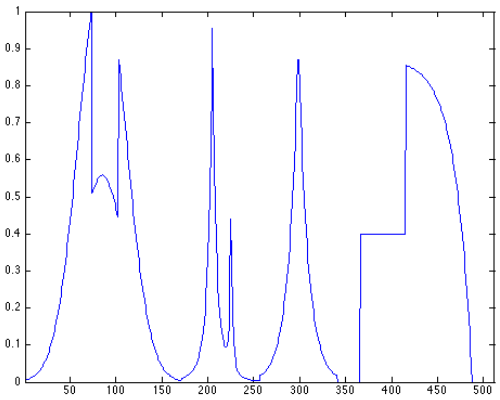
The set of wavelet coefficients are computed with a fast algorithm that exploits the embedding of the approximation spaces V_j spanned by the scaling function $\{\phi_{j,n}\}_n$.

First we load a 1-D signal.

```
name = 'piece-regular';
N = 512;
f = rescale( load_signal(name, N) );
```

Display it.

```
figure;
plot(f);
axis('tight');
```



We will store all the transformed coefficients d_j in a single vector fw. This vector is initialized as `f` and the left sub-part `fw(1:1^j)` of `fw` will be retransformed at each iteration for a decreasing scale index j.

Initialize the result vector.

```
fw = f;
```

Initialize the scale index j as $j = J = \log_2(N) - 1$.

```
j = log2(N)-1;
```

The wavelet transform of f is computed by using intermediate discretized low resolution signals obtained by projection on the spaces V_j :

$$a_j[n] = \langle f, \phi_{j,n} \rangle.$$

The algorithm processes by moving from scale j to the coarser scale $j-1$ using the filtering+subsampling :

$$a_{j-1} = (a_j * h) \downarrow 2 \quad \text{and} \quad d_{j-1} = (a_j * g) \downarrow 2$$

Retrieve the coefficients a_j from the variable fw and store them in the variable a1

```
a1 = fw(1:2^(j+1));
```

Apply high and low filtering+subsampling to obtain a_{j-1} and d_{j-1} (stored in a and d).

```
a = subsampling(cconvol(a1,h));
d = subsampling(cconvol(a1,g));
```

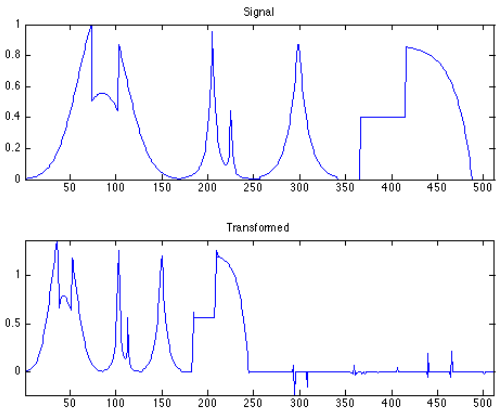
Note : `subsampling(A)` is equivalent to `A(1:2:end)`.

Concatenate them to get the result and store it in `fw`.

```
fw(1:2^(j+1)) = cat(1, a, d );
```

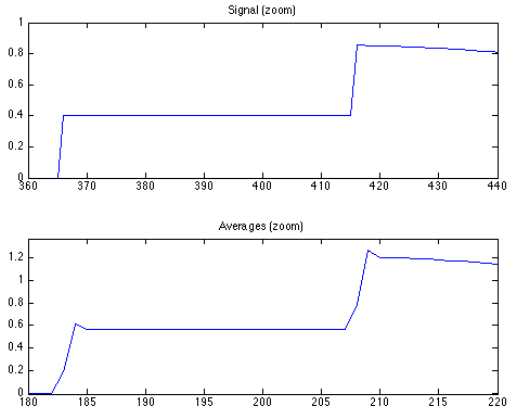
Display the result of the first step of the transform.

```
figure;
subplot(2,1,1);
plot(f); axis('tight');
title('Signal');
subplot(2,1,2); plot(fw);
axis('tight');
title('Transformed');
```



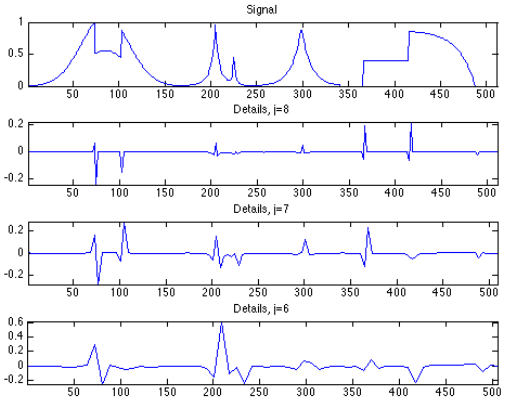
Display the signal and its coarse coefficients.

```
s = 400;
t = 40;
figure;
subplot(2,1,1);
plot(f,'.-'); axis([s-t s+t 0 1]);
title('Signal (zoom)');
subplot(2,1,2);
plot(a,'.-');
axis([(s-t)/2 (s+t)/2 min(a) max(a)]);
title('Averages (zoom)');
```



Exercice 1 : (check the solution) Implement a full wavelet transform that extract iteratively wavelet coefficients, by repeating these steps. Take care of choosing the correct number of steps.

```
figure;
exo1;
```



Check that the transform is orthogonal, which means that the energy of the coefficient is the same as the energy of the signal.

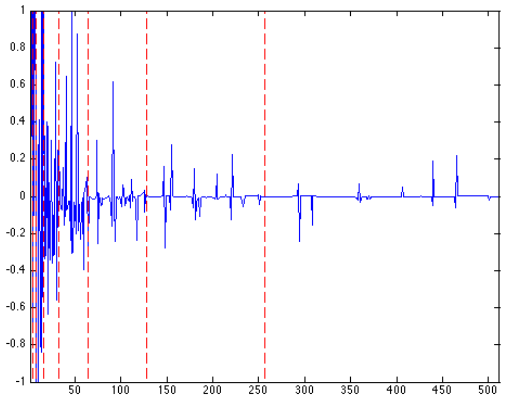
```
disp(strcat(['Energy of the signal       = ' num2str(norm(f).^2,3)]));
disp(strcat(['Energy of the coefficients = ' num2str(norm(fw).^2,3)]));
```

Energy of the signal = 88.6

Energy of the coefficients = 88.6

We display the whole set of coefficients `fw`, with red vertical separator between the scales. Can you recognize where are the low frequencies and the high frequencies ? You can use the function `plot_wavelet` to help you.

```
figure;
plot_wavelet(fw);
axis([1 N -1 1]);
```



45. **Backward Wavelet Transform**

The backward wavelet transform reconstructs a signal f from a set of wavelet coeffcients $\{d_j[n]\}_{j,n}$. For continuous functions, it corresponds to the following reconstruction formula :

$$f(x) = \sum_{j,n} d_j[n]\psi_{j,n}(x).$$

For discrete signal, it reconstructs a signal $f \in \mathbb{R}^N$ by inverting the wavelet filtering/sub-sampling steps.

It starts from the coarsest scale $j = 0$, where $a_0 \in \mathbb{R}$ is the single remaining coefficient.

