
Manipulations de traitement d'images

J.-F. Bercher

November 28, 2013

Part I

Manipulations de traitement d'images

Auteur: J.-F. Bercher 28 novembre 2013

1 Introduction

Ce TD-TP a pour but d'illustrer certaines notions de traitement du signal, appliquées sur des objets 2D; on aborde ainsi le traitement d'images. En particulier, on abordera les problèmes de représentation et de filtrage (dans le domaine spatial et dans le domaine transformé) (passe-bas et passe haut). Par la suite, on examinera des problèmes d'échantillonnage, voire de restauration d'images. Les manipulations se feront sous Python. Vous récupérerez les fichiers adéquats sur la page Moodle de l'unité, ou sur la page web de l'enseignant.

On utilisera donc à nouveau Python, avec les bibliothèques scientifiques `scipy` et `numpy`, ainsi que les modules `scipy.signal` et `scipy.ndimage`. Quelques indications sur les commandes et les fonctions utiles sont fournis dans l'énoncé. Pour le reste, il vous faudra utiliser l'enseignant et la documentation en ligne. Votre serviteur, dans le but noble de vous faciliter tâche sans sacrifier la compréhension, a préparé quelques fonctions utiles, notamment :

`rect2` -- crée un rectangle centré en 2D `filtre_passebande_2d` -- crée un filtre passe-bande en 2D (dans le domaine fréquentiel) `showfft2` -- affiche une image d'une TF 2D, correctement centrée et normalisée `mesh` -- affiche une représentation "3D" d'un objet

Pour lire un fichier image, vous utiliserez la fonction `imread`

Pour afficher une image en niveaux de gris, vous pourrez utiliser

`imshow(S, cmap='gray', origin='upper')`

```
In [2]: from __future__ import division
        from __future__ import print_function

        from pylab import *
        from scipy import ndimage as ndi
        from scipy import signal
        from numpy.fft import fft, ifft, fft2, ifft2
        from scipy.ndimage import convolve as convolve
        # pour les représentations 3D
        from mpl_toolkits.mplot3d.axes3d import Axes3D
```

In [3]:

```
# from defs_tp_img import *
#
## Definitions
# =====

def isodd(num):
    return num & 1 and True or False
# or num%2==1

##
def rect2(L,N):
    """ Rend un rectangle de taille (2L+1)x(2L+1) \n
    centré sur une dimension totale de NxN"""
    x=zeros((N,N))
    milieu=((N+1)/2,(N+1)/2) if isodd(N) else ((N)/2+1,(N)/2+1)
    x[milieu[0]-L:milieu[0]+L+1,milieu[1]-L:milieu[1]+L+1]=1
    return x

##
def filtre_passebande_2d(centre,largeur,dimension_totale):
    """ Retourne la réponse en fréquence d'un filtre passe-bande
    centré sur 'centre', de demi-largeur **largeur** et symétrisé
    en fréquence (le zéro est placé au centre de l'image en f)\n
    centre : fréquence centrale du filtre passe-bande (en points) -- les fréquences
    normalisées correspondantes sont ainsi centre/(dimension_totale/2) \n
    largeur : demi-bande de fréquence (en points) du filtre passe-bande\n
    dimension_totale : dimension de l'image\n
    Auteur : jfb"""
    N=dimension_totale
    milieu=((N+1)/2,(N+1)/2) if isodd(N) else ((N)/2+1,(N)/2+1)
    #
    H=zeros((N,N))
    H[milieu[0]+centre[0]-largeur:milieu[0]+centre[0]+largeur+1,milieu[1]+centre[1]-largeur:milieu[1]+centre[1]+largeur+1]=1
    centre=-array(centre) # Le symétrique en fréquence
    H[milieu[0]+centre[0]-largeur:milieu[0]+centre[0]+largeur+1,milieu[1]+centre[1]-largeur:milieu[1]+centre[1]+largeur+1]=1
    return H

## représentation en fréquence 2D
def showfft2(X,Zero='uncentered', Freq='normalized', num=None,cmap='hot'):
    """Affiche une image, dans le domaine de Fourier 2D, \n
    la représentation étant centrée et en fréquences réduites\n
    **/!\ la fonction *showfft2* **ne calcule pas la TF**. Les données passées sont
    Paramètres :
    -----
    X : données 2D dans le domaine de Fourier \n
    Num: numéro de figure (optionnel)\n
    Zero='uncentered' par défaut, la TF passée est supposée non centrée et un ffts
    Freq='normalized' par défaut; sinon on trace en points \n
    ``Auteur: jfb``
    """
    N,M=shape(X)
    figure(num)
    if Zero!='centered':
        X=fftshift(X)
    if Freq=='normalized':
        extent=(-0.5, 0.5, -0.5, 0.5)
    else:
        extent=(-N/2, N/2, -N/2, N/2)
    im=imshow(X,aspect='auto',origin='lower', extent=extent, cmap=cmap)
    if Freq=='normalized':
        ticks=arange(-0.5,0.6,0.1)
        plt.xticks(ticks)
        plt.yticks(ticks)
    colorbar()

def mesh(obj3D,numfig=None,subplt=(1,1,1),cmap='hot'):
```

```

"""
Emule le mesh de matlab : trace
une représentation en 2.5D \n
de l'objet obj3D
Auteur: jfb
"""
fig = plt.figure(numfig)
ax = fig.add_subplot(subplt[0],subplt[1],subplt[2], projection='3d')
m,n=shape(obj3D)
mm=range(m)
nn=range(n)
X,Y=meshgrid(mm,nn)
ax.plot_surface(X, Y, obj3D, rstride=max([round(m/50), 1]), cstride=max([round(n/50), 1]))

def saltpepper(percent=85, shape=None):
    s=numpy.random.random(size=shape)
    d=where(s>percent/100)
    out=ones(shape=shape)
    out[d]=0
    return out

```

2 Représentation fréquentielle - filtrage en fréquence

2.1 Une petite sinusoïde

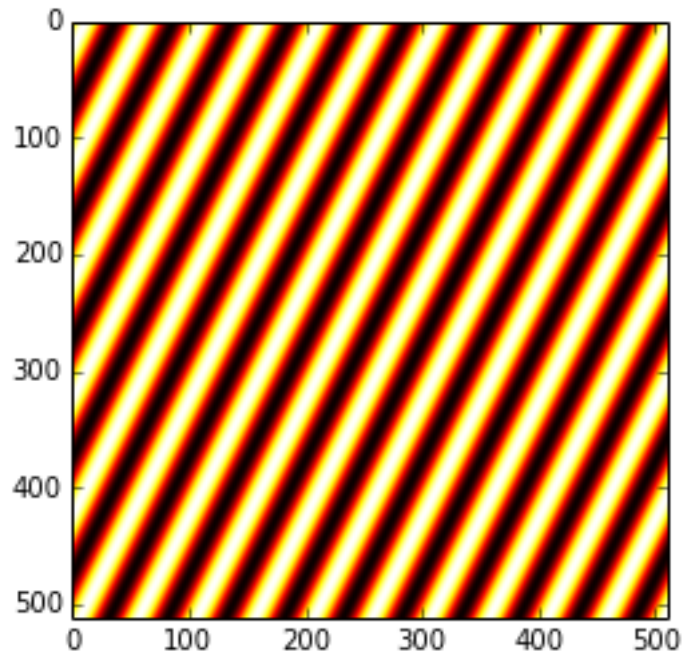
- Créez une sinusoïde 2D d'équation $f(x,y) = \sin(2\pi(f_x x + f_y y))$, avec f_x et f_y choisis entre 0.02 à 0.2, sur $N \times N$ points (par exemple $N = 512$). Pour l'implantation, vous pourrez utiliser une boucle `for`, ou une double liste compréhension, ou encore la belle fonction `fromfunction()` et une fonction `lambda`. Visualisez le résultat en utilisant la fonction `imshow`.

```

In [4]: fx,fy= 0.02, 0.01
x=arange(512)
y=arange(512)
# sous la forme d'une double liste compréhension
S=[[sin(2*pi*(fx*xx+fy*yy)) for xx in x] for yy in y]
# sinon, il y a ça qui est assez fort aussi...
Z=fromfunction(lambda x,y: sin(2*pi*(fx*x+fy*y)), (512,512))
# Visualisation
imshow(S, cmap='hot', origin='upper')
<matplotlib.image.AxesImage at 0x7fde66180ad0>

```

Out [4]:

