DURET Guillaume GR IMA
SCHIEB Antoine

TP Ondelettes 2D:

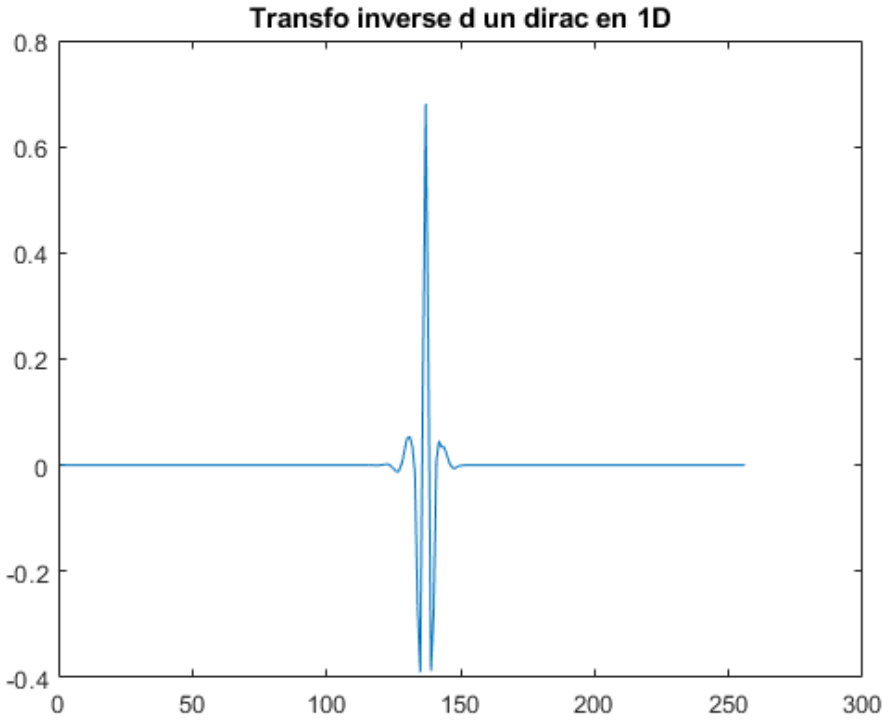
2018-2019

1) Debruitage d’une image

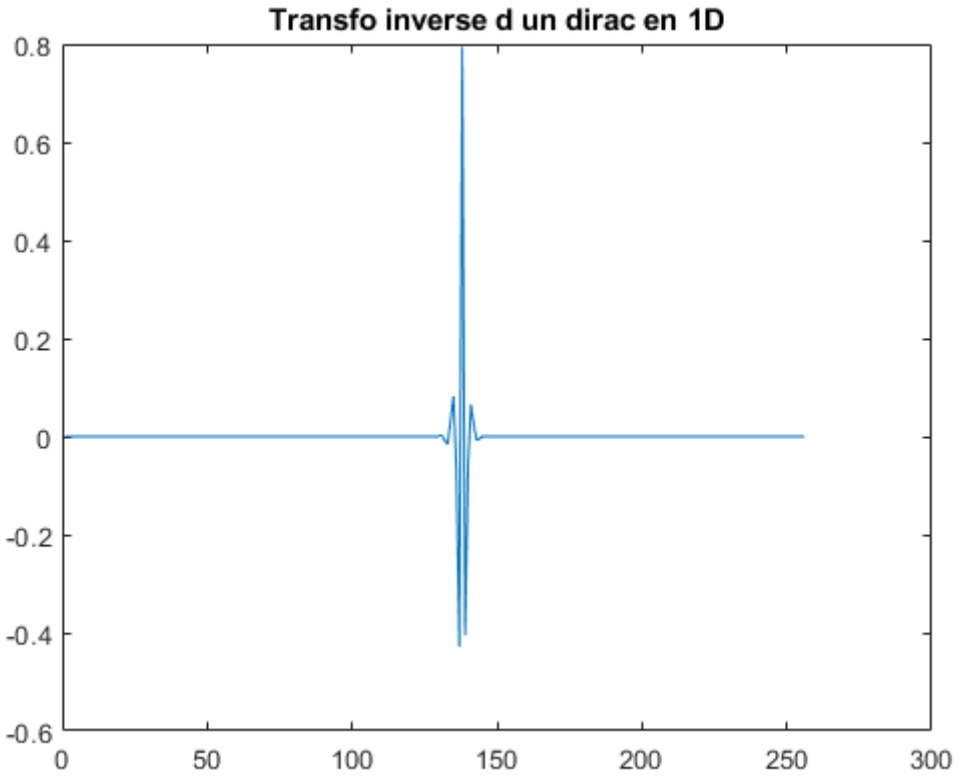
A/ Analyse multi-résolution : décomposition/reconstruction