The algorithm processes by moving from scale j to the finer scale $j + 1$ using the up-sampling/filtering :

$$a_{j+1} = (a_j \uparrow 2) * \tilde{h} + (d_j \uparrow 2) * \tilde{g}$$

Initialize the signal to recover `f1` as the transformed coefficient, and select the smallest possible scale.

```
f1 = fw;
j = 0;
```

Retrieve coarse and detail coefficients in the vertical direction.

```
a = f1(1:2^j);
d = f1(2^j+1:2^(j+1));
```

Perform the up-sampling/filtering and summation :

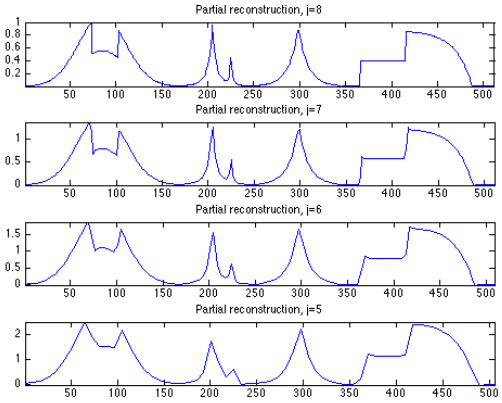
```
a = cconvol(upsampling(a,1),reverse(h),1) + cconvol(upsampling(d,1),reverse(g),1);
```

Replace the coefficients at the correct locations.

```
f1(1:2^(j+1)) = a;
```

Exercice 2 : Write the inverse wavelet transform that computes `f1` from the coefficients `fw`.

```
figure; exo2;
```



Check that we have correctly recovered the signal.

```
disp(strcat(['Error |f-f1|/|f| = ' num2str(norm(f-f1)/norm(f))]));
```

Error |f-f1|/|f| = 4.0189e-12

46. **Daubechies Wavelets Approximation**

A non-linear approximation of signal f is obtained by thresholding low amplitude wavelet coefficients, retaining only the M largest coefficients, larger than a threshold T. This defines the best M-terms approximation f_M of f :

$$f_M = \sum_{|\langle f,\psi_{j,n}\rangle|>T} \langle f,\psi_{j,n}\rangle \psi_{j,n}.$$

Set the threshold value.

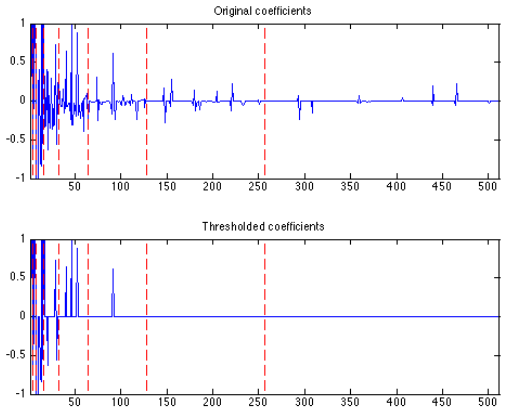
```
T = .5;
```

Coefficients `fw(i)` smaller in magnitude than T are set to zero.

```
fwT = fw .* (abs(fw)>T);
```

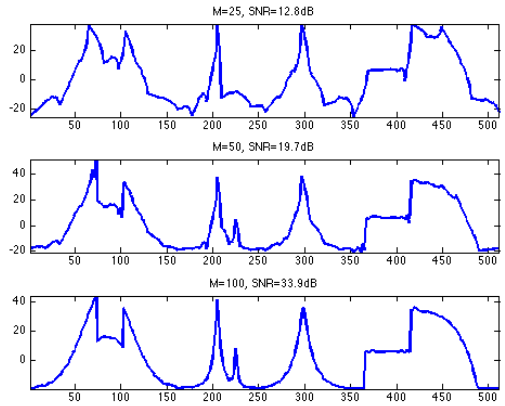
Display the coefficients before and after thresholding.

```
figure;
subplot(2,1,1);
plot_wavelet(fw);
axis([1 N -1 1]);  title('Original coefficients');
subplot(2,1,2);
plot_wavelet(fwT);
axis([1 N -1 1]);  title('Thresholded coefficients');
```



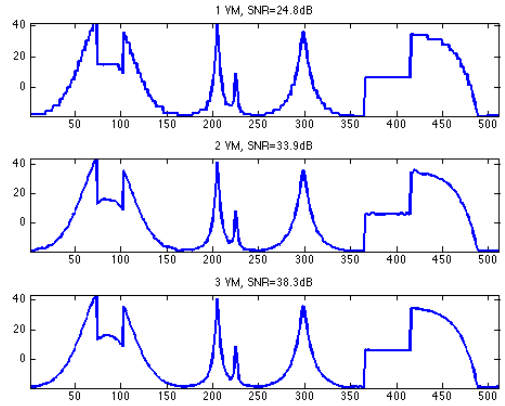
Exercice 3 : Find the threshold T to obtain a given number M of non zero coefficients. Try for an increasing number $M = 25,\ 50,\ 100$ of remaining coefficients. What is the initial number of non-threshold coefficients ? What is the "compression rate" in each case ?.

```
figure; exo3m;
```



Exercice 4 : Try with different kind of wavelets, with an increasing number of vanishing moments.

```
figure; exo4;
```



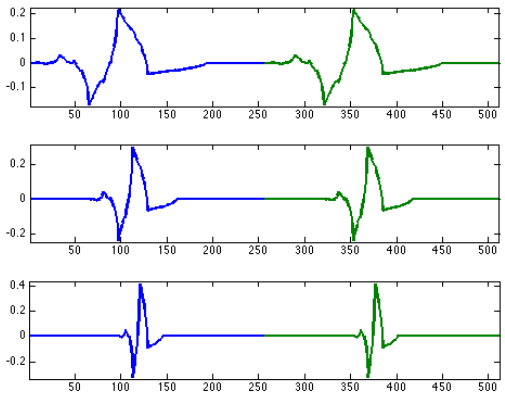
47. **The Shape of a Wavelet** How can I see and plot the wavelet signal used in the decomposition , i. e the wavelets components ? Here is the trick : A wavelet coefficient is an inner

product $d_j[n] = \langle f, \psi_{j,n} \rangle$ with a wavelet atom $\psi_{j,n}$.
A wavelet atom ψ_{j_0,n_0} can be computed by applying the inverse wavelet transform to coefficients $\{d_j[n]\}_{j,n}$ such that

$$d_j[n] = \begin{cases} 1 & \text{if } j = j_0 \text{ and } n = n_0, \\ 0 & \text{otherwise.} \end{cases}$$

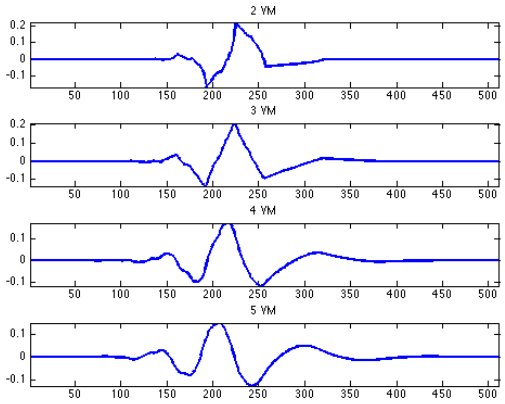
Exercice 5 : Compute wavelets at several positions and scales.

```
figure; exo5m;
```



Exercice 6 : Display Daubechies wavelets with an increasing number of vanishing moments.

```
figure; exo6m;
```



6.2 2-D Daubechies Wavelets

Include into PATH variable :

```
addpath('solutions/wavelet_4_daubechies2d/');
```