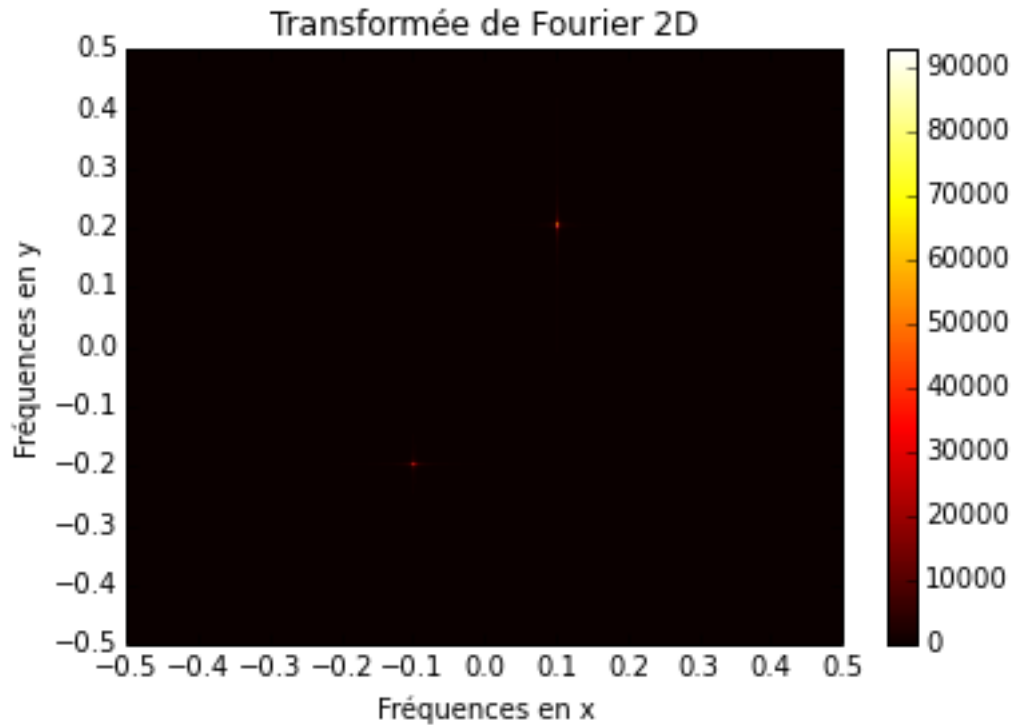


- Calculez la Transformée de Fourier 2D de $f(x, y)$ (en utilisant la fonction `fft2`) et visualisez le module (fonction `abs`) du résultat, via `showfft2`. Quelles sont les significations des axes ? Quelles sont les fréquences spatiales de la sinusoïde ? Expérimenter pour plusieurs fréquences f_x, f_y .

```
In [5]: # On change la fréquence pour que ça se voit mieux
fx,fy= 0.2, 0.1
x=arange(512)
y=arange(512)
# sinon, il y a ça qui est assez fort aussi...
S=fromfunction(lambda x,y: sin(2*pi*(fx*x+fy*y)), (512,512))

# Visualisation de la TF2D
showfft2(abs(fft2(S)),num=4)
title('Transformée de Fourier 2D')
xlabel('Fréquences en x')
ylabel('Fréquences en y')
<matplotlib.text.Text at 0x7fde6612b190>
```

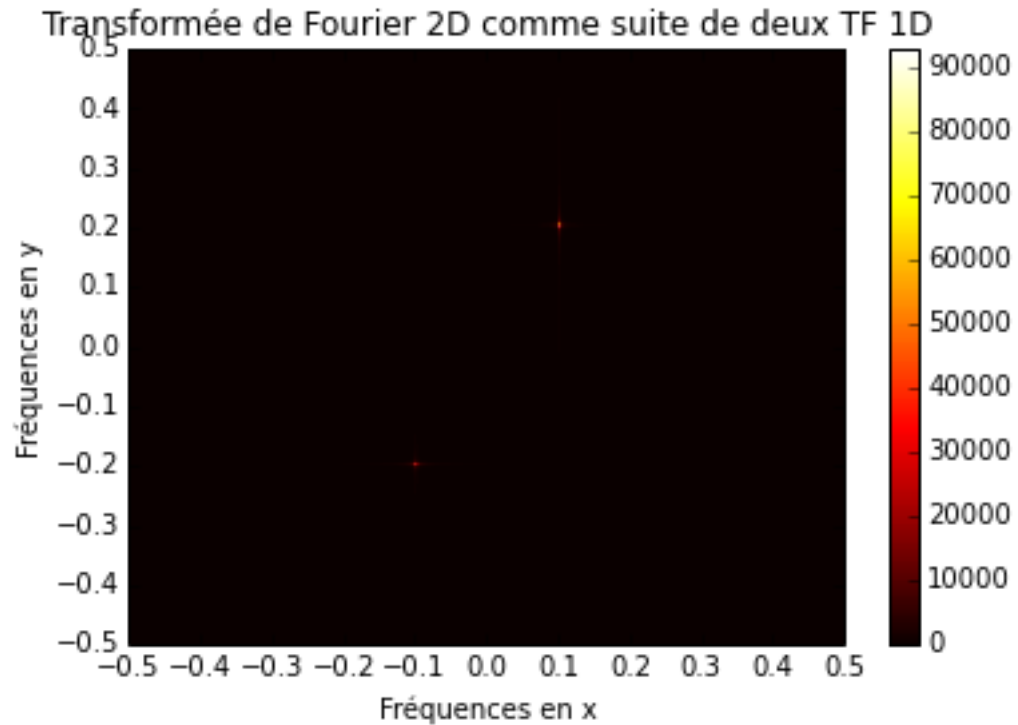
Out [5]:



- Montrez “théoriquement” et vérifiez sur machine que la transformée de Fourier 2D peut être obtenue comme la succession de transformées de Fourier 1D (fonction `fft`) appliquées sur les lignes puis les colonnes (ou inversement). Vous utiliserez le fait que la fonction `fft` prend un paramètre `axis` qui est la dimension sur laquelle est calculée la `fft`

```
In [6]: showfft2(abs(fft(fft(S,axis=0),axis=1)),num=5)
        title('Transformée de Fourier 2D comme suite de deux TF 1D')
        xlabel('Fréquences en x')
        ylabel('Fréquences en y')
        # On constate donc que le TF2D=succession de deux TF 1D
        # sur chaque dimension
        <matplotlib.text.Text at 0x7fde6603abd0>
```

Out [6]:

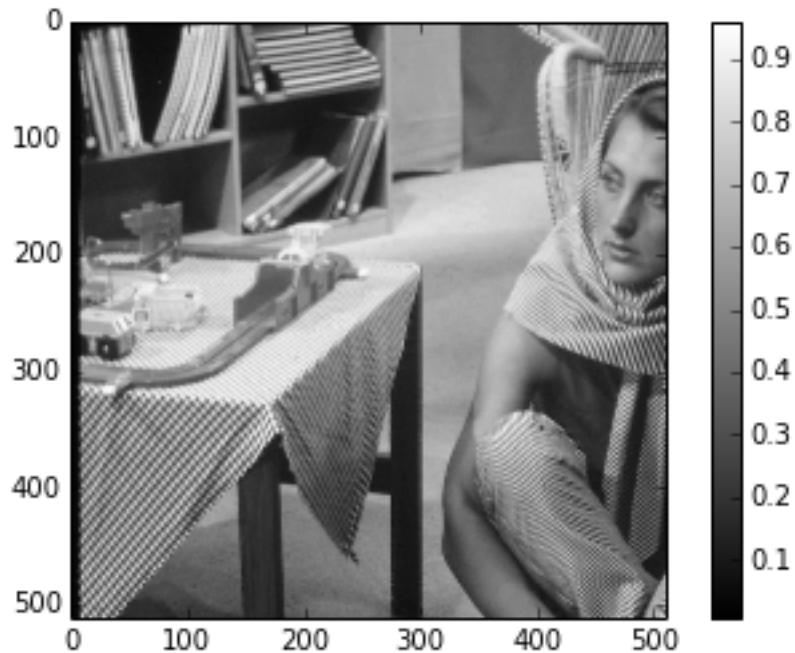


2.2 Barbara

- Chargez l'image de Barbara, par `imread('barbara.png')` et visualisez la.

```
In [7]: ### Lecture et affichage de l'image  
B=imread('barbara.png')  
figure(1)  
#pylab.rcParams['figure.figsize'] = (16.0, 8.0)  
#pylab.rcParams['savefig.dpi'] = 200  
#import matplotlib  
#matplotlib.rcParams['savefig.dpi'] = 72  
imshow(B,cmap='gray',origin='upper')  
  
colorbar()  
<matplotlib.colorbar.Colorbar at 0x7fde65f1d050>
```

Out [7]:



En réalité il y a un tout petit souci dans la visualisation ici. Dans le notebook, l'image est discrétisée (sous échantillonnée) et il y a quelques détails qui disparaissent (pb de recouvrement – aliasing) que l'on retrouvera un peu plus tard dans le cours

```
matplotlib.rcParams['savefig.dpi']
```

In [8]: 72

Out [8]:

- Visualisez la représentation en fréquence `showfft2`, en échelle logarithmique (prendre `log(abs())` !)

```
help(showfft2)
```

In [9]: Help on function `showfft2` in module `__main__`:

```
showfft2(X, Zero='uncentered', Freq='normalized', num=None,
cmap='hot')
```