Les ondelettes sont des bases de fonctions utilisées dans des domaines tels que le traitement du signal, de l’image, mais aussi en résolution d’équations aux dérivées partielles par l’outil numérique. Ce TP a pour but de visualiser les possibilités qu’offrent les ondelettes via la reconstruction d’image bruitée, et la reconnaissance de textures.

```
% débruitage

CoifQMF = MakeONFilter('Coiflet',3); % creation du filtre ON

figure(1);
% parametres de la transformation en ondelettes
J = 8;
L = 5;
dirac=zeros(1,2^J); % creation du dirac
dirac(5)=1;
transfo_inverse = IWT_PO(dirac,L,CoifQMF);
% affichage transfo inverse
plot(transfo_inverse);
title('Transfo inverse d un dirac en 1D');
```



f(t)=Dirac (t-100)



f(t)=Dirac (t-200)

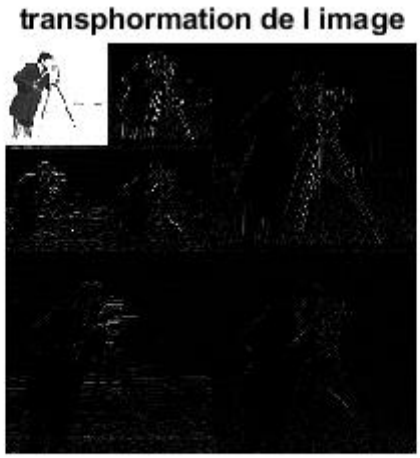
On peut assimiler ces transformations inverses d'impulsions à une réponse impulsionnelle de filtre 1D comme ceux vus en traitement du signal.
On remarque que plus on 'avance' la position du dirac d'origine plus la forme de l'ondelette augmente, car le filtre associé sera plus ou moins apte à sélectionner les hautes ou basses fréquences.

f(t)=Dirac(t-5)

On peut aussi noter que pour un dirac retardé d'une valeur inférieure à 2^L, on se trouve dans la partie approximation de l'image et la moyenne du signal n'est pas nulle contrairement aux transformées inverses d'un dirac placé après 2^L. Les signaux visualisés pour un dirac retardé d'une valeur inférieure à 2^L ne sont donc pas des ondelettes.

```
figure(3);

I_entier=imread('cameraman.tiff');
I=double(I_entier);
imshow(I_entier,[]);
title('image originale');

transfo_img = FWT2_PO(I,L_i,CoifQMF); % calcul de la transfo de l'image

transfo_inv_img = IWT2_PO(transfo_img,L_i,CoifQMF); % puis de la transformée
inverse de celle-ci.
```

```
figure(4);
imshow(uint8(transfo_img),[]);
title('transphormation de l image');



figure(5);
transfo_inv_img_int = uint8(transfo_inv_img);
imshow(transfo_inv_img_int,[]);
title('transphormation inverse de l image');
```



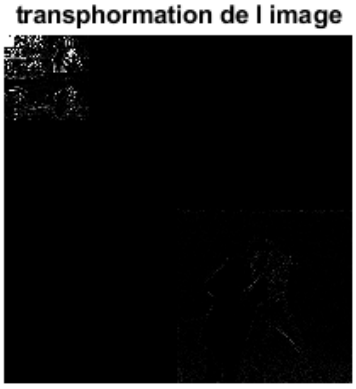
Transformation pour Li=6

On remarque que la majorité de l'information est contenue dans le coin supérieur gauche de la transformée. Cette zone correspond à la partie d'approximation qui occupe ¼ de l'image en largeur, et ¼ de l'image en longueur. (car 2^Li /2^Ji = 1/4)

Transformation pour Li=3

On peut voir que l'information est bien répartie au sein des différentes zones, que ce soit les zones de détail ou les zones d'approximation. La zone la plus en haut à gauche correspond à la partie d'approximation qui occupe 1/32 de l'image en largeur, et 1/32 de l'image en longueur. (car 2^Ji / 2^Li = 32) Quelque soit la valeur de Li l'image est toujours correctement reconstituée.