48. **Wavelets Filters**

The 2-D wavelet transform of a continuous image $f(x),\ x = (x_1, x_2)$, computes the set of inner products

$$d_j^k[n] = \langle f, \psi_{j,n}^k \rangle$$

for scales $j \in \mathbb{Z}$, position $n \in \mathbb{Z}^2$ and orientation $k \in \{H, V, D\}$. The wavelet atoms are defined by scaling and translating three mother atoms $\{\psi^H, \psi^V, \psi^D\}$:

$$\psi_{j,n}^k(x) = \frac{1}{2^j}\psi^k\left(\frac{x - 2^j n}{2^j}\right)$$

These oriented wavelets are defined by a tensor product of a 1-D wavelet function $\psi(t)$ and a 1-D scaling function $\phi(t)$

$$\psi^H(x) = \phi(x_1)\psi(x_2), \quad \psi^V(x) = \psi(x_1)\phi(x_2) \ \text{ and } \ \psi^D(x) = \psi(x_1)\psi(x_2).$$

The fast wavelet transform algorithm does not make use of the wavelet and scaling functions, but of the filters h and g that caracterize their interaction :

$$g[n] = \frac{1}{\sqrt{2}}\langle \psi(t/2), \phi(t-n) \rangle \ \text{ and } \ h[n] = \frac{1}{\sqrt{2}}\langle \phi(t/2), \phi(t-n) \rangle.$$

The simplest filters are the Haar filters

$$h = [1, 1]/\sqrt{2} \ \text{ and } \ g = [-1, 1]/\sqrt{2}.$$

Daubechies wavelets extends the haar wavelets by using longer filters, that produce smoother scaling functions and wavelets. Furthermore, the larger the size $p = 2k$ of the filter, the higher is the number k of vanishing moment. A high number of vanishing moments allows to better compress regular parts of the signal. However, increasing the number of vanishing moments also inceases the size of the support of the wavelets, wich can be problematic in part where the signal is singular (for instance discontinuous). Choosing the best wavelet, and thus choosing k, that is adapted to a given class of signals, thus corresponds to a tradeoff between efficiency in regular and singular parts.

- The filter with $k = 1$ vanishing moments corresponds to the Haar filter.
- The filter with $k = 2$ vanishing moments corresponds to the famous D4 wavelet, which compresses perfectly linear signals.
- The filter with $k = 3$ vanishing moments compresses perfectly quadratic signals.

Set the support size. We select the D4 filter.

```
p = 4;
```

Create the low pass filter h and the high pass g.

```
[h,g] = compute_wavelet_filter('Daubechies',p);
```

Note that the high pass filter g is computed directly from the low pass filter as :

$$g[n] = (-1)^{1-n} h[1-n]$$

Display.

```
disp(['h filter = [' num2str(h) ']']); disp(['g filter = [' num2str(g) ']']);
```

h filter = [0 0.48296 0.83652 0.22414 -0.12941]
g filter = [0 -0.12941 -0.22414 0.83652 -0.48296]

49. **Forward 2-D Wavelet transform**

The set of wavelet coefficients are computed with a fast algorithm that exploits the embedding of the approximation spaces V_j spanned by the scaling function $\{\phi_{j,n}\}_n$ defined as

$$\phi_{j,n}(x) = \frac{1}{2^j}\phi^0\left(\frac{x - 2^j n}{2^j}\right) \quad \text{where} \quad \phi^0(x) = \phi(x_1)\phi(x_2).$$

The wavelet transform of f is computed by using intermediate discretized low resolution images obtained by projection on the spaces V_j :

$$a_j[n] = \langle f, \phi_{j,n} \rangle.$$

First we load an image of $N = n \times n$ pixels.

```
n = 256;
name = 'hibiscus';
f = load_image(name,n);
f = rescale( sum(f,3) );
```

The algorithm starts at the coarsest scale $j = \log_2(n) - 1$

```
j = log2(n)-1;
```

The first step of the algorithm performs filtering/downsampling in the horizontal direction.

$$\tilde{a}_{j-1} = (a_j *^H h) \downarrow^{2,H} \quad \text{and} \quad \tilde{d}_{j-1} = (a_j *^H g) \downarrow^{2,H}$$

Here, the operator $*^H$ and $\downarrow^{2,H}$ are defined by applying $*$ and $\downarrow^2$ to each column of the matrix. The second step computes the filtering/downsampling in the vertical direction.

$$a_{j-1} = (\tilde{a}_j *^V h) \downarrow^{2,V} \quad \text{and} \quad d^V_{j-1} = (\tilde{a}_j *^V g) \downarrow^{2,V},$$

$$d^H_{j-1} = (\tilde{d}_j *^V h) \downarrow^{2,V} \quad \text{and} \quad d^D_{j-1} = (\tilde{d}_j *^V g) \downarrow^{2,V} .$$

A wavelet transform is computed by iterating high pass and loss pass filterings with h and g, followed by sub-samplings. Since we are in 2-D, we need to compute these filterings+subsamplings along the horizontal and then along the vertical direction (or in the reverse order, it does not matter).

Initialize the transformed coefficients as the image itself. `fW` will be iteratively transformed and will contain the coefficients.

```
fW = f;
```

Select the sub-part of the image to transform.

```
A = fW(1:2^(j+1),1:2^(j+1));
```

Apply high and low filtering+subsampling in the vertical direction (1st cooordinate), to get coarse and details.

```
Coarse = subsampling(cconvol(A,h,1),1);
Detail = subsampling(cconvol(A,g,1),1);
```

Note : subsampling(A,1) is equivalent to A(1 :2 :end, :) and subsampling(A,2) is equivalent to A(:,1 :2 :end). Concatenate them in the vertical direction to get the result.

```
A = cat3(1, Coarse, Detail );
```

Display the result of the vertical transform.

```
figure;
imageplot(f,'Original image',1,2,1);
imageplot(A,'Vertical transform',1,2,2);
```

Apply high and low filtering+subsampling in the horizontal direction (2nd coordinate), to get coarse and details.

```
Coarse = subsampling(cconvol(A,h,2),2);
Detail = subsampling(cconvol(A,g,2),2);
```

Concatenate them in the horizontal direction to get the result.

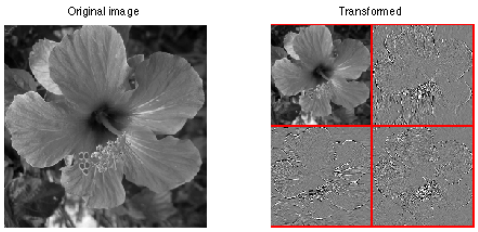
```
A = cat3(2, Coarse, Detail );
```

Assign the transformed data.

```
fW(1:2^(j+1),1:2^(j+1)) = A;
```

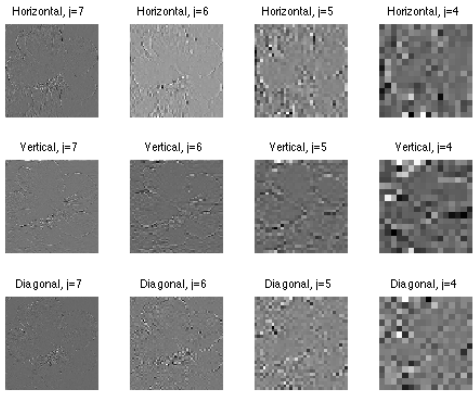
Display the result of the horizontal transform.

```
figure;
imageplot(f,'Original image',1,2,1);
subplot(1,2,2);
plot_wavelet(fW,log2(n)-1);  title('Transformed')
```



Exercice 1 : Implement a full wavelet transform that extract iteratively wavelet coefficients, by repeating these steps. Take care of choosing the correct number of steps.

```
figure; exo1;
```