Affiche une image, dans le domaine de Fourier 2D,

la représentation étant centrée et en fréquences réduites

`**/!\ la fonction *showfft2* **ne calcule pas la TF**. Les données passées sont sensées être des données fréquentielles`

Paramètres :

X : données 2D dans le domaine de Fourier

Num: numéro de figure (optionnel)

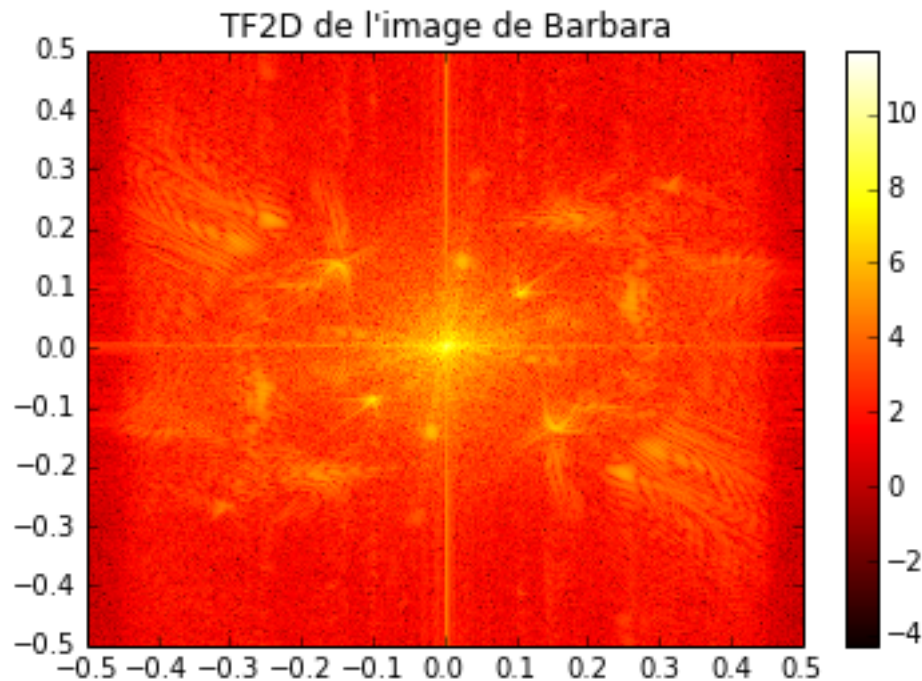
Zero='uncentered' par défaut, la TF passée est supposée non centrée et un `fftshift` est appliqué

Freq='normalized' par défaut; sinon on trace en points

``Auteur: jfb``

```
In [10]: showfft2(log(abs(fft2(B))))
         title("TF2D de l'image de Barbara")
         <matplotlib.text.Text at 0x7fde65e6b690>
```

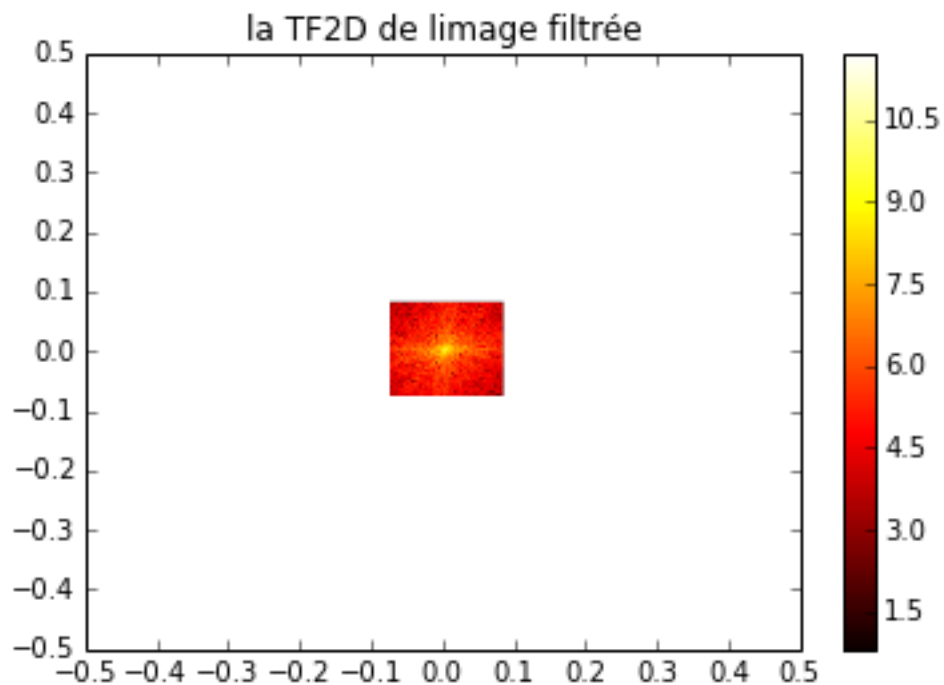
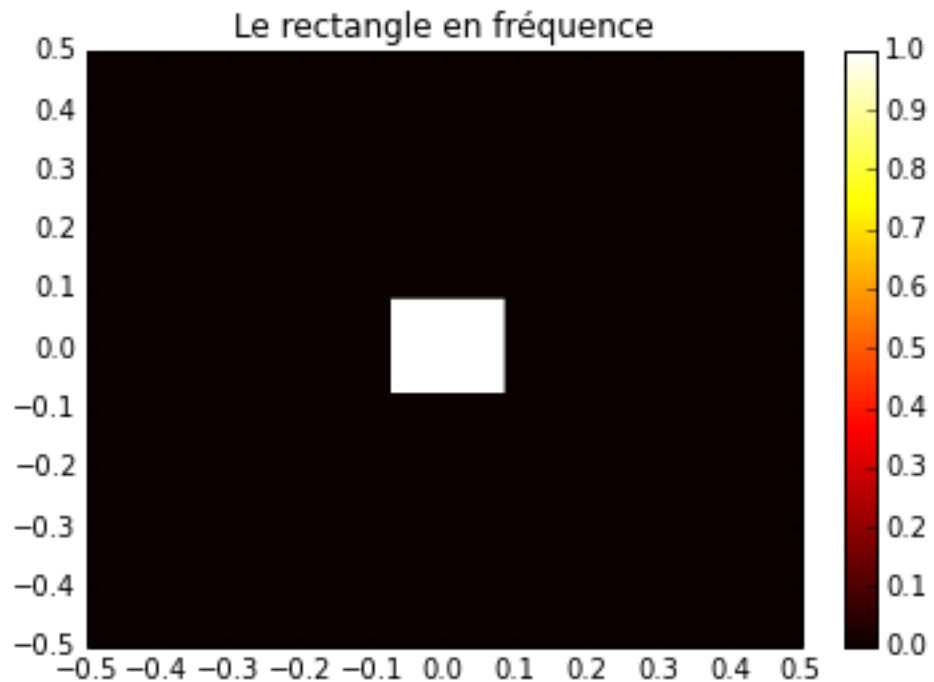
Out [10]:



- Filtrez cette image à l'aide d'un filtre de réponse en fréquence rectangulaire (utilisez la fonction `rect2`), pour des rectangles de demi-largeur 40, 80, 100. Visualisez les différentes images obtenues – filtrées passe-bas, ainsi que les différences à l'image de départ. Observations, conclusions.

```
In [11]: L=40
         H=rect2(L,512)
         showfft2(H,Zero='centered') ## !
         title('Le rectangle en fréquence')
         # Filtrage
         Bf_filtered=fft2(B)*fftshift(H)
         showfft2(log(abs(Bf_filtered)))
         title('la TF2D de l''image filtrée')
         -c:7: RuntimeWarning: divide by zero encountered in log
         <matplotlib.text.Text at 0x7fde65cf9650>
```