On a essayé de supprimer les données contenues dans les partie détails de l'image pour li=3 et on obtient les figures suivantes :



On remarque qu'en refaisant la transformation de l'image, celle-ci est flou car les détails qui représente les hautes fréquences de l'image ne sont plus présente.

B/ Application au débruitage d'une image

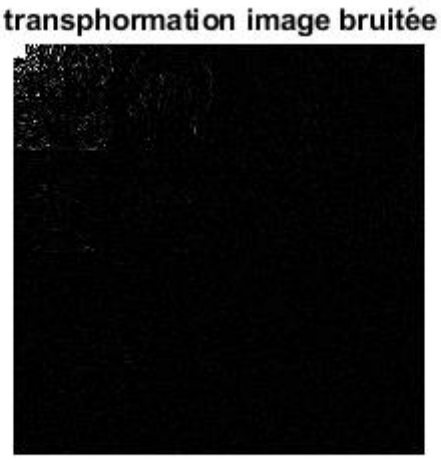
Nous voulons ensuite appliquer la décomposition en ondelette pour le débruitage d'une image.

```
% application au debruitage d'une image

%Lecture puis bruitage de l'image d'origine
Ingrid = ReadImage('Daubechies');
NoisyIngrid = Ingrid + 5*WhiteNoise(Ingrid);


% Transformation ondelettes
L_i = 3; %imposÃ© par l'Ã©noncÃ©
CoifQMF = MakeONFilter('Coiflet',3);
transfo_ingrid = FWT2_PO(NoisyIngrid,L_i,CoifQMF);


% seuillage de la transformation
seuil=10; %imposÃ© par l'Ã©noncÃ©
transfo_ingrid_seuil = zeros(256,256);
for i=1:256
   for j=1:256
      pixel=transfo_ingrid(i,j);
      if (pixel > 0)
        transfo_ingrid_seuil(i,j) = max(0 ,transfo_ingrid(i,j)-seuil);
      else
        transfo_ingrid_seuil(i,j) = min(0 ,transfo_ingrid(i,j)+seuil);
      end
   end
end

%reconstruction Ã  partir de l'image bruitÃ©e grÃ¢ce Ã  la transfo seuillÃ©e
reconstruct_seuil = IWT2_PO(transfo_ingrid_seuil,L_i,CoifQMF);
reconstruct = IWT2_PO(transfo_ingrid,L_i,CoifQMF);


% affichage
figure(1);
imshow(uint8(Ingrid),[]);
title('image originale')
figure(2);
imshow(uint8(NoisyIngrid),[]);
title('image bruitée')
figure(3);
imshow(uint8(transfo_ingrid),[]);
title('transphormation image bruitée')
figure(4);
```

```
imshow(uint8(transfo_ingrid_seuil),[]);
title('transphormation image bruitée seillée')
figure(5);
imshow(uint8(reconstruct),[]);
title('transphormation image bruitée sans seuil')
figure(6);
axis([110 160 110 160]); axis square;
imshow(uint8(reconstruct_seuil),[]);
title('transphormation image bruitée avec seuil')
```

En effet on commence par appliquer un bruit blanc sur l'image puis on réalise une transformation en ondelette sur l'image bruitée

De plus on réalise la transformé à l'échelle 3



La transformée en ondelette de l'image permet de mettre en valeur le bruit blanc car celui-ci a une intensité faible dans la transformée. Nous pouvons donc supprimer ce bruit en supprimant les faibles intensités.

transphormation image bruitée sans seuil

transphormation image bruitée avec seuil

On remarque que l'image finale reconstituée est correctement débruitée, mais elle a tout de même perdu en finesse et en précision elle paraît plus floue.