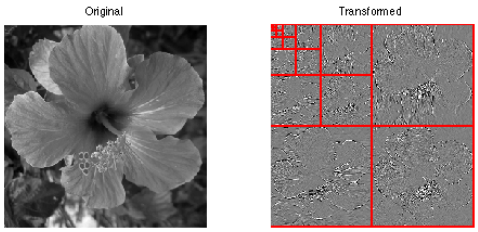


Check for orthogonality of the transform (conservation of energy).

```
disp(strcat(['Energy of the signal       = ' num2str(norm(f(:)).^2)]));
disp(strcat(['Energy of the coefficients = ' num2str(norm(fW(:)).^2)]));
```

Energy of the signal = 10863.2801
Energy of the coefficients = 10863.2801
Display the wavelet coefficients.

```
figure;
subplot(1,2,1);
imageplot(f);
title('Original');
subplot(1,2,2);
plot_wavelet(fW, 1);
title('Transformed');
```



50. **Inverse 2-D Wavelet transform**

Inversing the wavelet transform means retrieving a signal f1 from the coefficients fW. If fW are exactly the coefficients of f, then f=f1 up to machine precision.

Initialize the image to recover f1 as the transformed coefficient, and select the smallest possible scale.

```
f1 = fW;
j = 0;
```

Select the sub-coefficient to transform.

```
A = f1(1:2^(j+1),1:2^(j+1));
```

Retrieve coarse and detail coefficients in the vertical direction (you can begin by the other direction, this has no importance).

```
Coarse = A(1:2^j,:);
Detail = A(2^j+1:2^(j+1),:);
```

Undo the transform by up-sampling and then dual filtering.

```
Coarse = cconvol(upsampling(Coarse,1),reverse(h),1);
Detail = cconvol(upsampling(Detail,1),reverse(g),1);
```

Recover the coefficient by summing.

```
A = Coarse + Detail;
```

Retrieve coarse and detail coefficients in the vertical direction (you can begin by the other direction, this has no importance).

```
Coarse = A(:,1:2^j);
Detail = A(:,2^j+1:2^(j+1));
```

Undo the transform by up-sampling and then dual filtering.

```
Coarse = cconvol(upsampling(Coarse,2),reverse(h),2);
Detail = cconvol(upsampling(Detail,2),reverse(g),2);
```

Recover the coefficient by summing.

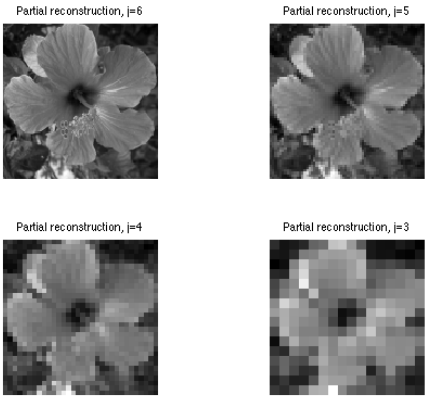
```
A = Coarse + Detail;
```

Assign the result.

```
f1(1:2^(j+1),1:2^(j+1)) = A;
```

Exercice 2 : Write the inverse wavelet transform that computes f1 from the coefficients fW. Compare f1 with f.

```
figure; exo2;
```



Check that we recover exactly the original image.

```
disp(strcat(['Error |f-f1|/|f| = ' num2str(norm(f(:)-f1(:))/norm(f(:)))]));
```

Error |f-f1|/|f| = 8.6602e-12

51. **Linear 2-D Wavelet Approximation**

Linear approximation is performed by setting to zero the fine scale wavelets coefficients and then performing the inverse wavelet transform. Here we keep only `1/eta^2` of the wavelet coefficients, thus calculating an approximation with only `m=n^2/eta^2` wavelet coefficients.

```
eta = 4;
fWLin = zeros(n,n);
fWLin(1:n/eta,1:n/eta) = fW(1:n/eta,1:n/eta);
```

Exercice 3 : Compute and display the linear approximation fLin obtained from the coefficients fWLin by inverse wavelet transform.

```
figure; exo3;
```

Test different values for `eta` : $\{1, 4, 8, 16, 32\}$. How many wavelet coefficients are kept ? Observe the result on the reconstructed image.

52. **Non-Linear 2-D Wavelet Approximation**

A non-linear M-term approximation is obtained by keeping only the M largest coefficients, which creates the smallest possible error.

Removing the smallest coefficient, to keep the M-largest, is equivalently obtained by thresholding the coefficients to set to 0 the smallest coefficients.

First select a threshold value (the largest the threshold, the more agressive the approximation).

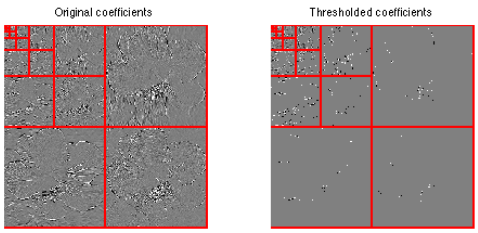
```
T = .2;
```

Then set to 0 coefficients with magnitude below the threshold.

```
fWT = fW .* (abs(fW)>T);
```

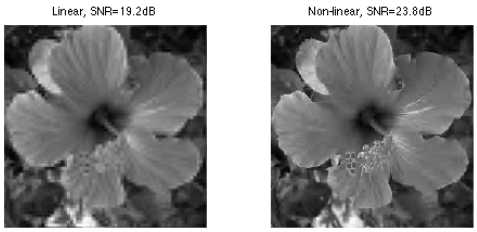
Display the coefficients selected over the threshold.

```
figure;
subplot(1,2,1);
plot_wavelet(fW);
axis('tight');
title('Original coefficients');
subplot(1,2,2);
plot_wavelet(fWT);
axis('tight');
title('Thresholded coefficients');
```



Exercice 4 : Find the threshold values T so that the number M of remaining coefficients in fWT are `M=n^2/eta^2`. Use this threshold to computefWT and then display the corresponding approximation $Mnlin$ of f. Compare this result with the linear approximation.

```
figure; exo4;
```



Test different values for `eta` : $\{1, 4, 8, 16, 32\}$. How many wavelet coefficients are kept ? Observe the result on the reconstructed image.

Exercice 5 : Try with different kinds of wavelets, with an increasing number of vanishing moments.

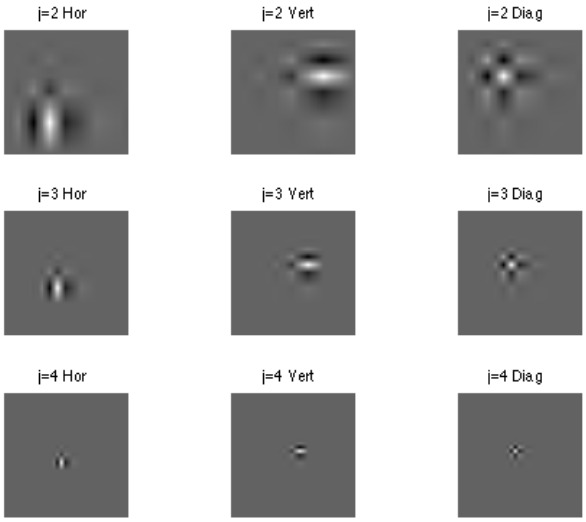
```
figure; exo5;
```

53. **The Shape of a 2D Wavelet**

A wavelet coefficient $fW[k]$ corresponds to an inner product $\langle f, \psi^s_{j,p} \rangle$ where k depends on the scale j and the position p and the orientation s (horizontal, vertical or diagonal). The wavelet image $f1 = \psi_{j,p}$ is computed by applying the inverse wavelet transform to fW where $fW[k] = 1$ and $fW[l] = 0$ for $l \neq k$.