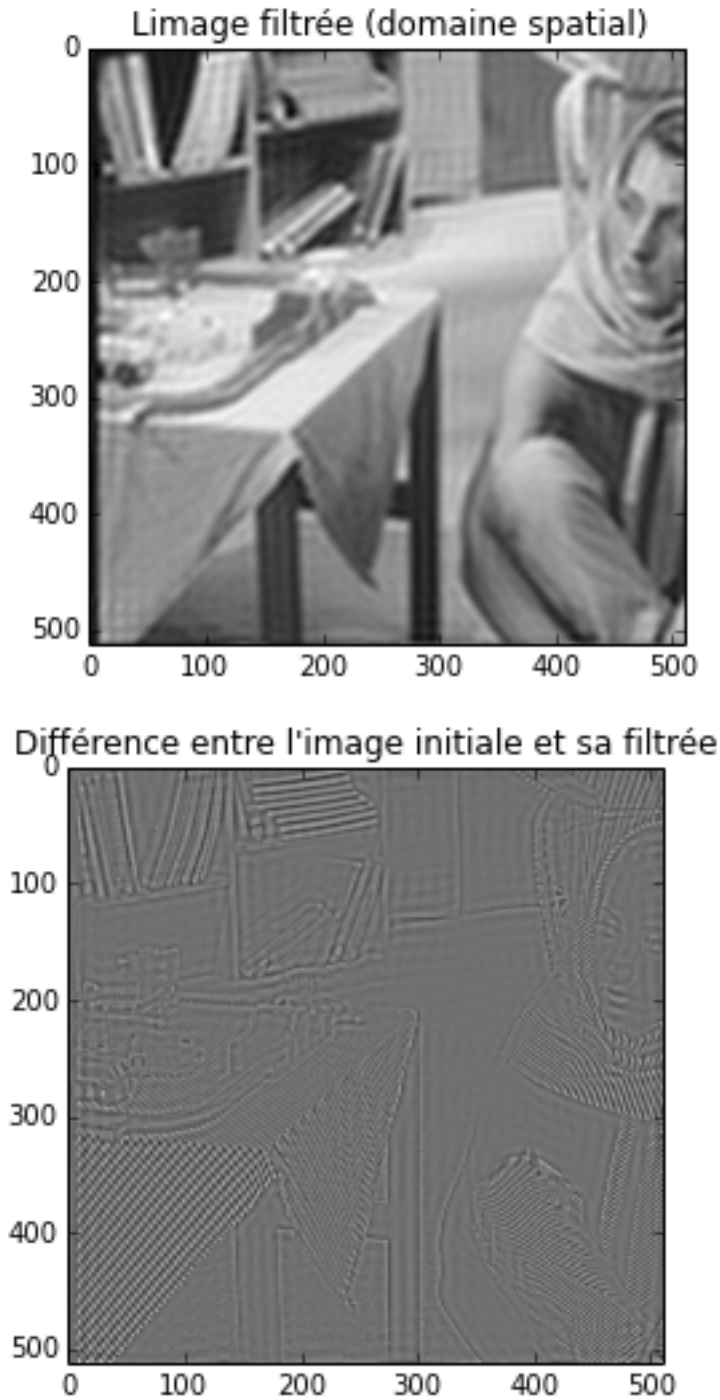
Out [11]:



```
In [12]: #
imshow(real(ifft2(Bf_filtered)), cmap='gray', origin='upper')
title('L' image filtrée (domaine spatial)')
# regarder également L=80, L=100
# Les différences :
figure()
imshow(B-real(ifft2(Bf_filtered)), cmap='gray', origin='upper')
title("Différence entre l'image initiale et sa filtrée")
```

```
<matplotlib.text.Text at 0x7fde65d9b150>
```

Out [12]:



Ce que l'on voit apparaître dans l'image de différences, ce sont bien des hautes fréquences, qui sont les "détails" de l'image de départ.

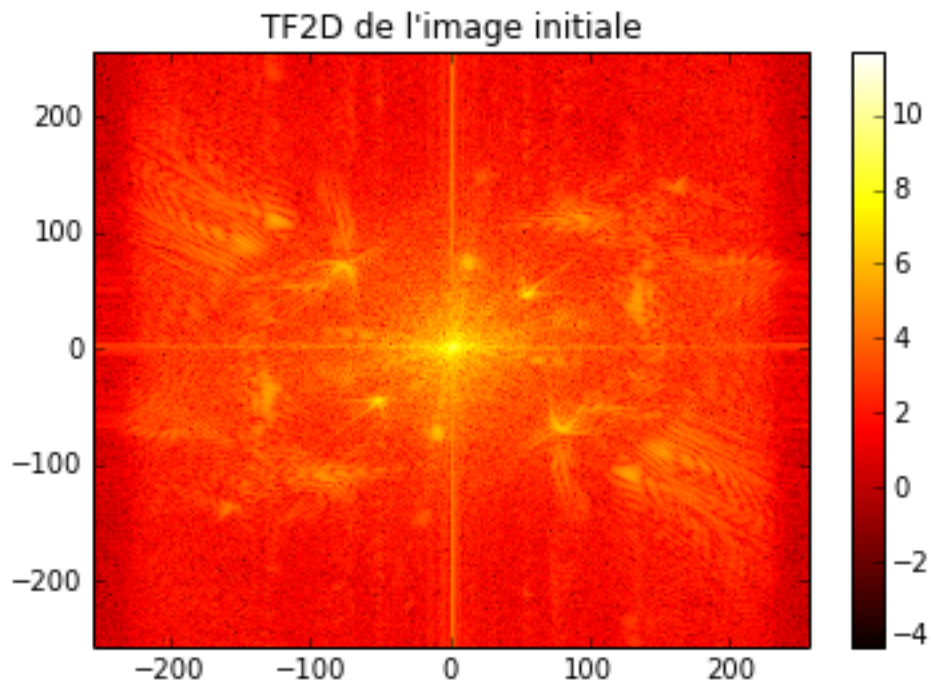
- Construisez une réponse en fréquence qui élimine sélectivement les fréquences situées autour des points (46,54) et (-70,79), par exemple sur un voisinage de ± 10 points. Pour ce faire, vous utiliserez le filtre passe-bande `filtre_passebande_2d` et créez un réjecteur de fréquence par `1-filtre_passebande_2d`. Réexaminez la TF2D de Barbara, graduée en points, afin de comprendre ce que vous faites. Appliquez ce filtre à l'image de départ et interprétez le résultat obtenu dans le domaine spatial. *Observez la nappe !* Visualisez

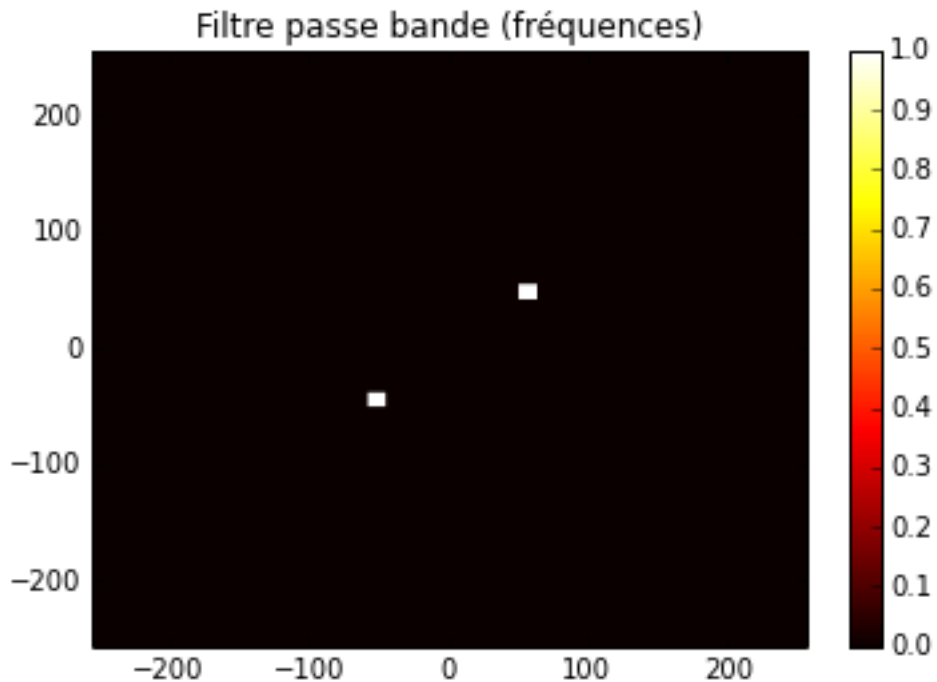
également l'image différence.

```
In [13]: # Filtrage à encoche
B=imread('barbara.png');
Bf=fft2(B)
showfft2(log(abs(Bf)),Zero='uncentered',Freq='unnorm') ## !
title("TF2D de l'image initiale")
## figure(3)
X=filtre_passebande_2d((46,54),6,512)
# Attention ; quand on adresse le tableau par A(x,y), les "x" ce sont les
# lignes (1er indice) et les "y" les colonnes, donc pour les "coordonnées", il faut
# échanger les deux indices

showfft2(X,Zero='centered',Freq='unnorm')
title('Filtre passe bande (fréquences)')
<matplotlib.text.Text at 0x7fde65c07e50>
```