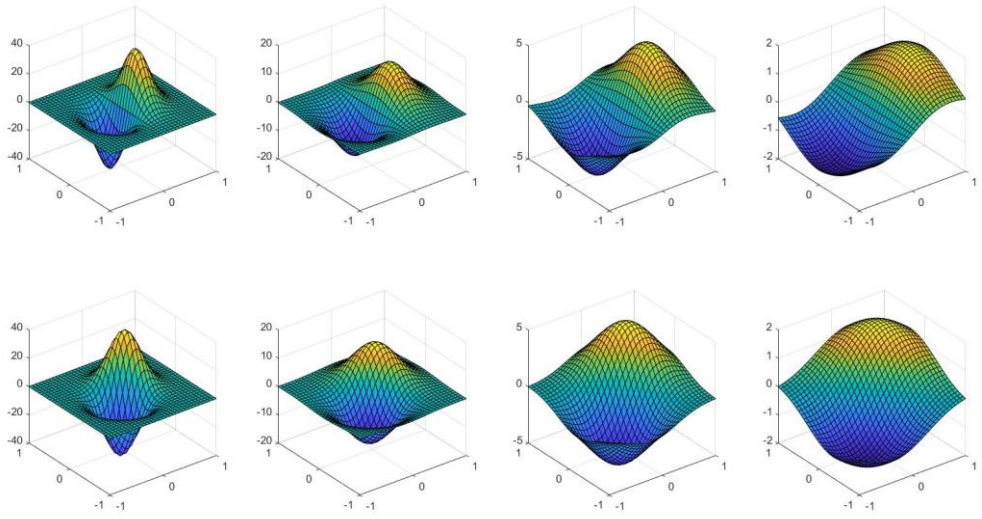
2) Classification: transformée en ondelettes

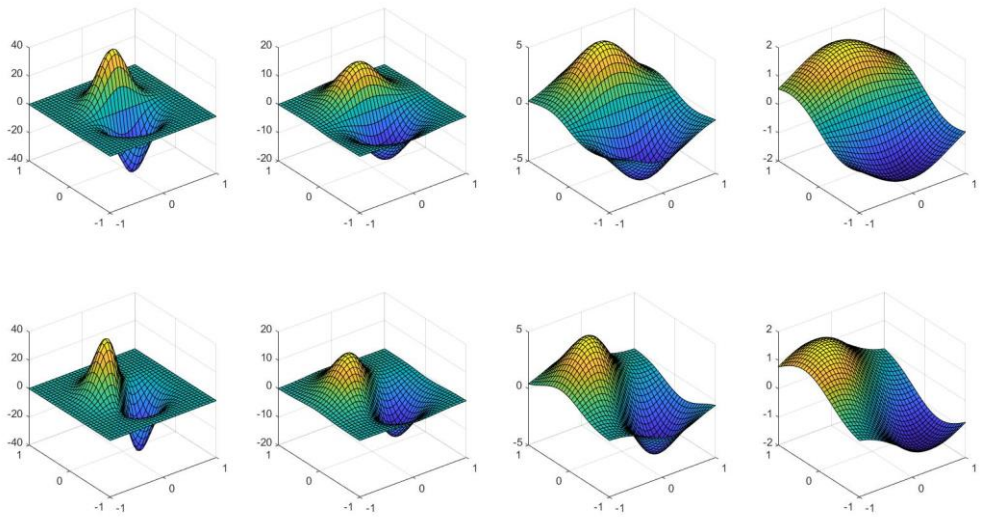
A/ Ondelettes orientées : derivées gaussienne

```
close all; clear variables;

%Classification: transformÃ©e en ondelettes continues
dx=2/64;
alpha=pi;
[X,Y]=meshgrid(-1:dx:+1);
T=exp((-1/2)*(X.^2+Y.^2));
Psi=(X*cos(alpha)+Y*sin(alpha)).*T;
figure(1);
surf(X,Y,Psi); % Visualisation d'une ondelette simple



dx=4/64;
l=1;
dtheta=3.1415/4;
```

```
R_l_dtheta = [cos(l*dtheta) sin(l*dtheta);-sin(l*dtheta) cos(l*dtheta)];
[X,Y]=meshgrid(-1:dx:+1);

% On affiche l'ondelette avec un jeu de 16 paramètres différants
for l=0:1
   for j = -4:-1
       disp((j+5)*(l+1));
       subplot(2,4,(j+5)+(l*4));
       scal=1/(2^j);
       T=exp((-1/2)*(scal*X.^2+scal*Y.^2));
       alpha=l*dtheta;
       Phi =(scal*X*cos(alpha)+scal*Y*sin(alpha)).*T;
       phi_j_l=(1/2^j)*Phi;
       surf(X,Y,phi_j_l);
   end
end

figure();
[X,Y]=meshgrid(-1:dx:+1);
phi_j_l = zeros(1,length(X));
for l=2:3
   for j = -4:-1
       disp((j+5)*(l+1));
       subplot(2,4,(j+5)+((l-2)*4));
       scal=1/(2^j);
       T=exp((-1/2)*(scal*X.^2+scal*Y.^2));
       alpha=l*dtheta;
       Phi =(scal*X*cos(alpha)+scal*Y*sin(alpha)).*T;
       phi_j_l=(1/2^j)*Phi;
       surf(X,Y,phi_j_l);
   end
end
```