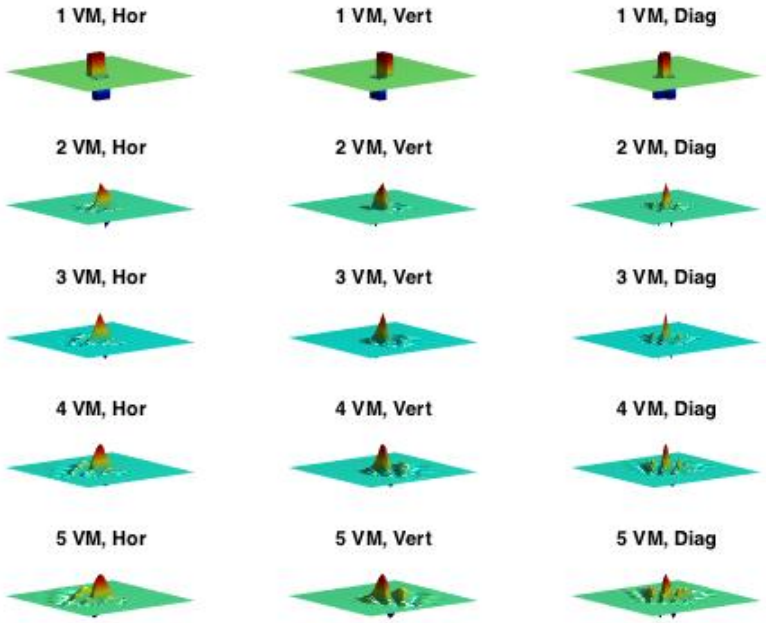
Exercice 9 : Compute wavelets at several scales and orientation. Here we show only horizontal wavelets, in 2-D.

```
figure; exo9;
```



Exercice 10 : Display Daubechies wavelets with different orientation, for different number of vanishing moments.

```
figure; exo10;
```



6.3 Débruitage par la décomposition en ondelettes

Pour compléter la partie debruitage de la section 4, on réutilise ici la représentation en Ondelettes en appliquant un seuil sur les coefficients. Cette méthode consiste à calculer une transformée orthogonale (transformée en ondelettes : T.O.) de l'image bruitee. Pour une image non bruitée les coefficients de la T.O. se concentrent sur une petite partie du domaine transformé, sous forme de coefficients de fortes valeurs. La plupart des autres coefficients sont proches de 0 (propriété de parcimonie).

54. Lancer le programme `wavelets`, comprendre son fonctionnement et noter et interpréter comme précedemment les différents résultats (σ^2, PSNR, SNR). Les images débruitées se trouvent dans `./images/wavelets`. L'ajustement du seuil peut être optimisé selon le contenu de l'image, le type et la proportion du bruit, et la transformée appliquée.

    Afficher l'histogramme des valeurs des coefficients d' ondelettes pour deux images différentes et chaque transformée. Comment choisir le seuil imposé aux coefficients ?

7 Section Complémentaire Wavelets

7.1 Image Approximation with Orthogonal Bases

Include of code solutions into your PATH variable :

```
addpath('solutions/coding_1_approximation')
```

55. **Best M-terms Non-linear Approximation**

This tours makes use of an orthogonal base $\mathcal{B} = \{\psi_m\}_{m=0}^{N-1}$ of the space $\mathbb{R}^N$ of the images with N pixels. The best M-term approximation of f is obtained by a non-linear thresholding

$$f_M = \sum_{|\langle f,\psi_m\rangle|>T} \langle f, \psi_m\rangle \psi_m,$$

where the value of $T > 0$ should be carefully selected so that only M coefficients are not threshold, i.e.

$$\mathrm{card}(\{m, |\langle f, \psi_m\rangle| > T\}) = M.$$

The goal is to use an ortho-basis $\mathcal{B}$ so that the error $\|f - f_M\|$ decays as fast as possible when M increases, for a large class of images.

This tour studies several different orthogonal bases : Fourier, wavelets (which is at the heart of the JPEG-2000 codec), cosine, local cosine (which is at the heart of JPEG block codec).

First we load an image of $N = n \times n$ pixels.

```
n = 512;
f = rescale( load_image('lena', n) );
```

Display it.

```
figure;
imageplot(f);
```

56. **Fourier Approximation**

The discrete 2-D Fourier atoms are defined as :

$$\psi_m(x) = \frac{1}{\sqrt{N}} e^{\frac{2i\pi}{n}(x_1 m_1 + x_2 m_2)},$$

where $0 \leq m_1, m_2 < n$ indexes the frequency.

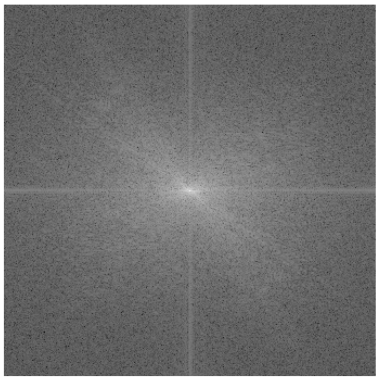
The set of inner products $\{\langle f, \psi_m \rangle\}_m$ is computed in $O(N \log(N))$ operations with the 2-D Fast Fourier Transform (FFT) algorithm (the Matlab function is fft2).

Compute the Fourier transform using the FFT algorithm. Note the normalization by $1/n$ to ensure orthogonality (energy conservation) of the transform.

```
fF = fft2(f)/n;
```

Display its magnitude (in log scale). We use the function fftshift to put the low frequency in the center.

```
figure;
imageplot(log(1e-5+abs(fftshift(fF))));
```



An image is recovered from a set of coefficients c_m using the inverse Fourier Transform (Matlab function ifft2) than implements the formula

$$f_M = \sum_m c_m \psi_m.$$

Performs a thresholding.

```
T = .3;
c = fF .* (abs(fF)>T);
```

Inverse the Fourier transform.

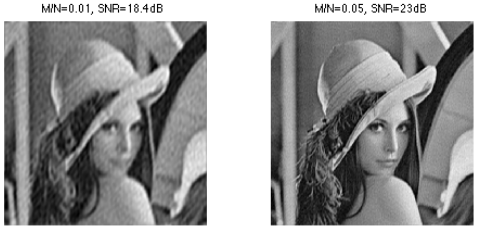
```
fM = real(ifft2(c)*n);
```

Display the approximation.

```
figure;
imageplot(clamp(fM));
```

Exercice 1 : Compute a best M-term approximation in the Fourier basis of f, for $M \in \{N/100, N/20\}$,$(N = n^2)$. Compute the approximation using a well chosen hard threshold value T.

```
figure; exo1;
```



The best M-term approximation error is computed using the conservation of energy as

$$\epsilon[M]^2 = \|f - f_M\|^2 = \sum_{|\langle f,\psi_m\rangle| \leq T} |\langle f, \psi_m \rangle|^2 .$$

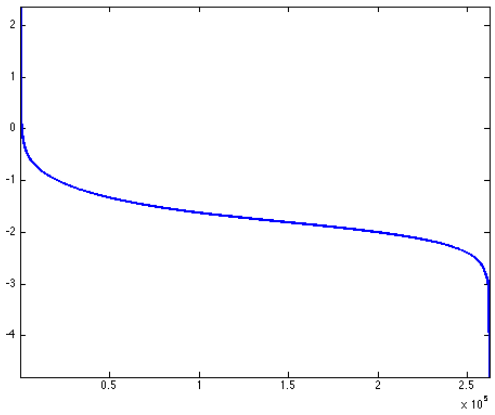
If one denotes by $\{c_R[k]\}_{k=0}^{N-1}$ the set of coefficients magnitudes $|\langle f, \psi_m \rangle|$ ordered by decaying magnitudes, then this error is easily computed as

$$\epsilon[M]^2 = \sum_{k=M}^{N-1} c_R[k]^2 = \|f\|^2 - \sum_{k=0}^{M-1} c_R[k]^2.$$

This means that ϵ^2 is equal to the energy of f i.e. $\|f\|^2$ minus the energy of the M-term approximation.