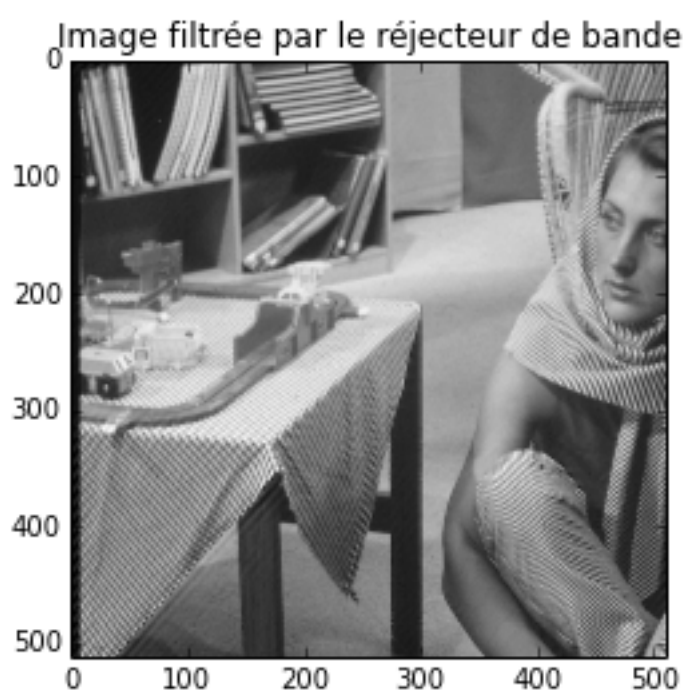
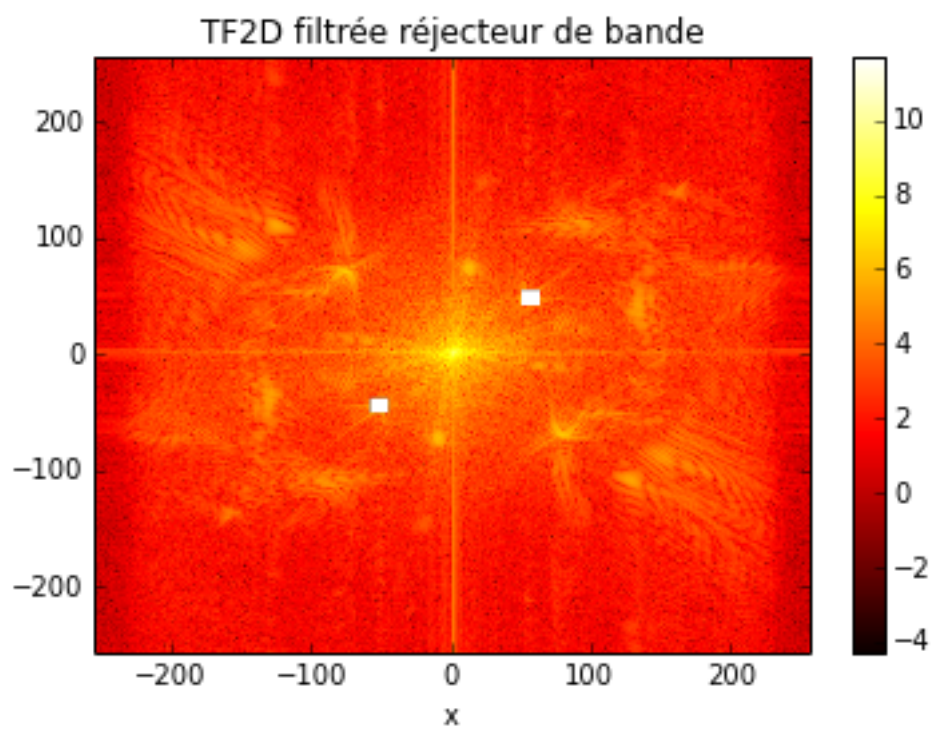
Out [13]:

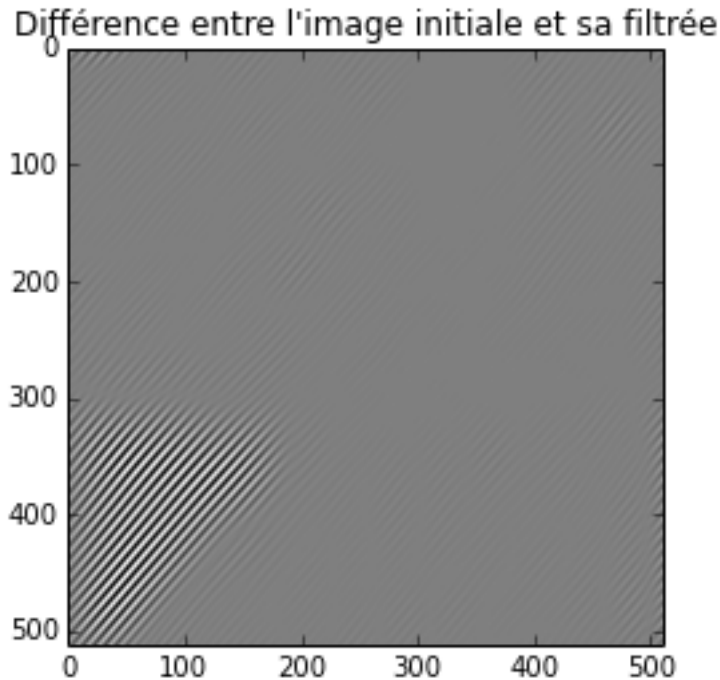




```
In [14]: X=filtre_passebande_2d((46,54),6,512)
## figure(4)
X=1-X
Bf_filtered=X*fftshift(Bf)
showfft2(log(abs(Bf_filtered)),Zero='centered',Freq='unnorm') ## !
title("TF2D filtrée réjecteur de bande")
xlabel('x')
# et la version spatiale est alors
B_filtered=real(ifft2(fftshift(Bf_filtered)))
figure()
imshow(B_filtered,cmap='gray',origin='upper')
title("Image filtrée par le réjecteur de bande")
# Les différences :
figure()
imshow(B-B_filtered,cmap='gray',origin='upper')
title("Différence entre l'image initiale et sa filtrée")
-c:5: RuntimeWarning: divide by zero encountered in log
<matplotlib.text.Text at 0x7fde65b23150>
```

Out [14]:





Observez la nappe !!

3 Filtrage par convolution

La fonction qui sera utile pour cette partie est la fonction `convolve` de `scipy.ndimage` (appel par `ndi.convolve`) si `ndimage` a été importé sous le nom de `ndi`. On utilisera également `standard_normal` pour ajouter un peu de bruit gaussien (facteur de 0.1); ou `saltpepper` pour du bruit en poivre et sel.

- Vous débuterez par l'implantation d'une convolution à deux dimensions, en comprenant les lignes suivantes :

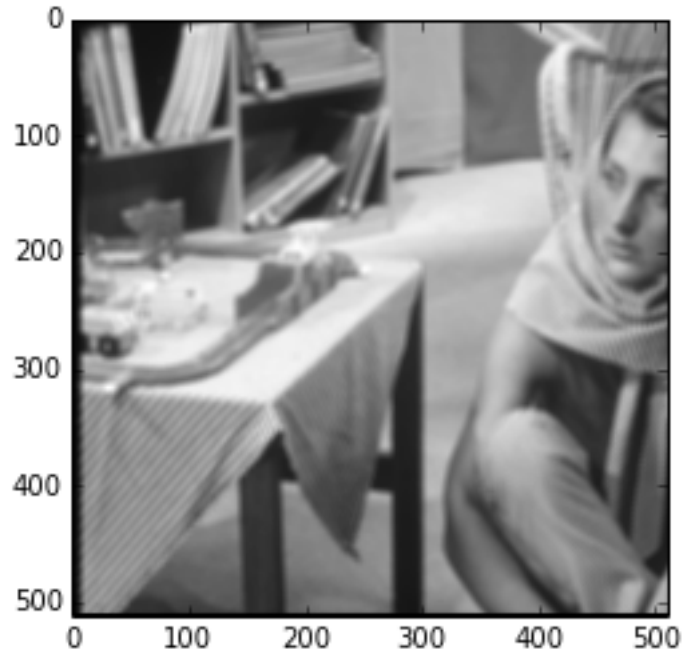
```
h=ones((2*ll+1,2*ll+1)) # h la RI
for m in range(ll,M-ll):
    for n in range(ll,N-ll):
        B_filtered[m,n]=sum(sum(h*B[m-ll:m+ll+1,n-ll:n+ll+1]))
```

Effectuez un filtrage passe-bas (h constant sur une demi largeur de 3 à 10) de l'image de Barbara, et examinez le résultat.

```
In [15]: (N,M)=shape(B)
ll=3
h=ones((2*ll+1,2*ll+1)) # la RI
for m in range(ll,M-ll):
    for n in range(ll,N-ll):
        B_filtered[m,n]=sum(sum(h*B[m-ll:m+ll+1,n-ll:n+ll+1]))

figure(9)
imshow(B_filtered,cmap='gray',origin='upper')
<matplotlib.image.AxesImage at 0x7fde65da0610>
```

Out [15]:

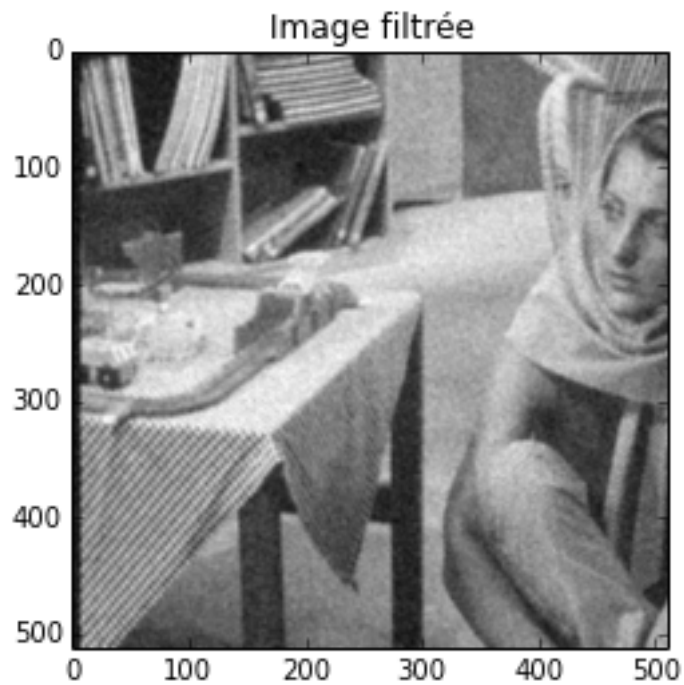
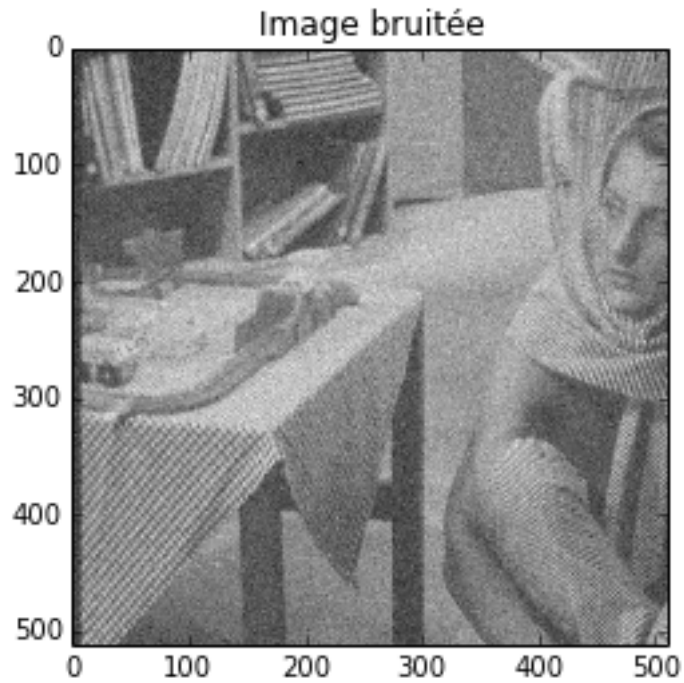


Comme on le voit, c'est très simple. Juste une somme de produits. En implantant mieux, et en tenant compte des effets de bords qu'on a évacués ici

- Reprenez ce filtrage en utilisant cette fois-ci la fonction `ndi.convolve`. Examinez l'effet du filtrage avec un bruit gaussien ou un bruit en poivre et sel. Vérifiez qu'il s'agit bien d'un filtrage passe-bas en visualisant la fonction de transfert en fréquence – utilisez un zero-padding lors du calcul de la TF2D, `fft2(h, s=(1000, 1000))` ; éventuellement, utiliser la fonction `mesh` pour la représentation).

```
In [17]: B=imread('barbara.png')
h=ones((4,4)) # 1a RI
B=B+0.1*standard_normal((512,512))
figure(9)
imshow(B,cmap='gray',origin='upper')
title('Image bruitée')
B_filtered=ndi.convolve(B,h)
figure(10)
imshow(B_filtered,cmap='gray',origin='upper')
title('Image filtrée')
<matplotlib.text.Text at 0x7fde65a2cbd0>
```

Out [17]:

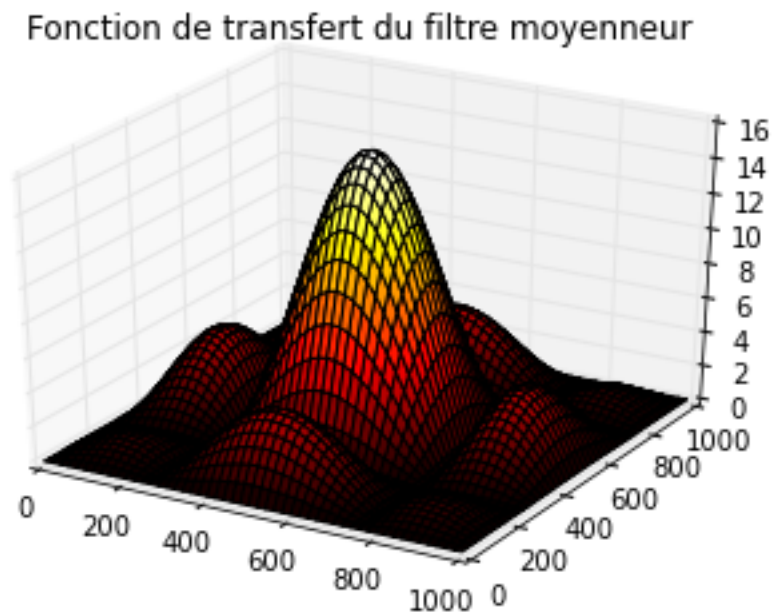
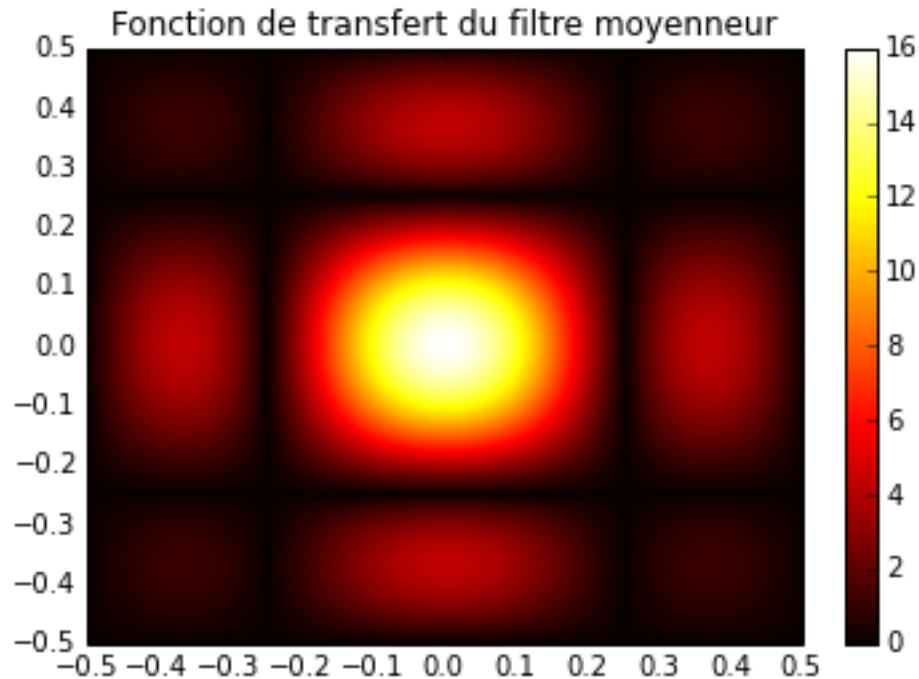


**** Fonction de transfert en fréquence ****

```
In [18]: # Regardons la TF2D de cette RI..
H=fft2(h,s=(1000,1000))
showfft2((abs(H)),Zero='uncentered',Freq='normalized', num=11) ## !
title('Fonction de transfert du filtre moyennneur')
#
mesh(abs(fftshift(H)),numfig=12)
title('Fonction de transfert du filtre moyennneur')
```


<matplotlib.text.Text at 0x7fde659de8d0>

Out [18]:



Conclusion : c'est bien passe-bas !

- Sur l'image de Barbara, puis sur l'image des cellules, ou de bactéries. Testez un gradient de Prewit ou de Sobel de réponses impulsionnelles

```
dx=np.array([[1.0, 0.0, -1.0],[1.0, 0.0, -1.0],[1.0, 0.0, -1.0],])
```

```
# dx=np.array([[1.0, 0.0, -1.0],[2.0, 0.0, -2.0],[1.0, 0.0, -1.0],]) #Sobel dy=np.transpose(dx)
```

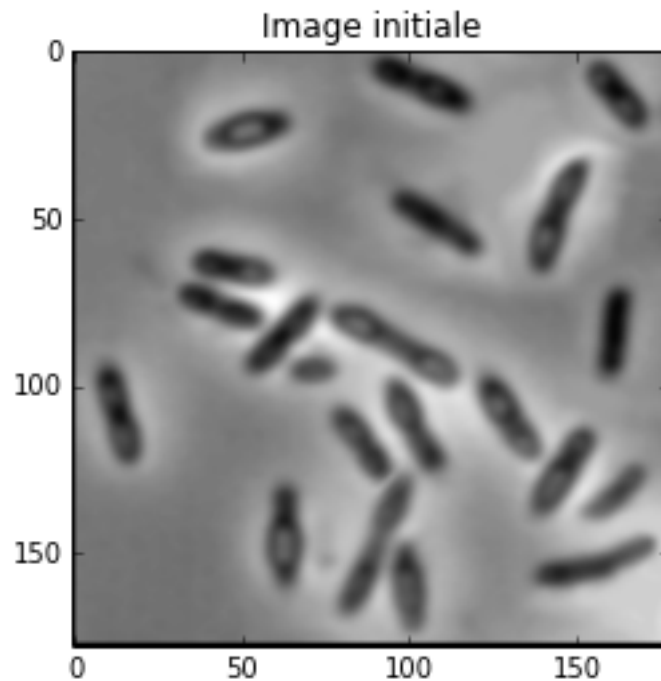
appliqué aux deux directions (x,y), par `ndi.convolve` et construisez une carte de direction.