Ce code nous permet d'afficher les figure suivantes :





On observe que sur chaque ligne on a un angle différent et sur chaque colonne on a une échelle différente, la forme de l'ondelette restant la même

B/ Application à la caractérisation de textures

Ensuite nous cherchons à appliquer la méthode précédente à des images de texture.

```
%exoTextures
close all; clear variables;
Texture1 = im2double(imread('Texture1.png'));
Texture2 = im2double(imread('Texture2.png'));
Texture3 = im2double(imread('Texture3.png'));
Texture4 = im2double(imread('Texture4.png'));
Texture5 = im2double(imread('Texture5.png'));
Texture6 = im2double(imread('Texture6.png'));
Texture7 = im2double(imread('Texture7.png'));

Image1 = im2double(imread('Image1.png'));
Image2 = im2double(imread('Image2.png'));

% dÃ©claration des paramÃ¨tres des ondelettes
alpha=pi;
```

```
dtheta=pi/4;
dx=2/64;
[X,Y]=meshgrid(-1:dx:+1);

% vect contient la valeur de la norme au carrÃ© du rÃ©sultat de conv2(), pour un
jeu de paramÃ¨tres donnÃ©.
vect=zeros(9,1);

% matrice_caract contient les 16 vecteurs vect concatÃ©nÃ©s (car on utilise
% 16 paramÃ¨tre diffÃ©rents)
matrice_caract=zeros(9,16);
loop_count=0;
for l=0:3
   for j = -4:-1
       loop_count=loop_count+1;
       scal=1/(2^j);
       alpha=l*dtheta;
       T=exp((-1/2)*(X.^2+Y.^2));
       psi = scal*(scal*X*cos(alpha)+scal*Y*sin(alpha)).*T;


       vect(1)= norm(conv2(Texture1,psi)).^2;
       vect(2)= norm(conv2(Texture2,psi)).^2;
       vect(3)= norm(conv2(Texture3,psi)).^2;
       vect(4)= norm(conv2(Texture4,psi)).^2;
       vect(5)= norm(conv2(Texture5,psi)).^2;
       vect(6)= norm(conv2(Texture6,psi)).^2;
       vect(7)= norm(conv2(Texture7,psi)).^2;
       vect(8)= norm(conv2(Image1,psi)).^2;
       vect(9)= norm(conv2(Image2,psi)).^2;
       matrice_caract(:,loop_count) = vect;

   end
end

% on fabrique deux matrices : une matrice des vecteurs correspondant Ã
% chaque texture mais centrÃ©e sur la texture de l'image 1, et l'autre
% centrÃ©e sur la texture de l'image 2.
matrice_caract_diff1 = zeros(7,16);
matrice_caract_diff2 = zeros(7,16);
for texture=1:7
    for coord=1:16
        matrice_caract_diff1(texture,coord) = matrice_caract(texture,coord) -
matrice_caract(8,coord);
        matrice_caract_diff2(texture,coord) = matrice_caract(texture,coord) -
matrice_caract(9,coord);
    end
end

% il suffit alors de calculer la distance Ã  l'origine de chaque vecteur
% (grÃ¢ce Ã  la fonction norm() et de stocker ces normes dans deux tableaux
% diffÃ©rents, un pour l'image 1, l'autre pour l'image 2.
dist_image1=zeros(7,1);
dist_image2=zeros(7,1);
for texture=1:7
```

```
    dist_image1(texture) = norm(matrice_caract_diff1(texture,:));
    dist_image2(texture) = norm(matrice_caract_diff2(texture,:));
end


% On cherche puis on affiche l'indice du vecteur le plus proche de
% l'origine pour chaque image.
min1 = find(dist_image1==min(dist_image1));
min2 = find(dist_image2==min(dist_image2));

disp(min1);
disp(min2);
```

Nous voulons à partir de 2 images retrouver parmi 7 images les textures les images correspondantes à la texture la plus proche

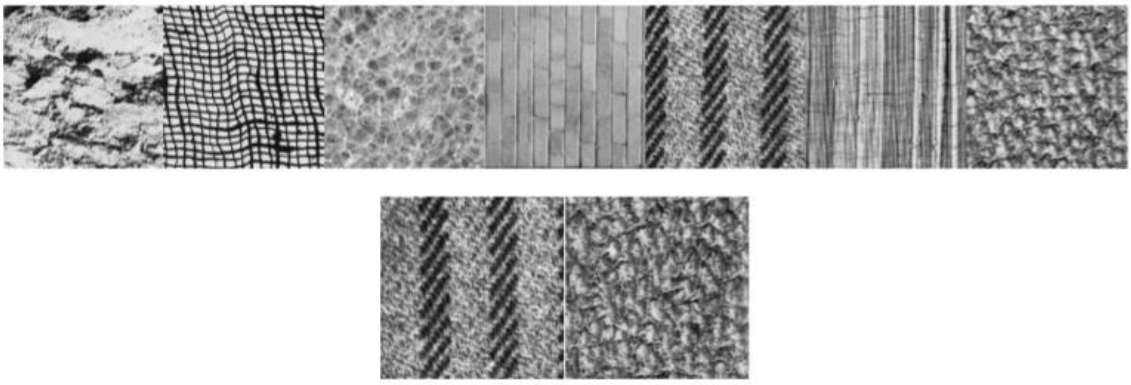


Figure 3 : images de 7 textures (haut) et deux images à "classer" (bas)

Une texture est une succession de motifs plus ou moins périodiques. Cela dit, pour notre algorithme de reconnaissance, deux textures doivent être considérées comme identiques même si elles ont une orientation différente, ou une échelle différente au sein de l'image

Effectuer la transformée en ondelette continue 2D d'une texture permet d'obtenir une matrice, dont la norme représente la « corrélation » entre les deux images, ie. plus cette valeur est grande, plus les deux images sont corrélées entre elles.

On construit donc un vecteur contenant ces valeurs pour chaque jeu de paramètre : ce vecteur sera donc placé dans un espace de dimension égale au nombre de combinaison de paramètres utilisées (ici 16 car nous faisons varier 4 fois l'orientation et 4 fois l'échelle des textures avant d'effectuer leur transformée en ondelette continue avec l'image testée).

On obtient donc un vecteur pour chaque texture, et un vecteur pour chaque image. La suite et fin de l'algorithme consiste donc à trouver quel est le vecteur le plus proche de celui de l'image testée au sens de la norme euclidienne, afin de trouver la texture qui est corrélée le mieux à l'image.

Finalement le programme fait bien correspondre les images à leurs textures correspondantes, même si elles sont à des échelles et rotations différentes.

Conclusion :

Nous avons pu voir que la transformation en ondelette est une transformée sans perte d'information qui peut avoir plusieurs applications comme le débruitage et la reconnaissance de texture. On peut noter que pour le débruitage avec cette méthode est très efficace mais a pour conséquence de flouter légèrement l'image. Dans le cas des textures l'inconvénient est de devoir tester toutes les combinaisons de paramètres différentes donc une tendance demander beaucoup de ressources, mais a pour avantage d'être une méthode plus précise que l'analyse des textures par Fourier car elle est plus adaptée aux textures locales.