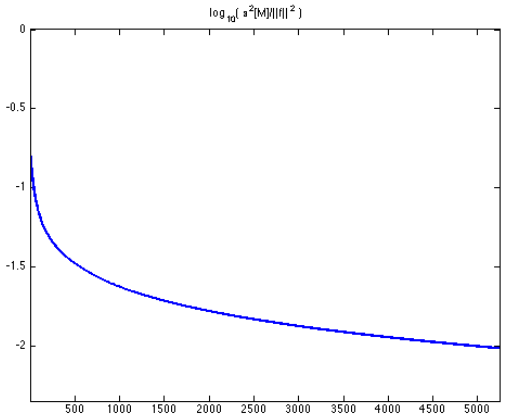
Exercice 2 : Compute and display in log scales the ordered coefficients c_R.

```
figure; exo2;
```



Exercice 3 : Compute and display in log-scale the non-linear approximation error $\epsilon[M]^2$ as a function of M. Store the values of $\epsilon[M]^2$ in a vector `err_fft`.

```
figure; exo3;
```



57. **Wavelet Approximation**

The Wavelet basis of continuous 2-D functions is defined by scaling and translating three mother atoms $\{\psi^H, \psi^V, \psi^D\}$:

$$\psi^k_{j,n}(x) = \frac{1}{2^j}\psi^k\left(\frac{x - 2^j n}{2^j}\right)$$

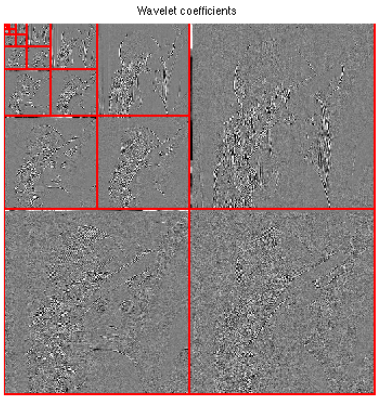
Non-linear wavelet approximation is at the heart of the JPEG-2000 compression standard. The set of inner products $\{\langle f, \psi_m\rangle\}_m$ is computed in $O(N)$ operations with the 2-D Fast Wavelet Transform algorithm.

Perform a wavelet transform. Here we use a Daubechies wavelet transform.

```
Jmin = 1;
options.h = compute_wavelet_filter('Daubechies',10);
fW = perform_wavortho_transf(f,Jmin,+1, options);
```

Display the coefficients.

```
figure;
plot_wavelet(fW,Jmin);
title('Wavelet coefficients');
```

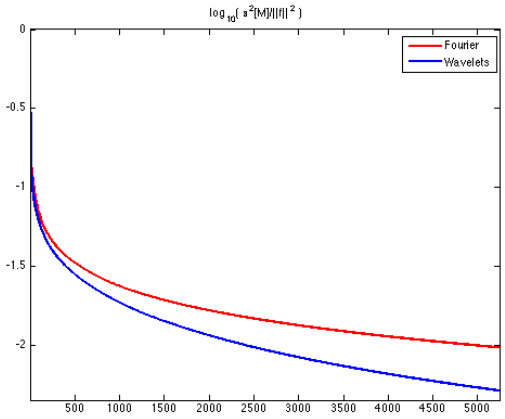


Exercice 4 : Compute a best M-term approximation in the wavelet basis of f, for $M \in \{N/100, N/20\}$. Compute the approximation using a well chosen threshold value T. Note that the inverse wavelet transform is obtained by replacing the $+1$ by a -1 in the definition of the transform.

```
figure; exo4;
```

Exercice 5 : Compute and display in log-scale the non-linear approximation error $\epsilon[M]^2$. Compares the Fourier and wavelets approximations. Store the values of $\epsilon[M]^2$ in a vector `err_wav`.

```
figure; exo5;
```



58. **Cosine Approximation**

The discrete cosine approximation (DCT) is similar to the Fourier approximation, excepted that it used symmetric boundary condition instead of periodic boundary condition, and is thus more useful to approximate image. A 1-D cosine atom of n sample is defined as

$$\bar{\psi}_m(x) = \frac{1}{\sqrt{N}} \cos\left(\frac{2\pi}{N}(x-1/2)m\right).$$

A 2-D cosine atom is obtained by tensor product of 1-D atoms

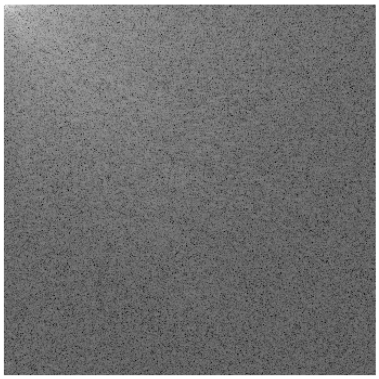
$$\bar{\psi}_{m_1,m_2}(x_1,x_2) = \bar{\psi}_{m_1}(x_1)\bar{\psi}_{m_2}(x_2).$$

The set of inner products $\{\langle f, \psi_m\rangle\}_m$ is computed in $O(N\log(N))$ operations with the 2-D Fast Cosine Transform algorithm (the Matlab function is dct2).

```
fC = dct2(f);
```

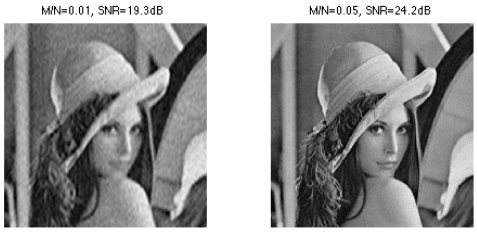
Display the magnitude of the DCT coefficients. Note that the low frequencies are in the upper-left corner.

```
figure;
imageplot(log(1e-5+abs(fC)));
```



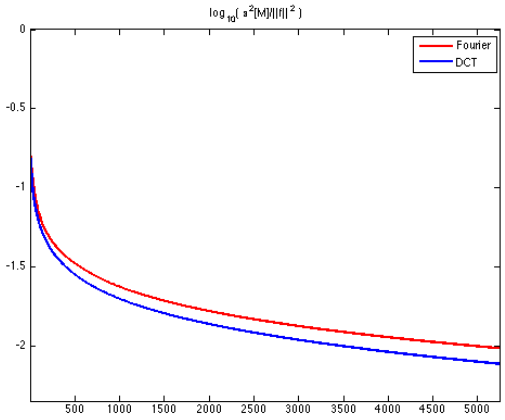
Exercice 6 : Compute a best M-term approximation in the DCT basis of f, for $M \in \{N/100, N/20\}$. Compute the approximation using a well chosen threshold value T. Note that the inverse DCT transform is obtained with the function `idct2`.

```
figure; exo6;
```



Exercice 7 : Compute and display in log-scale the non-linear approximation error $\epsilon[M]^2$. Compares the Fourier and DCT approximations. Store the values of $\epsilon[M]^2$ in a vector `err_dct`.

```
figure; exo7;
```



59. **Local Cosine Approximation**

To improve the global DCT approximation, one can approximate independently small patches in the image. This corresponds to a decomposition in a local cosine basis, which is at the heart of the JPEG image compression standard.

