Compte-rendu du TP 1

(16 octobre 2009)

L'objectif de ce premier TP est de faire des manipulations simples sur les images avec MatLab (création, visualisation) et commencer à appliquer la transformée de Fourier discrète (TDF) sur des images en niveaux de gris. On testera différentes techniques de zooms (avant et arrière), pour enfin coder des fonctions de translation et de rotation d'images.

I Premières manipulations

I.1 Quelques images simples

▷ Question 1 : On cherche à créer une image de taille 200×256 constante sur chaque colonne et de valeurs de niveaux de gris qui croît de 0 à 255. Le code suivant nous donne le résultat souhaité.

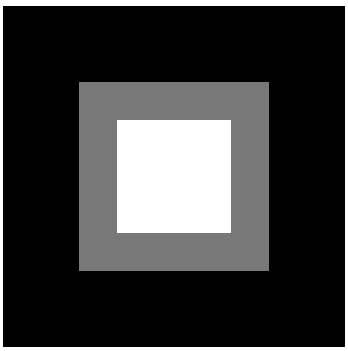
```
u = zeros(200,256);
for i = 1:200
    u(i,:) = (0:255);
end
imagesc(uint8(u),[0,255]);
colormap gray;
axis image;
```

▷ Question 2 : L'image suivante

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	120	120	120	120	120	0	0
0	0	120	255	255	255	120	0	0
0	0	120	255	255	255	120	0	0
0	0	120	255	255	255	120	0	0
0	0	120	120	120	120	120	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

correspond à un carré blanc de taille 9×9, avec un carré 3×3 noir au centre et une couronne grise d'un pixel.

```
v = zeros(9,9);
v(3:7,3:7) = 120*ones(5,5);
v(4:6,4:6) = 255*ones(3,3);
imagesc(uint8(v),[0,255]);
colormap gray;
axis image;
```



I.2 Visualisation des images

▷ QUESTION 3 : On compare la visualisation de l'image `water.png` avec la fonction `vis_img_mat` ; à gauche, on a donné la valeur `opt` 0 et à droite 1 :

`opt` $= 0$ affiche l'image par MATLAB sans changement, tandis que `opt` $= 1$ ramène la valeur la plus faible de l'image à 0 et la valeur la plus grande à 255 par un changement de variable affine. On observe donc une image plus contrastée (les pixels sombres le deviennent encore plus, alors que les pixels clairs s'éclaircissent davantage).

II Transformée de Fourier discrète

II.1 Dimension 1

▷ QUESTION 4 : La fonction `fft` (Fast Fourier Transform) de MATLAB utilise la formule suivante pour le calcul de la transformée de Fourier :

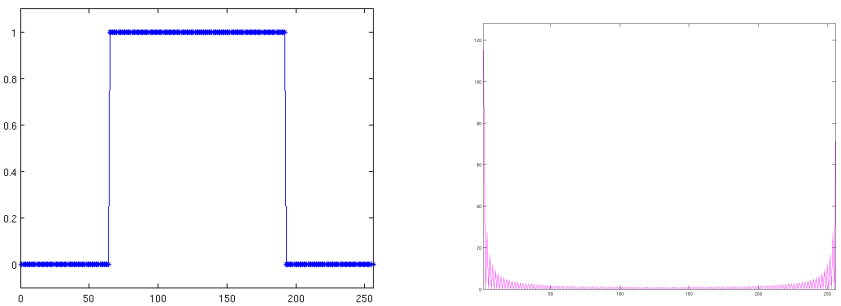
```
              N
X(k) =      sum   x(n)*exp(-j*2*pi*(k-1)*(n-1)/N), 1 <= k <= N.
            n=1
```

alors que notre formule est :

$$\widetilde{u}_k = \frac{1}{N}\sum_{n=0}^{N-1} u_n \omega_N^{-kn} = \frac{1}{N}\sum_{n=0}^{N-1} u_n \exp\left(\frac{-2i\pi kn}{N}\right) \quad , \quad -\frac{N}{2} \leq k \leq \frac{N}{2} + 1.$$

On remarque qu'il manque le coefficient $\frac{1}{N}$ et un décalage dans les coefficients.