NB : si D_x et D_y sont les images de gradient obtenues dans les directions x , y , la carte d'amplitude est $\sqrt{D_x.^2 + D_y.^2}$

Le module ndimage contient un certain nombre de filtres prédéfinis, par exemple `scipy.ndimage.filters.sobel`. Ici, on utilise la convolution directe sans faire appel à ces fonctions.

```
In [19]: B=imread('bacteria.png') # ou cellules.png
figure()
imshow(B,cmap='gray',origin='upper')
title('Image initiale')
# Sobel Filter
dx=np.array([[1.0, 0.0, -1.0],[2.0, 0.0, -2.0],[1.0, 0.0, -1.0],])
dy=np.transpose(dx)
fo1=ndi.convolve(B,dx, output=np.float64, mode='nearest')
fo2=ndi.convolve(B,dy, output=np.float64, mode='nearest')
magnitude=np.sqrt(fo1**2+fo2**2)

### Prewitt Filter
#dx=np.array([[1.0, 0.0, -1.0],[1.0, 0.0, -1.0],[1.0, 0.0, -1.0],])
#dy=np.transpose(dx)
```



```
In [20]: figure()
imshow(fo1,cmap='gray',origin='upper')
title('Image filtrée par un gradient de Sobel suivant x')
figure()
imshow(fo2,cmap='gray',origin='upper')
title('Image filtrée par un gradient de Sobel suivant y')
figure()
imshow(magnitude,cmap='gray',origin='upper')
title('Module du gradient de l\'Image filtrée par un gradient de Sobel')
<matplotlib.text.Text at 0x7fde64bf1c10>
```

Out [20]:

Image filtrée par un gradient de Sobel suivant x

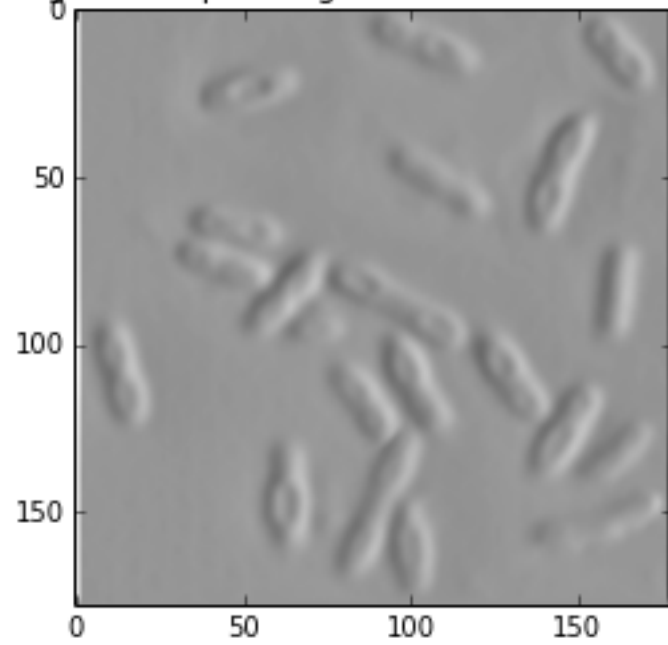
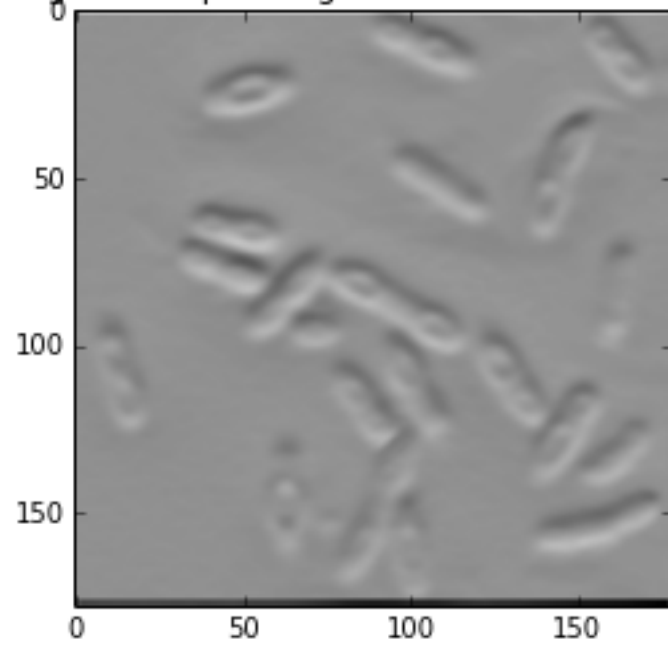
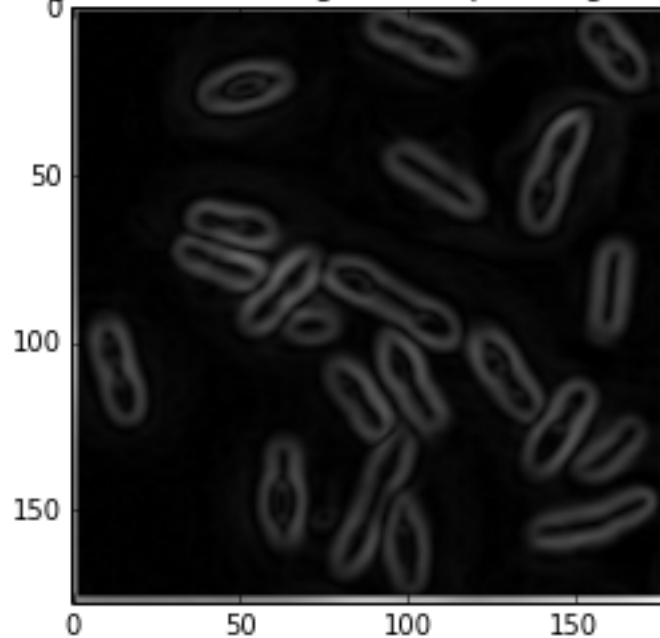


Image filtrée par un gradient de Sobel suivant y



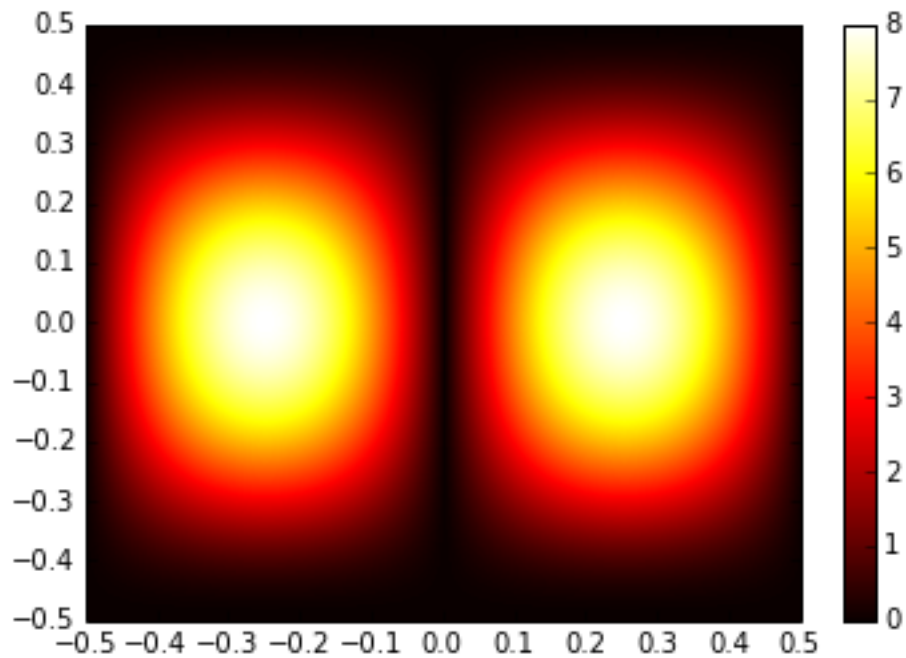
Module du gradient de l'image filtrée par un gradient de Sobel