The only parameter of the transform is the size of the square.

```
w = 16;
```

Initialize at zero the transformed image in the local DCT basis.

```
fL = zeros(n,n);
```

Example of patch index.

```
i = 5;
j = 7;
```

For a given path index (i,j), we extract a (w,w) patch.

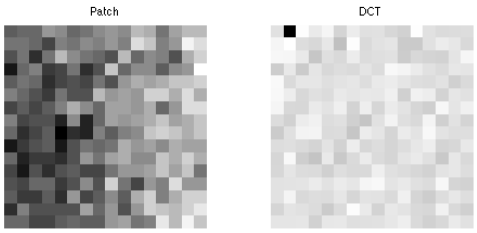
```
seli = (i-1)*w+1:i*w;
selj = (j-1)*w+1:j*w;
P = f(seli,selj);
```

Compute the Cosine transform of the patch using the fast DCT algorithm.

```
fL(seli,selj) = dct2(P);
```

Display the patch and its coefficients. Note that we removed the low frequency of P for display purposes. What happens if this is not done ?

```
figure;
imageplot(P,'Patch',1,2,1);
imageplot(dct2(P-mean(P(:))),'DCT',1,2,2);
```

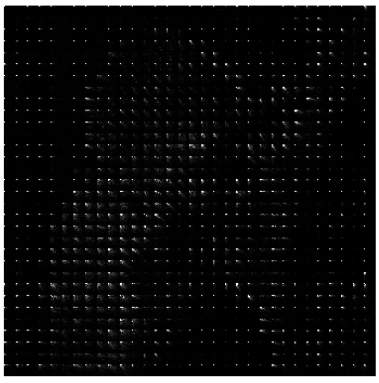


Exercice 8 : Compute the local DCT transform fL by transforming each patch.

```
exo8;
```

Display the coefficients.

```
figure;
imageplot(min(abs(fL),.005*w*w));
```



Exercice 9 : Compute the inverse local DCT transform of the coefficients `fL` by inverse transforming each patch using the function `idct2`.

```
exo9;
```

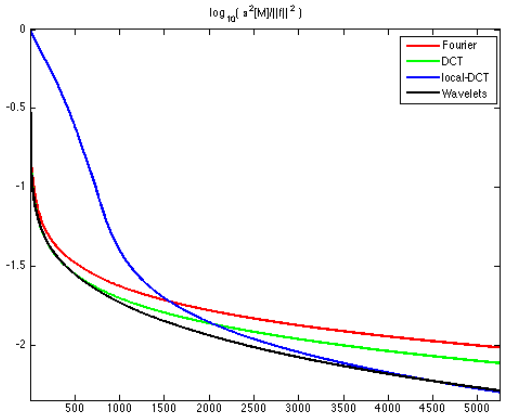
Error |f-f1|/|f| = 3.8496e-16

Exercice 10 : Compute a few best M-term approximations in the Local DCT basis of `f`.

```
figure; exo10;
```

Exercice 11 : Compute and display in log-scale the non-linear approximation error $\epsilon[M]^2$. Store the values of $\epsilon[M]^2$ in a vector `err_ldct`. Compare the Fourier, Wavelets, DCT and local-DCT approximations. Conclusion ?

```
figure; exo11;
```



60. **Comparison of Wavelet Approximations of Several Images**

An image is more complicated than an other one for a given orthogonal basis if its approximation error decays more slowly.

First load several images with different spatial statistics.

```
n = 512;
fList(:,:,1) = rescale( load_image('regular3',n) );
fList(:,:,2) = rescale( load_image('phantom',n) );
fList(:,:,3) = rescale( load_image('lena',n) );
fList(:,:,4) = rescale( load_image('mandrill',n) );
```

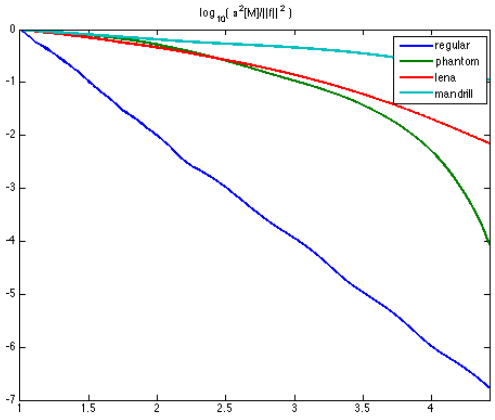
Display them.

```
figure;
for i=1:4
imageplot(fList(:,:,i),'', 2,2,i);
end
```




Exercice 12 : Compare the approximation error decay for those images. Display $\log_{10}(\|f - f_M\|)$ as a function of $\log_{10}(M)$.

```
figure; exo12;
```



7.2 Image Compression with Wavelets

Inclusion of solutions into your PATH variable :

```
addpath('solutions/coding_4_wavelet_compression')
```

61. **Wavelet Domain Quantization**

Image compression is performed by first quantizing the wavelet coefficients of an image.

A scalar quantizer of step size |T| uses the function |floor|. It has a zero bin which is twice larger than the other bins, as will be seen.

Create values evenly spaced for quantization.

```
v = linspace(-1,1, 2048);
```

Bin size for the quantization. The larger, the more agressive the compression.

```
T = .1;
```

For compression, we compute quantized integer values.

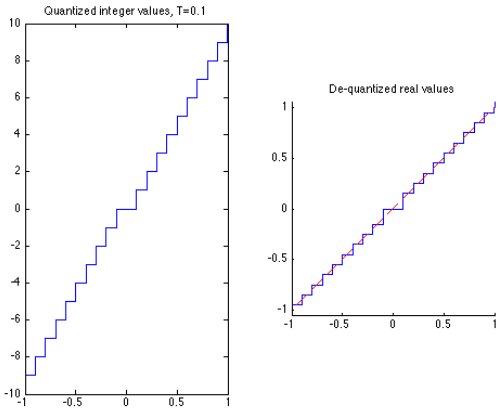
```
vI = floor(abs(v/T)).*sign(v);
```

For decompression, we compute de-quantized values from `vI`, which are chosen as the mid-point of each quantization bin.

```
vQ = sign(vI) .* (abs(vI)+.5) * T;
```

Display the quantization curve.

```
figure;
subplot(1,2,1);
plot(v, vI);
axis('tight');
title(strcat(['Quantized integer values, T=' num2str(T)]));
subplot(1,2,2);
hold('on');
plot(v, vQ);
plot(v, v, 'r--');
axis('equal'); axis('tight');
title('De-quantized real values');
```



62. **Quantization and Approximation of Wavelet Coefficients**

The compression is obtained by setting to 0 the wavelet coefficients which are smaller than |T|, and then by quantizing the remaining non-zero coefficients. It thus creates an error that is slightly larger than simply performing an approximation with thresholding at |T|.

First we load an image.

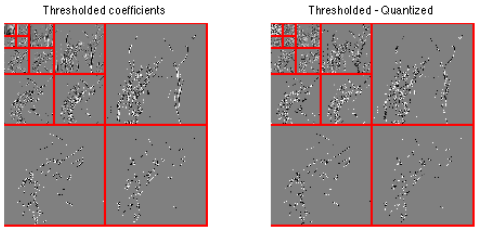
```
n = 256;
M = rescale( load_image('lena', n) );
```

Compute its wavelet transform.

```
Jmin = 4;
MW = perform_wavelet_transf(M,Jmin, +1);
```

Exercise 1 : Compute the coefficients `MWT` obtained by thresholding at `T` the coefficients `MW`. Compute the coefficients `MWQ` obtained by quantizing with bin size `T` the same coefficients. Display them using the function `plot_wavelet`.

```
figure; exo1()
```



Exercise 2 : Compare the effect of quantizing at `T=.2` and thresholding at `T=.2` the wavelet coefficients of an image. Display the difference between the two methods.

```
figure; exo2()
```

Exercise 3 : Compute a bin size `T0` to quantize the original image `M` itself, to obtain `MQ0` such that `norm(M-MQ,'fro')` is as close as possible to the error obtained with wavelet domain quantization.

```
figure; exo3()
```

Spatial quantization step T0=0.089.

63. **Entropy Coding the Wavelet Coefficients**

To store the quantized coefficients in a file, one need to compute a binary code from `MWI`. The length of this code is the number of bits used by the compressor, which typically increases when `T` decays toward 0.

To reduce the number of bits, an entropic coder makes use of the statistical distribution of the quantized values.

First we quantize the coefficients.

```
MWI = floor(abs(MW/T)).*sign(MW);
MWQ = sign(MWI) .* (abs(MWI)+.5) * T;
```

Assuming that all the coefficients of `MWI` are drawn independently from the same distribution with histogram `h`, the minimum bit per pixel achievable is the entropy lower bound.

$$E = -\sum_i h(i).\log_2(h(i))$$

Huffman trees and more precisely block-Huffman tree codes get increasingly closer to this bound when the data size increases. Arithmetic coders also achieves very good results and are fast to compute.

Compute the normalized histogram of the quantized wavelet coefficients.

```
a = max(abs(MWI(:)));
t = -a:a;
h = hist(MWI(:), t); h = h/sum(h);
```

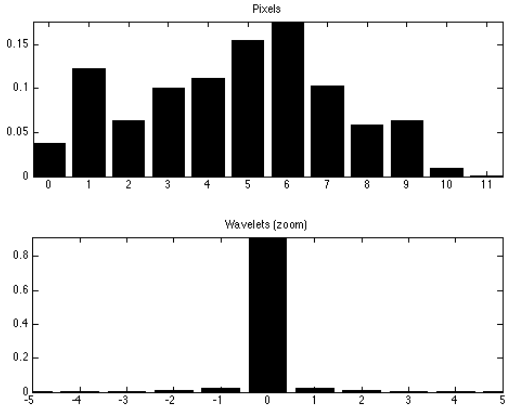
Compute the histogram of the quantized pixels of the original image.

```
t0 = 0:1/T0;
MI = floor(abs(M/T0)); % quantized pixel values
h0 = hist(MI(:), t0); h0 = h0/sum(h0);
```

Display the histograms.

```
figure;
subplot(2,1,1);
```

```
bar(t0,h0); axis('tight');
title('Pixels');
subplot(2,1,2);
bar(t,h); axis([-5 5 0 max(h)])
title('Wavelets (zoom)');
```



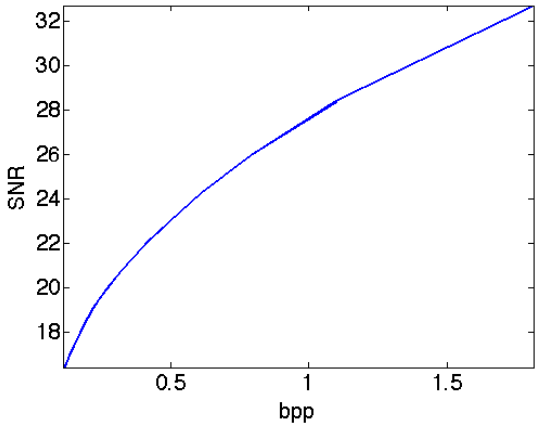
Exercise 4 : Compute the entropy lower bound for the quantized wavelet coefficients and for the quantized pixel values. Take care of |log2(0)| when |h(i)=0|.

```
exo4()
```

Pixels entropy : 3.2 Wavelet entropy : 0.72

Exercise 5 : Compute, for various threshold `T`, the number of bits per pixels `E(T)` of the quantized wavelet coefficients, the wavelet decompression error `err(T)` and the corresponding SNR. Display the SNR as a function of the number of bits per pixels `E`.

```
figure; exo5()
```



64. **Scale-by-scale Entropy Coding**

Wavelet coefficients of an image do not have the same distribution across the scales. Taking this into account can further reduce the number of bits for coding.

Quantize the coefficients.

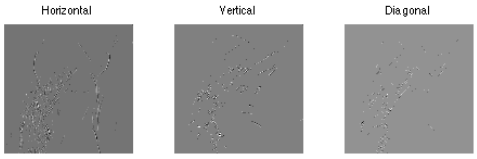
```
T = .1;
MWI = floor(abs(MW/T)).*sign(MW);
```

Extract the fine scale wavelet coefficients.

```
MWH = MWI(1:n/2,n/2+1:n);
MWV = MWI(n/2+1:n,1:n/2);
MWD = MWI(n/2+1:n,n/2+1:n);
```

Display.

```
figure;
imageplot(MWH,'Horizontal',1,3,1);
imageplot(MWV,'Vertical',1,3,2);
imageplot(MWD,'Diagonal',1,3,3);
```



Exercise 6 : Extract the three fine scale wavelet coefficients (horizontal, vertical, diagonal directions) and quantize them, for instance with |T=.1|. Compute the entropy of the three sets together, and compute the entropy of each set.

```
exo6()
```

Entropy, all : 0.287 Entropy, hor : 0.466 Entropy, vert : 0.212 Entropy, diag : 0.157

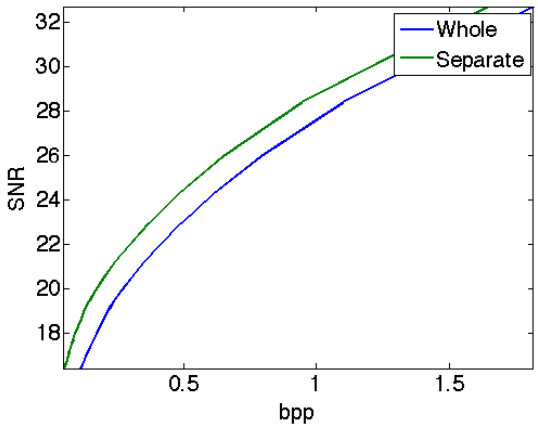
Exercise 7 : Compare the number of bits needed to code all the wavelet coefficients together, and the number of bits needed to code independently each scale of wavelet coefficients for `Jmin=4<=j<=log2(n)-1` (and group together the remaining coefficients for `j<Jmin`).

```
exo7()
```

nb.bis, whole : 0.719 bpp nb.bis, separate : 0.58 bpp

Exercise 8 : Compute the rate distortion curve obtained by coding the coefficient separately through the scales, and compare with the rate distortion curve obtained by coding the coefficients as a whole. Conclusion ?

```
exo8()
```


Whole
Separate
SNR
bpp
32
30
28
26
24
22
20
18
0.5
1
1.5