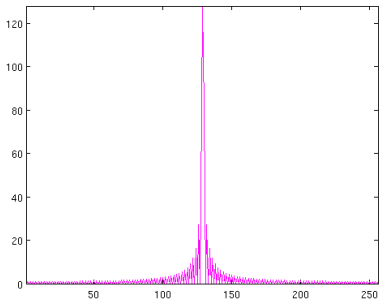
▷ QUESTION 5 : On applique la transformée de Fourier discrète sur un signal créneau ; la fonction `fft` donne le résultat suivant pour le module :



(a) Le signal créneau (b) Le spectre correspondant

Le signal créneau présente des discontinuités au centre du signal ; on s'attend donc à voir de nombreux coefficients non nuls. Cependant, la localisation des discontinuités suggèrent que les coefficients élevés doivent

se trouver au centre du spectre, et non aux bords. Si on applique la fonction `fftshift`, on obtient le spectre suivant :

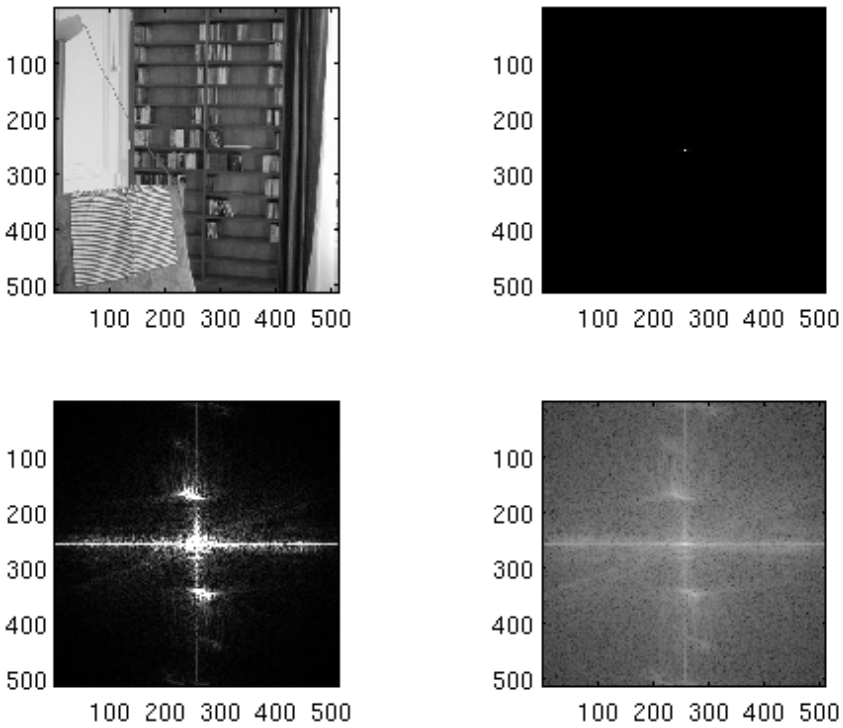


qui est plus conforme à nos attentes. Ceci est dû au décalage des indices dans la formule de `fft`. La fonction `fftshift` va donc translater la fenêtre de visionnage du spectre (si on imagine par exemple que le spectre est périodisé).

II.2 Dimension 2

On va maintenant appliquer la transformée de Fourier sur des images en niveaux de gris : à chaque point de l'image (x, y) on associe donc une valeur comprise entre 0 et 255. Pour cela, on utilisera les fonctions `fft2` et la TDF inverse sera donnée par `ifft2`.

▷ QUESTION 6 : On teste la fonction `vis_mod_img` qui permet de visualiser le module de la transformée de Fourier d'une image en tout point (en lui associant un niveau de gris). On le fait avec l'image `room.png` et les trois jeux de paramètres `(d,opt)` = (0,0), (1,0), (1,1) :



`(d,opt)` = (0,0) affiche simplement le module de l'image et ramène l'intervalle des valeurs obtenues à l'intervalle d'entiers $[0, 255]$ de manière affine ; cela nous donne une image noire avec un unique point blanc au centre. En effet, le coefficient au centre est tellement grand qu'il écrase les autres (qui deviennent donc nuls après la transformation affine). On va donc appliquer un *seuillage* qui permet de considérer les variations de valeurs sur un intervalle plus restreint ; `(d,opt)` = (1,0) applique un seuillage de 1% : on ne considère que les valeurs qui représentent une part de 99% et les valeurs hors de cet intervalles sont ramenées soit à 0 (si elles sont en-dessous) soit à 255. Le résultat obtenu est déjà beaucoup plus intéressant. Comme les valeurs sont très petites,

en utilisant les paramètres `(d,opt)` = (1,1), on applique un logarithme qui va encore gonfler les variations ; d'où un résultat plus satisfaisant.

On peut par ailleurs observer des symétries sur le module, qui correspondent aux régularités que l'on avait dans l'image de départ (verticales pour les meubles et relativement horizontales pour le tissu).

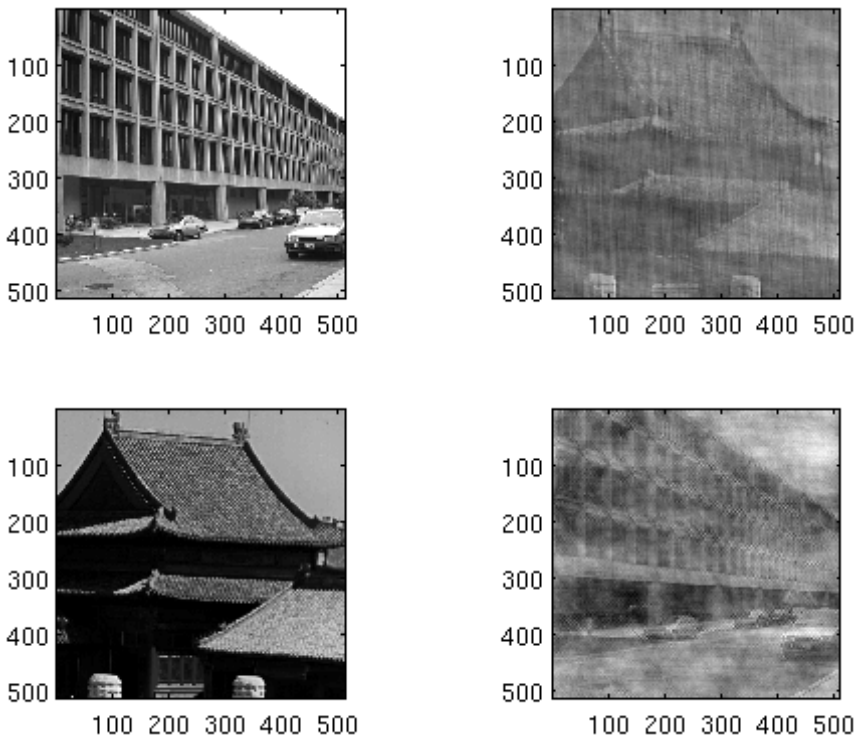
▷ QUESTION 7 : Puisque l'on peut séparer le module et la phase d'une image, on va les intervertir entre deux images et observer le résultat. On commence avec les images `building.png` et `tile.png`.

```
u = double(imread('building.png'));
u = u(:,:,1);
v = double(imread('tile.png'));
v = v(:,:,1);
fu = fft2(u);
modu = abs(fu); phiu = fu./modu;
fv = fft2(v);
modv = abs(fv); phiv = fv./modv;
echu = abs(fu).*phiv;
echv = abs(fv).*phiu;
imu = real(ifft2(abs(fu).*phiv));
imv = real(ifft2(abs(fv).*phiu));

%visualisation
subplot(2,2,1);
vis_img_mat(u,1);
subplot(2,2,2);
vis_img_mat(imu,1);
subplot(2,2,3);
vis_img_mat(v,1);
subplot(2,2,4);
```

On commence par charger les deux images dans `u` et `v`. MATLAB considérant (à tort) que les images sont en couleurs, on est forcé de faire une légère manipulation pour que `u` et `v` soient dans un format correct.
On applique `fft2` ; cela nous donne une matrice complexe de coefficients $r_{ij}e^{i\theta_{ij}}$, r_{ij} étant réel. Notre but est de séparer les r_{ij} des $e^{i\theta_{ij}}$. Pour cela, on utilise `abs` qui va calculer le module de chacun des coefficients de la matrice, puis utiliser la division terme-à-terme de MATLAB (`.\`) pour récupérer la matrice des phases. Pour intervertir module et phase des deux images, il nous suffit donc d'échanger les matrices de phase et d'effectuer une multiplication terme-à-terme (`.*`). Il ne reste plus qu'à calculer l'image à partir des matrices obtenues avec `ifft2`.
En théorie, les matrices obtenues après TDF inverse sont réelles ; cependant, des erreurs de calculs peuvent introduire de petites parties imaginaires dans la matrice ; on s'en débarrasse en appliquant la fonction `real` qui va ne conserver que la partie réelle de nos matrices.

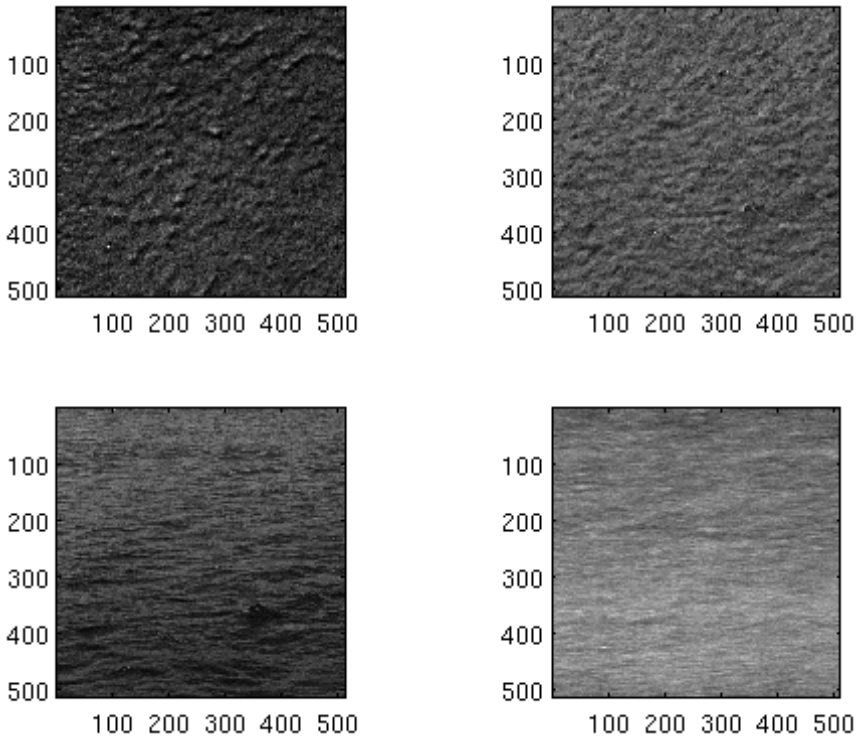
On obtient alors les images suivantes (en haut à gauche : `building.png` originale ; en haut à droite : module de `building.png`, phase de `tile.png` ; en bas à gauche : `tile.png` ; en bas à droite : module de `tile.png`, phase de `building.png`) :



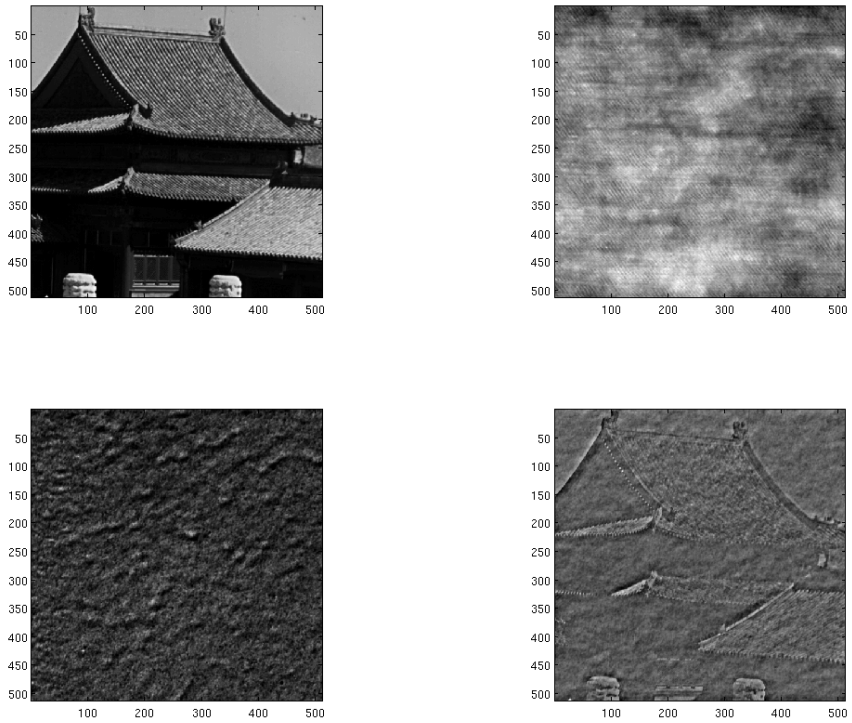
On observe déjà que l'on peut reconnaître les traits de `tile.png` dans l'image où on a conservé sa phase (il en est de même pour `building.png`) ; cela tendrait à montrer que la phase contient des informations sur

les formes de l'image. Par ailleurs, il semble que les symétries (verticales pour `building.png` et obliques pour `tile.png`) des images soient conservées quand on garde le module ; le module enregistrerait des informations sur les symétries (ce que l'on a déjà pu observer dans la manipulation précédente).

Testons la même expérience sur les couples d'images (`sand.png`, `water.png`) :



et (`tile.png`, `sand.png`) :



Il est très visible dans le dernier échange que la texture semble conservée par le module (et on distingue très bien l'empreinte de `tile.png` dans l'image où on a gardé la phase de cette image).

III Effet de Gibbs et zooms avant

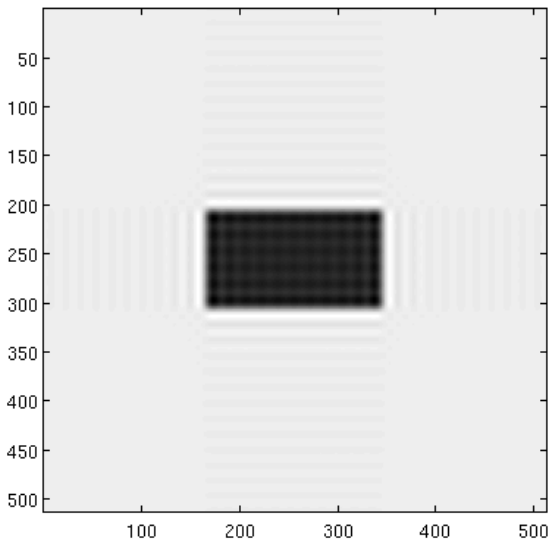
III.1 Effet de Gibbs sur les images

L'effet de Gibbs se manifeste lorsque l'on supprime les hautes fréquences d'un signal et apparaît sous la forme d'oscillations autour des discontinuités.

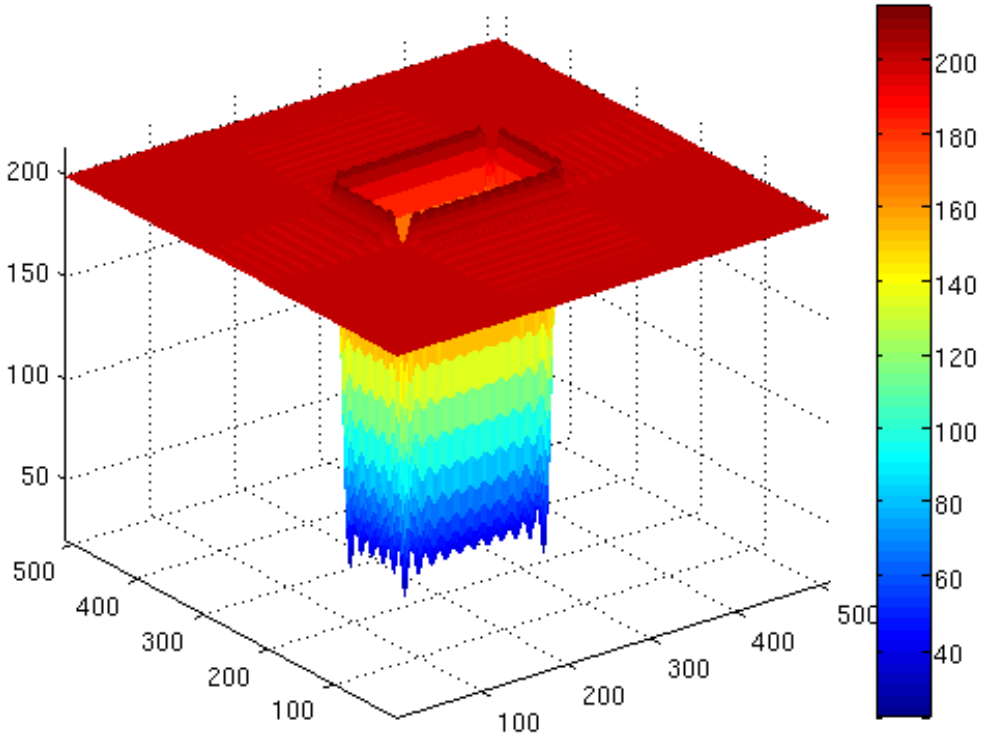
Pour observer ce phénomène, on crée une image présentant une discontinuité importante (par exemple un rectangle foncé sur un fond clair). On applique alors la TDF sur cette image, puis on annule les fréquences qui ne sont pas sur le carré central 64×64 et on calcule la nouvelle image obtenue :

```
rectangle = 200*ones(512,512);
rectangle(206:305,166:345) = 50*ones(100,180);
imagesc(uint8(rectangle),[0,255]);
colormap gray;
axis image;

vis_mod_img(rectangle,1,0);
fr = fftshift(fft2(rectangle));
fr2 = zeros(512,512);
fr2(225:288,225:288) = fr(225:288,225:288);
rectbf = real(ifft2(fftshift(fr2)));
vis_img_mat(rectbf,1);
surf(rectbf,'EdgeColor','none');
colormap(jet);
colorbar;
axis tight;
```



Les dernières instructions nous permettent de mieux visualiser le phénomène d'oscillations aux bords du rectangle :

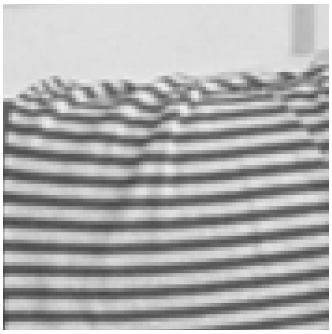
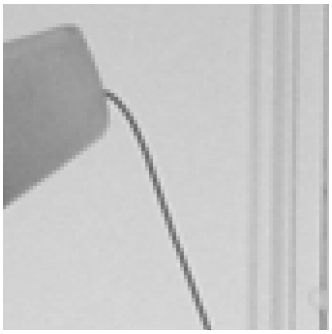


III.2 Zooms avant

▷ Question 8 : On teste plusieurs méthodes de zooms avant : on souhaite, partant d'une image d'une certaine taille, l'agrandir. Cette opération agrandit donc d'autant son spectre.

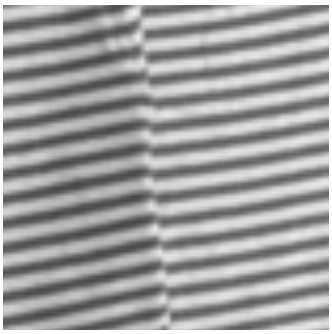
Méthode du pixel le plus proche

Elle consiste à remplacer un pixel par quatre pixels de même valeur. L'image devient alors moins fluide, plus "pixelisée" (puisque les pixels – pris comme plus petits constituants de l'image – ont grossi). C'est la manière la plus grossière d'effectuer un zoom.

Méthode par interpolation bilinéaire

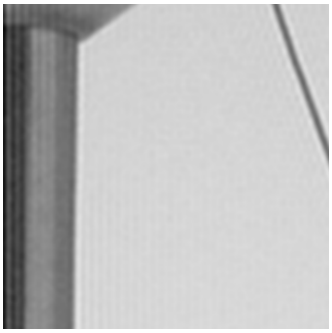
On considère les pixels de l'image d'origine comme étant des échantillons et on interpole entre les points :

Cette méthode détruit une partie des discontinuités (puisqu'elle suppose que l'image est un signal continu). L'image obtenue paraît donc floue. Elle nécessite par ailleurs d'interpoler tous les points.

Méthode par zéro-padding

Elle consiste à agrandir le spectre en ajoutant des valeurs nulles aux bords. Celles-ci correspondent aux hautes fréquences de la nouvelle image. C'est donc comme si on était parti d'une image plus grande et que l'on avait annulé les hautes fréquences du spectre : on observe donc un effet de Gibbs (notamment au niveau de la cordelette, et bien sûr aux bords de l'image) :

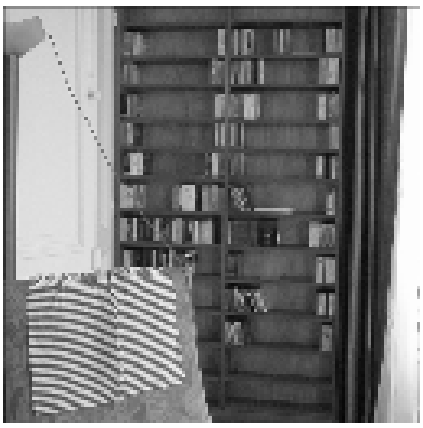



IV Zoom arrière

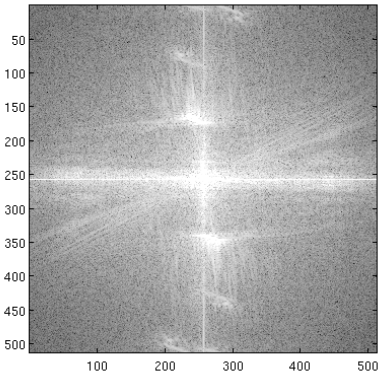
▷ Question 9 : On souhaite maintenant effectuer un zoom arrière, c'est-à-dire réduire la taille de l'image.

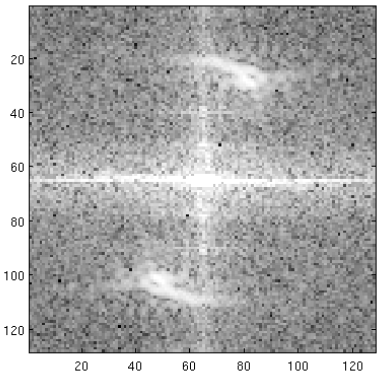
Zoom arrière par sous-échantillonnage

On crée une image plus petite en ne gardant qu'un pixel sur `r` ; on obtient donc une image 16 fois plus petite :

On remarque que cette opération affecte la direction des rayures du tissu qui n'est plus la même que dans l'image d'origine.





Cela se voit dans le spectre qui ne présente plus les mêmes symétries que celui de départ (cf. *Phénomène de repliement du spectre* plus loin).

Zoom arrière d'une onde pure haute fréquence

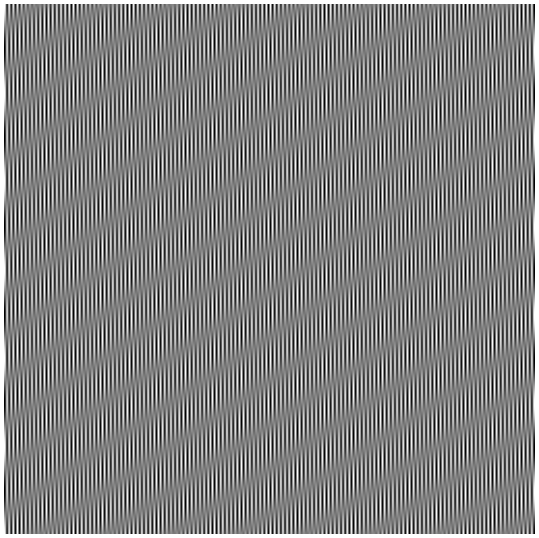
On crée une onde pure haute fréquence de taille 512 de spectre qui est nul sauf en deux points :

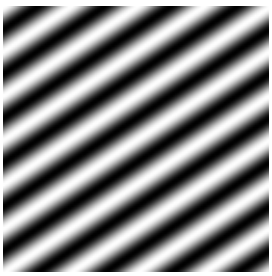
```
N = 512;
f1 = 250;
f2 = 124;
tfu = zeros(N);
tfu(f1+N/2+1,f2+N/2+1) = 1;
tfu(-f1+N/2+1,-f2+N/2+1) = 1;
```

qui correspondent à deux points symétriques par rapport au centre ; on a donc bien le spectre d'une onde (le polynôme trigonométrique correspondant vaut, à un décalage d'indice près, $2\cos\dfrac{2\pi(f_1x+f_2y)}{N}$)

Lorsque l'on sous-échantillonne par 2, le polynôme devient $2\cos\dfrac{4\pi(f_1x+f_2y)}{N}$ (pour un sous-échantillonnage par 4, il vaut $2\cos\dfrac{8\pi(f_1x+f_2y)}{N}$) – on ne garde que les points pairs. Ainsi, on conserve toujours une onde

à chaque étape, mais avec moins de points, donc les maxima (resp. les minima) des points diminuent (resp. augmentent) ; comme on visualise les images avec l'option 1, les images sont de plus en plus contrastées (ce que l'on observe effectivement). La fréquence est par ailleurs doublée à chaque échantillonnage.





Zoom arrière par coupure fréquentielle

Lorsque l'on sous-échantillonne une image, on peut assister à un phénomène dit d'*aliasing* : des coefficients de hautes fréquences viennent s'ajouter en plus (ce sont des "alias") ; ce phénomène de repliement du spectre affecte en particulier certaines symétries ; on élimine cet effet en annulant les hautes fréquences avant l'échantillonnage (on élimine ainsi les alias potentiels). On peut voir le résultat dans l'exemple suivant :

Cependant, on voit que, si le sous-échantillonnage n'a pas créé des symétries inexistantes dans l'image d'origine, il en a supprimé (ces informations étaient sûrement contenues dans les hautes fréquences).

V Translation non entière et rotation

▷ Question 10 : Pour une image $(u_{k,l})$, la translation de vecteur (a, b) donne le polynôme trigonométrique suivant :

$$P_u(x-a, y-b) = \sum_{m=-\frac{N}{2}}^{\frac{N}{2}-1} \sum_{n=-\frac{N}{2}}^{\frac{N}{2}-1} \tilde{u}_{m,n} e^{-\frac{2i\pi ma}{N}} e^{-\frac{2i\pi nb}{N}} e^{\frac{2i\pi mx}{N}} e^{\frac{2i\pi ny}{N}}$$

soit

$$P_v(x, y) = \sum_{m=-\frac{N}{2}}^{\frac{N}{2}-1} \sum_{n=-\frac{N}{2}}^{\frac{N}{2}-1} \tilde{v}_{m,n} e^{\frac{2i\pi mx}{N}} e^{\frac{2i\pi ny}{N}}$$

avec $\tilde{v}_{m,n} = \tilde{u}_{m,n} e^{-\frac{2i\pi ma}{N}} e^{-\frac{2i\pi nb}{N}}$.

On peut alors écrire la fonction `translation_fourier` :

```
function v = translation_fourier(u,a,b)
u = u(:,:,1);
fu = fft2(u);
N = size(u,1);
A = zeros(N);
for m = 1:N
    for n = 1:N
        A(m,n) = exp(-2*i*pi*m*a/N)
                 *exp(-2*i*pi*n*b/N);
    end
end
fv = fu.*A;
v = real(ifft2(fv));
```

Il s'agit de calculer l'image $(v_{k,l})$ définie précédemment ; pour cela, on a besoin des coefficients $\tilde{u}_{m,n}$. Il faut ensuite les multiplier par les complexes $e^{-\frac{2i\pi ma}{N}} e^{-\frac{2i\pi nb}{N}}$; pour ce faire, on crée la matrice des $(e^{-\frac{2i\pi ma}{N}} e^{-\frac{2i\pi nb}{N}})$ pour (n, m) variant de 1 à N, que l'on multiplie ensuite membre à membre avec la TDF de u. On obtient ainsi la TDF de v. La TDF inverse nous donne alors l'image correspondante.

On obtient le résultat suivant en testant sur l'image `room.png` par une translation (50,50) :