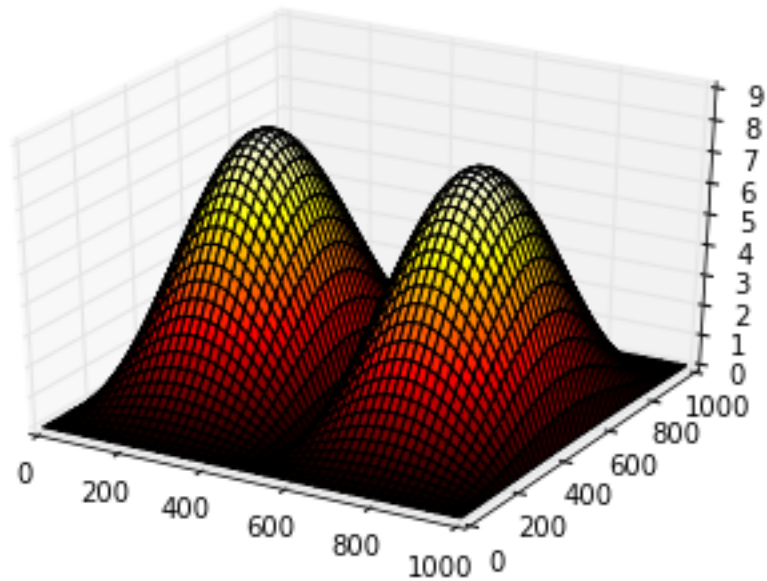


Fonction de transfert du Sobel :

In [21]:

```
# fonction de transfert du Sobel
Dx=fft2(dx,s=(1000,1000))
showfft2((abs(Dx)),Zero='uncentered',Freq='normalized') ## !
# ou mesh(abs(fftshift(Dx)))
mesh(abs(fftshift(Dx)))
# C'est bien un passe-haut !
```





Même chose avec Barbara

```
In [22]: B=imread('barbara.png') # ou cellules.png
figure()
imshow(B,cmap='gray',origin='upper')
title('Image initiale')
# Sobel Filter
dx=np.array([[1.0, 0.0, -1.0],[2.0, 0.0, -2.0],[1.0, 0.0, -1.0],])
dy=np.transpose(dx)
fo1=ndi.convolve(B,dx, output=np.float64, mode='nearest')
fo2=ndi.convolve(B,dy, output=np.float64, mode='nearest')
magnitude=np.sqrt(fo1**2+fo2**2)
#
figure()
imshow(fo1,cmap='gray',origin='upper')
title('Image filtrée par un gradient de Sobel suivant x')
figure()
imshow(fo2,cmap='gray',origin='upper')
title('Image filtrée par un gradient de Sobel suivant y')
figure()
imshow(magnitude,cmap='gray',origin='upper')
title('Module du gradient de l\'Image filtrée par un gradient de Sobel')
<matplotlib.text.Text at 0x7fde65997110>
```

Out [22]:

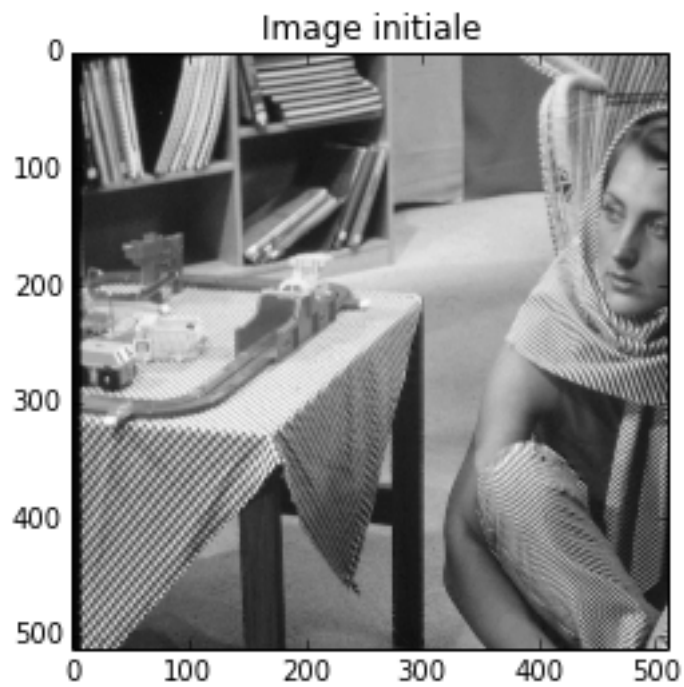


Image filtrée par un gradient de Sobel suivant x

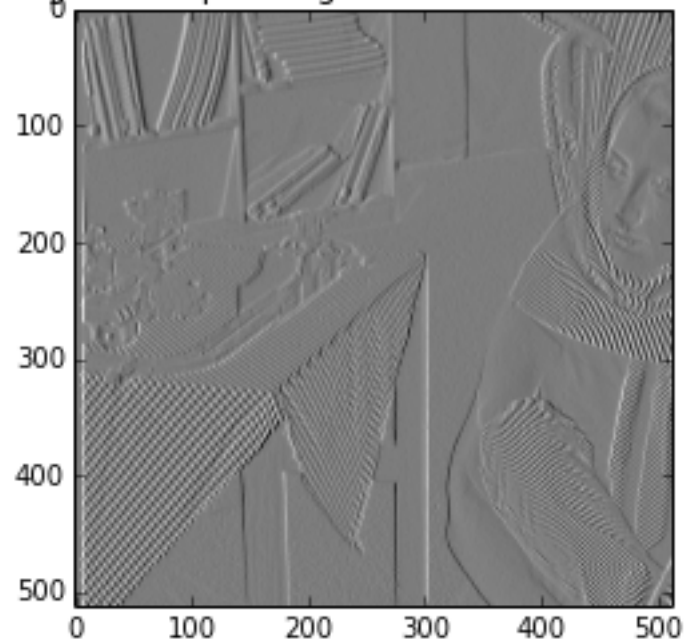
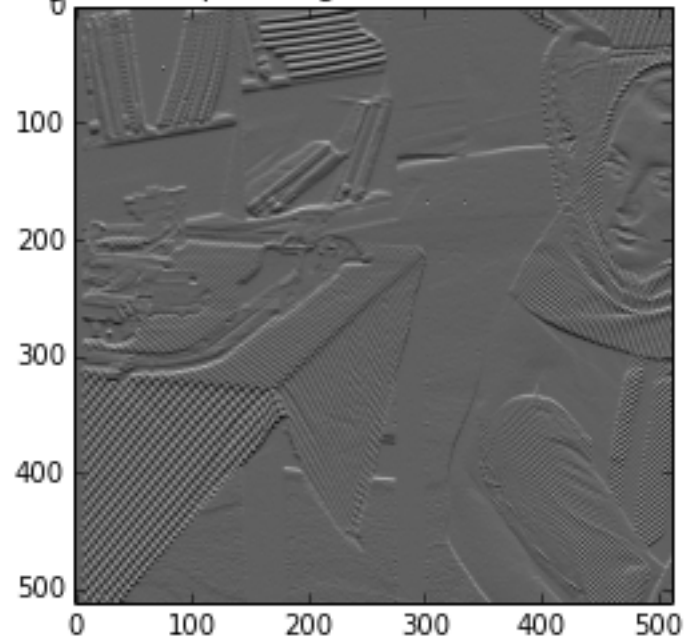
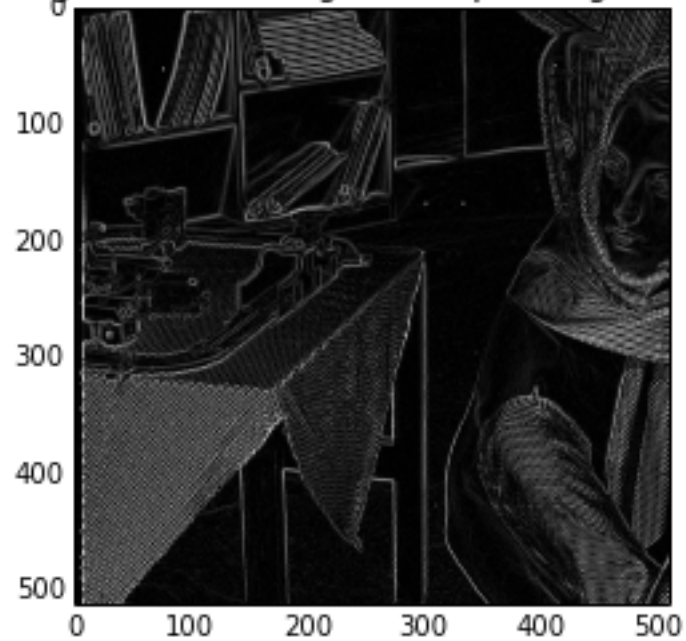


Image filtrée par un gradient de Sobel suivant y



Module du gradient de l'image filtrée par un gradient de Sobel



In []: