

# Compte redudu du TP 1 réalisé par DAGHMOURA Haithem

## I-Préliminaires

### Section I : Imports

```
In [1]: import numpy as np
import platform
import tempfile
import os
import matplotlib.pyplot as plt
from scipy import ndimage as ndi
# necessite scikit-image
from skimage import io as skio
```

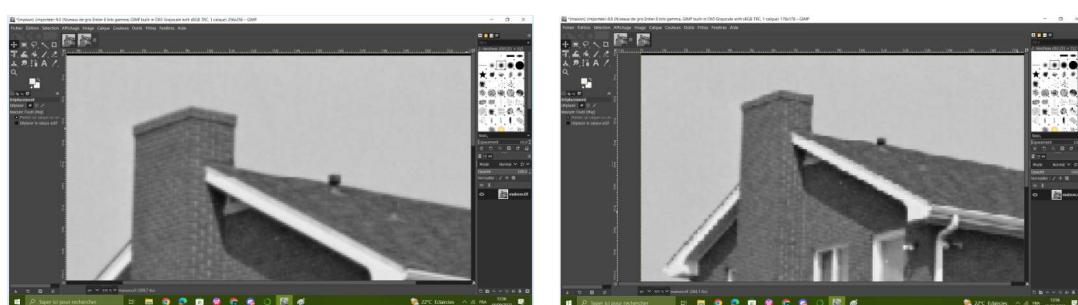
### Section II : Fonctions utiles

```
In [2]: from useful_functions import *
```

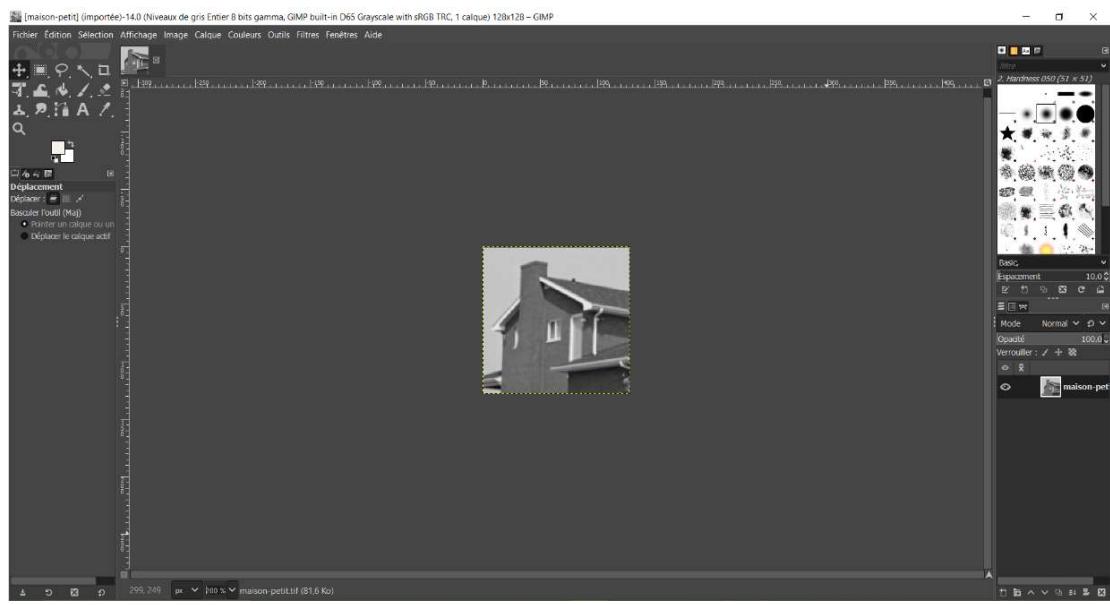
## II- Visulalisation et utilisation de gimp

### 2.1 Zoom

En faisant un zoom, gimp augemente la taille du chaque pixel sans modifier aucune chose sur l'image

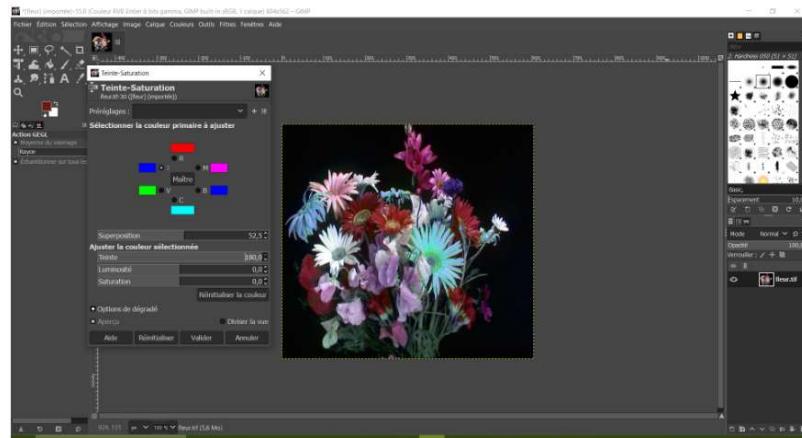


la premiere image est à échelle réel et la deuxieme est après la reduction d'un facteur de 2. En comparant les deux images (avec un zoom de 800%), on remarque que le nombre de pixel a été reduit donc en faisant la reduction d'echelle on a reduit le nombre de pixel dans la deuxieme image. On a donc apparition des raillure explique par le fait du replmenent du spectre du au suréchantillonage



Pour cette image on un un nombre plus petit de pixel (du au fait que la resolution est plus petite 128x128) ce qui explique que sa taille lorsqu'elle apparaît est plus petite

## 2.2 Espace couleurs



Les deux positions extrêmes de ce boutons font correspond à la même transformation, en fait la teinte correspond à un angle et en trigonométrie  $-180^\circ$  est congrue à  $180^\circ$  modulo  $360^\circ$  c'est le même angle !!



image pour saturation 100%

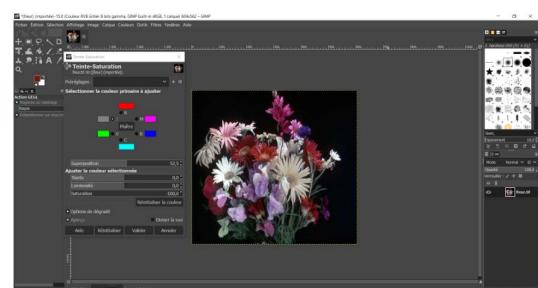


image pour saturation -100%

# III-Niveau de gris, histogrammes et statistiques

## 3.1 Histogramme

```
In [3]: im=skio.imread('images/maison.tif')
```

```
In [4]: type(im)
```

```
Out[4]: numpy.ndarray
```

```
In [5]: im.shape
```

```
Out[5]: (256, 256)
```

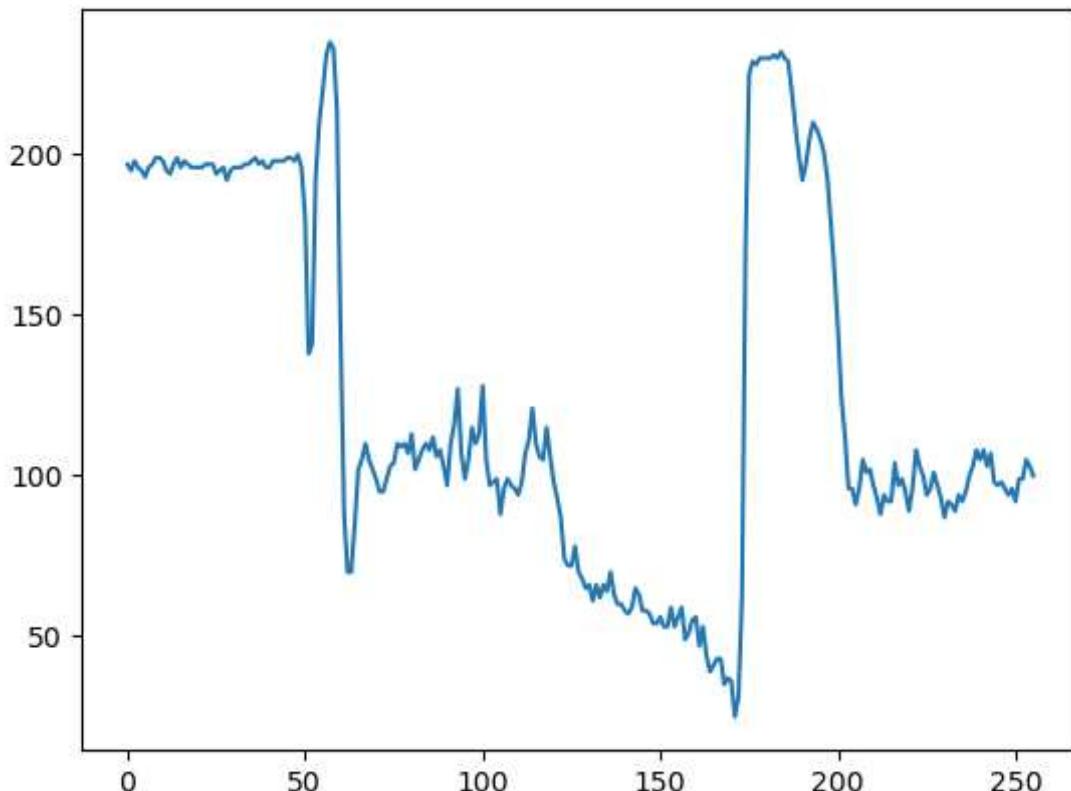
```
In [6]: im[100,100]
```

```
Out[6]: 128
```

```
In [7]: viewimage(im)
```

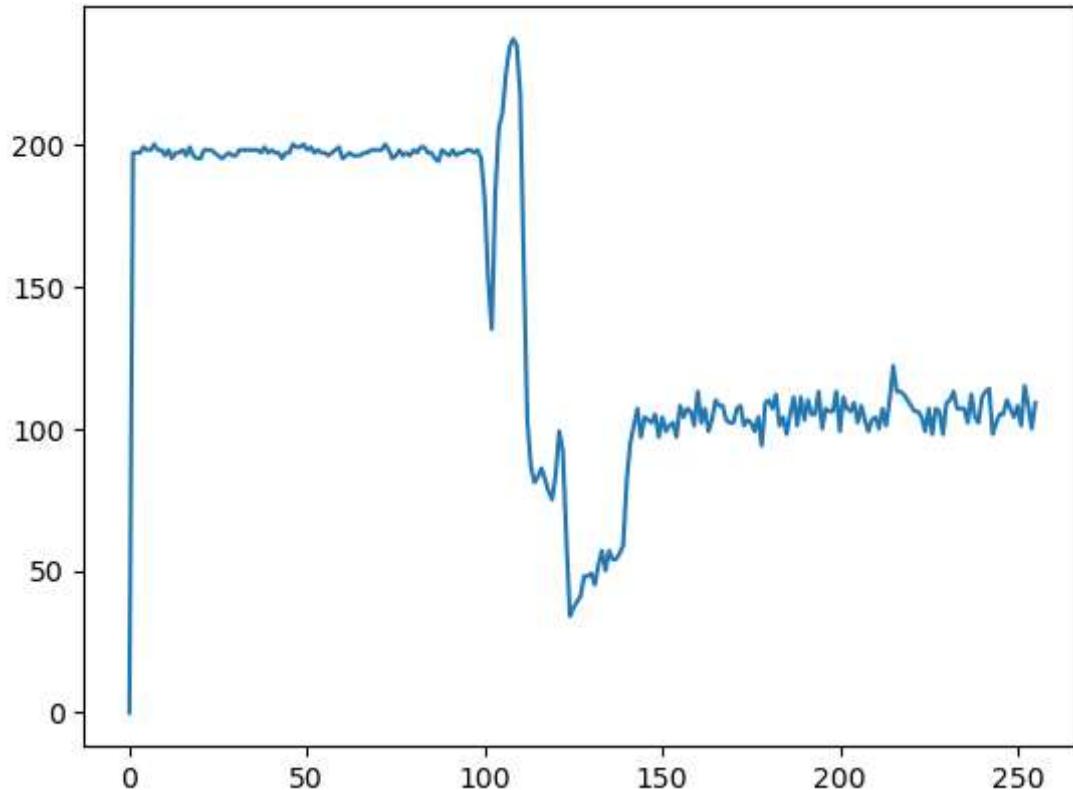
```
In [8]: plt.plot(im[100,:])
```

```
Out[8]: [
```



```
In [9]: plt.plot(im[:,50])
```

```
Out[9]: [
```



```
In [10]: im.max()
```

```
Out[10]: 246
```

```
In [11]: im.min()
```

```
Out[11]: 0
```

```
In [12]: imfloat=np.float32(im)
```

```
In [13]: imfloat[50,50]
```

```
Out[13]: 198.0
```

```
In [14]: abs(im)
```

```
Out[14]: array([[ 0,  0,  0, ...,  0,  0,  0],
       [198, 197, 197, ..., 198, 199, 199],
       [197, 198, 195, ..., 200, 200, 199],
       ...,
       [192, 194, 194, ..., 54, 72, 167],
       [192, 194, 194, ..., 62, 76, 165],
       [191, 193, 195, ..., 74, 79, 164]], dtype=uint8)
```

```
In [15]: im_c=skio.imread('images/fleur.tif')
```

```
In [16]: viewimage(im_c)
```

```
In [17]: viewimage_color(im_c)
```

```
In [18]: viewimage(im_c[:, :, 0])
```

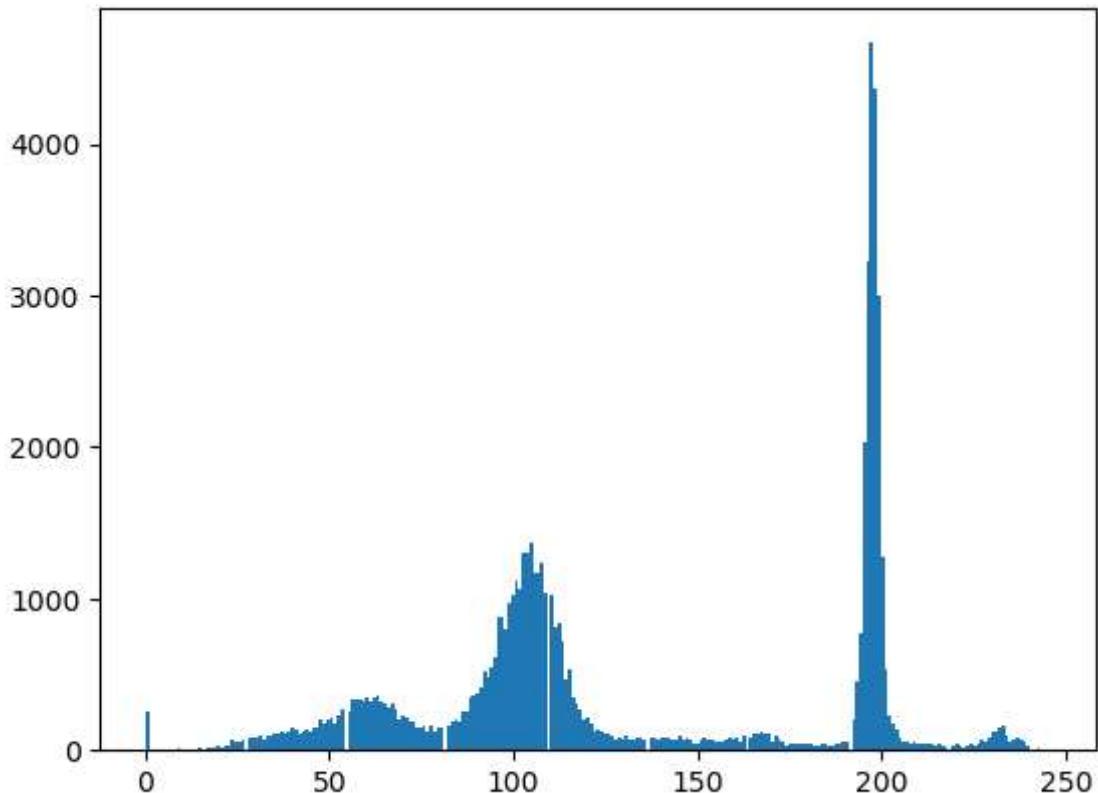
```
In [19]: viewimage(im_c.mean(axis=2)) #La moyenne des trois canaux
```

```
In [20]: red=np.zeros_like(im_c)
red[:, :, 0]=im_c[:, :, 0]
```

```
In [21]: viewimage(red)
```

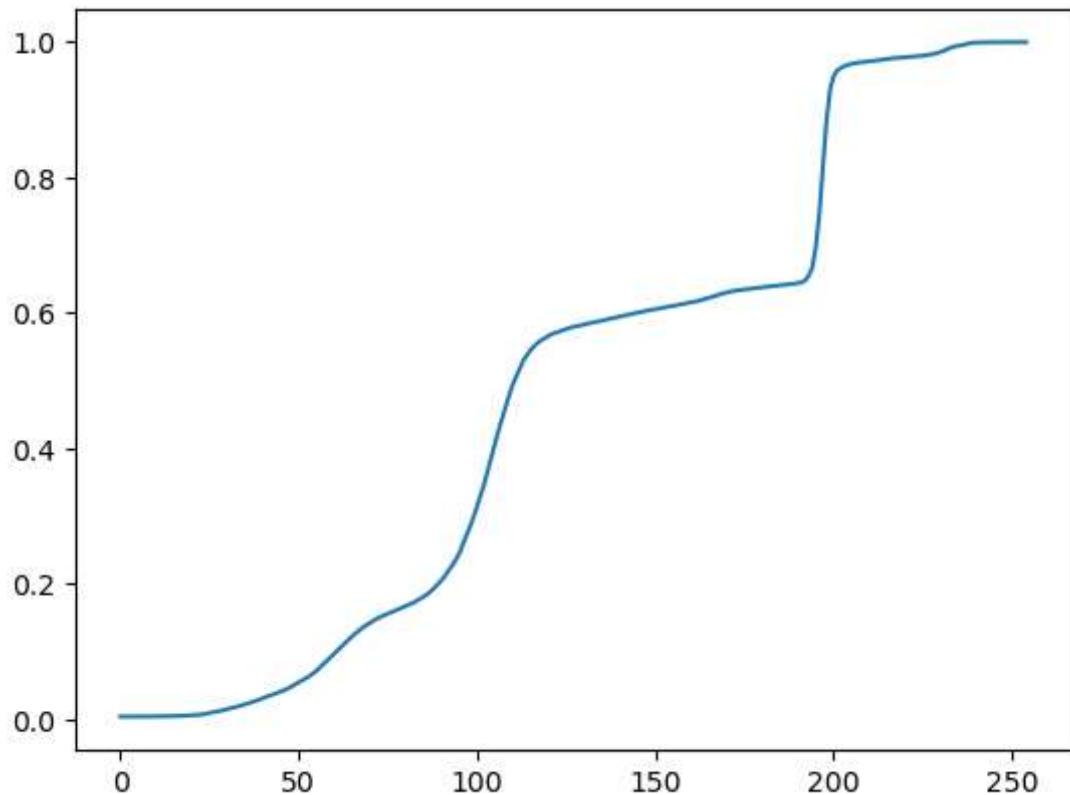
```
In [22]: viewimage(im)
```

```
In [43]: plt.hist(im.reshape((-1,)),255);
```



```
In [24]: (histo,bins)=np.histogram(im.reshape((-1,)),np.arange(0,256))
histo=histo/histo.sum()
histocum=histo.cumsum()
plt.plot(histocum)
```

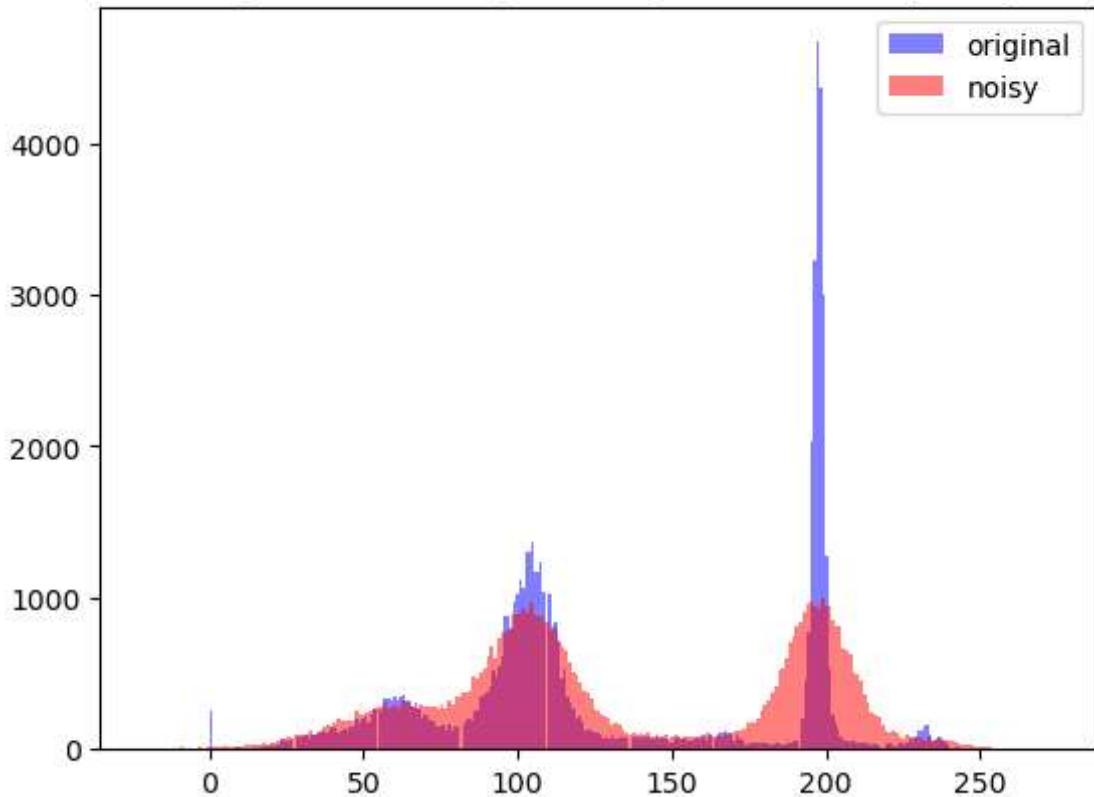
```
Out[24]: [<matplotlib.lines.Line2D at 0x1719ec46c10>]
```



```
In [57]: imbr=noise(im,10)  
viewimage(imbr)
```

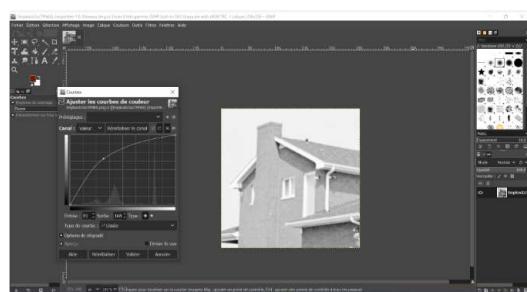
```
In [46]: plt.hist(im.reshape((-1,)),255, color='blue', alpha=0.5, label='original')  
plt.hist(imbr.reshape((-1,)),255, color='red', alpha=0.5, label='noisy')  
plt.legend()  
plt.title("histograms of the orginal image and the noisy image")  
plt.show()
```

histograms of the orginal image and the noisy image

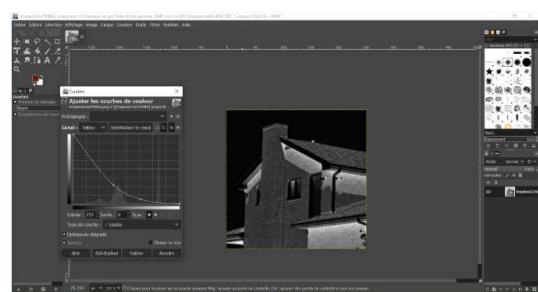


-l'effet d'ajout du bruit gaussien peut etre vu comme un filtre moyenneur gaussien (les pics sont plus aplatis en particulier au niveau gris 200 qui compté plus que 4000 de pixel dans la premiere image et a pres ajout du bruit ce pic devient plus aplati) -en considérant les niveaux de gris d'une image comme la réalisation d'une variable aléatoire dont la loi est l'histogramme de l'image, l'ajout de bruit donne un histogramme convolué avec une gaussienne centrée d'ecart-type br.

## 3.2 Changement de contraste

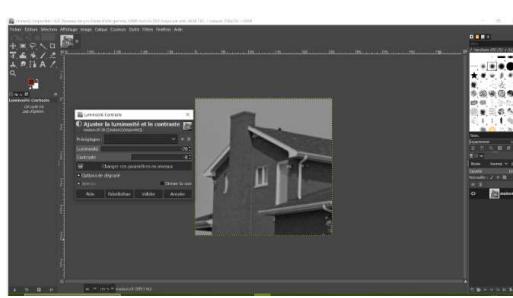


fonction croissante pour le changement de contraste

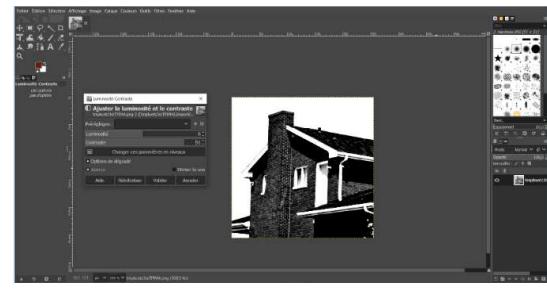


fonction décroissante pour le changement de contraste

On remarque que le changement de contraste par une fonction croissante ne modifie pas notre perception de l'image et on reste capable à la comprendre très rapidement, par contre lorsqu'on applique une fonction qcq (décroissante en particulier) notre perception de l'image devient plus difficile et on arrive pas toujour à la comprendre



ajout d'une constante négative



multiplication par une constante après  
centrage

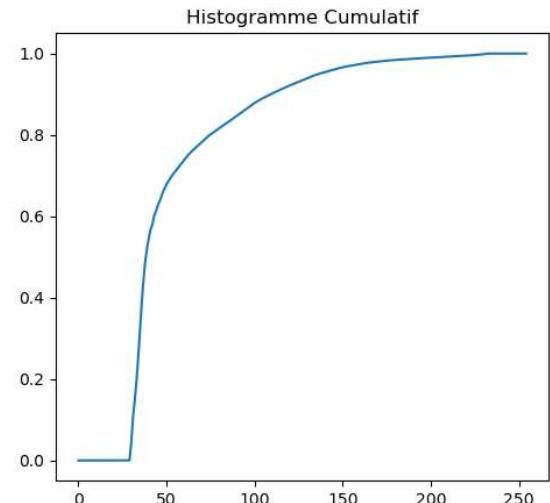
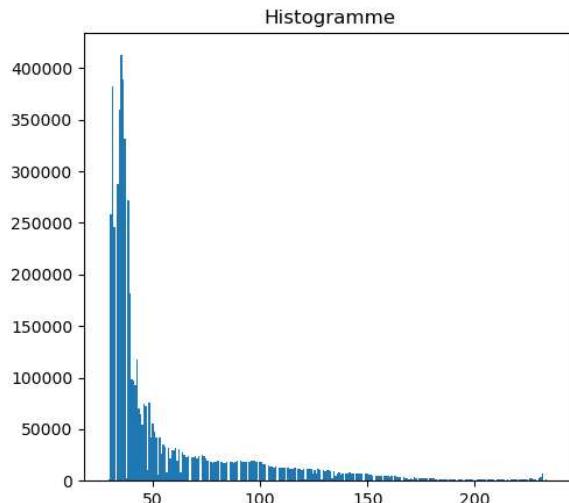
### 3.3 Egalisation d'histogramme

```
In [52]: im=skio.imread('images/sombre.jpg')
im=im.mean(axis=2)
viewimage(im)

fig, axes = plt.subplots(1, 2, figsize=(12, 5))
axes[0].hist(im.reshape((-1,)), 255)
axes[0].set_title('Histogramme')

(histo, bins) = np.histogram(im.reshape((-1,)), np.arange(0, 256))
histo = histo / histo.sum()
histocum = histo.cumsum()
axes[1].plot(histocum)
axes[1].set_title('Histogramme Cumulatif')

plt.show()
```



```
In [53]: imequal=histocum[np.uint8(im)]
```

```
In [30]: viewimage(imequal)
```

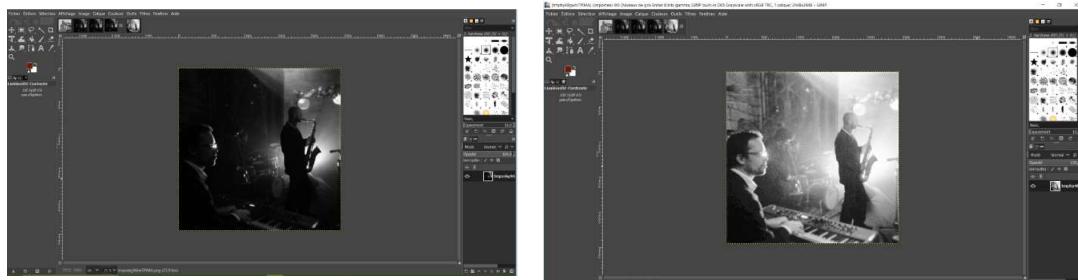


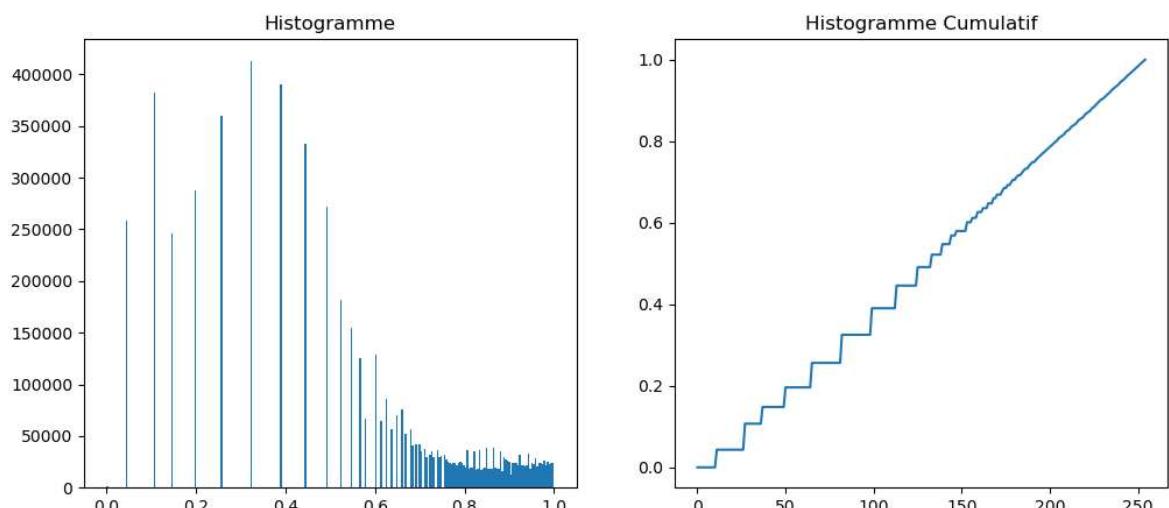
image originelle

image après égalisation d'histogramme

```
In [54]: fig,axes = plt.subplots(1, 2, figsize=(12, 5))

axes[0].hist(imequal.reshape((-1,)),255);
axes[0].set_title("Histogramme")
(histo,bins)=np.histogram(imequal.reshape((-1,)),255) # 255 valeurs réelles qui
histo=histo/histo.sum()
histocum=histo.cumsum()
axes[1].plot(histocum)
axes[1].set_title("Histogramme Cumulatif")
```

Out[54]: Text(0.5, 1.0, 'Histogramme Cumulatif')



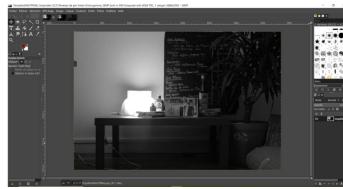
On observe que dans le histogramme les valeurs au début sont espacé. cela est du au quantification des valeurs prises par les pixels. cet effet est claire dans l'histogramme cummulé plus particulièrement au début entre 0 et 100 et donc on a une agmentation dans l'histogramme cummulé seulement au point qui correspondent a des valeurs prises par les pixels.

On considerant l'histogramme comme la fonction de densité de densité alors l'histogramme cumulé est la fonction de répartition F de la variable aléatoire et donc son image par F suit la loi uniforme

### 3.4 Prescription d'histogramme

```
In [3]: u = skio.imread('images/vue1.tif')
v = skio.imread('images/vue2.tif')
viewimage(u)
```

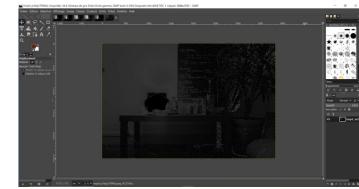
```
viewimage(v)
viewimage(np.abs(u-v))
```



vue 1



vue 2



vue 1 - vue 2

la difference entre les deux images est peu laisible : on ne parvient pas à extraire de l'information à l'oeil nu en regardant l'image de la differnce

```
In [63]: ind=np.unravel_index(np.argsort(u, axis=None), u.shape) #unravel donne les index
unew=np.zeros(u.shape,u.dtype)
unew[ind]=np.sort(v, axis=None)
viewimage(unew) #u avec l'histogramme de v
```



vue 1



vue 1 apres transformation

En appliquant cette transformation on remarque qu'on est capable de voir quelque nouvelle détail de la lampe.



vue 2



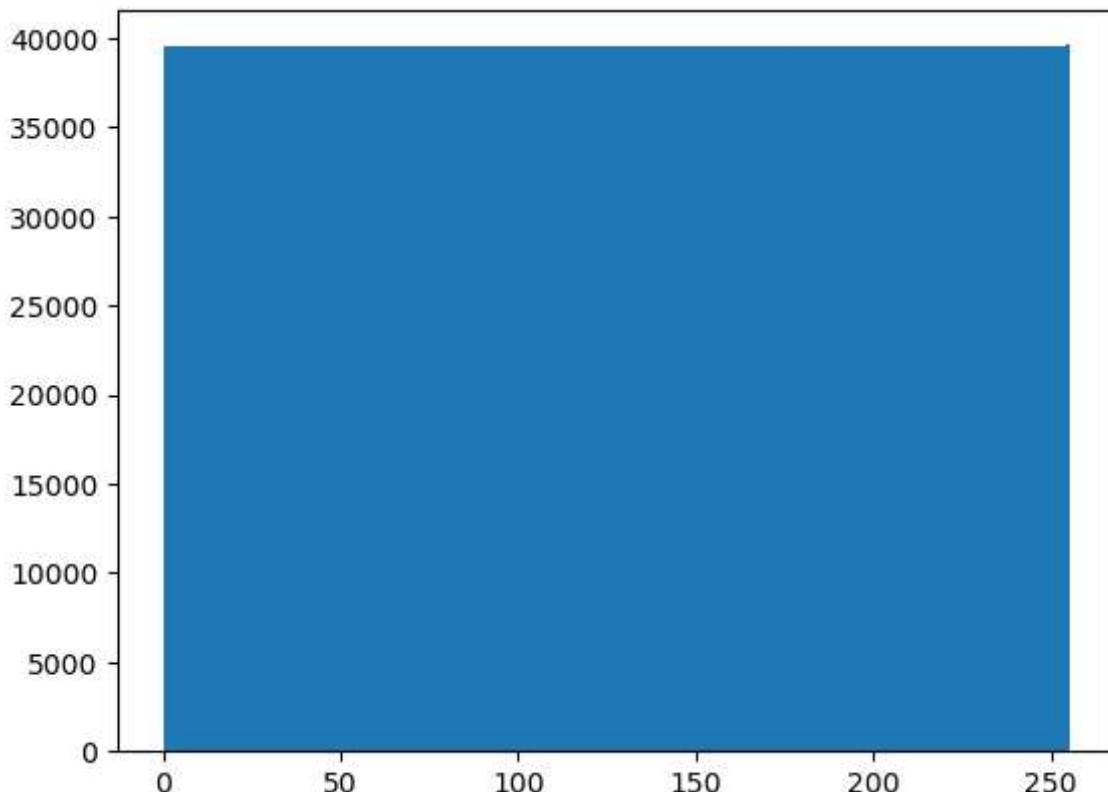
vue 1 apres transformation

Après avoir donné l'histogramme de la vue 2 à la vue 1 la comparaison des deux images est plus facile à l'oeil nu et on peut distinguer plusieurs differences

Pour répondre à la deuxième question sur un moyen plus simple d'obtenir le même histogramme pour les deux images, on peut directement égaliser les histogrammes des images. Cela permet d'harmoniser les niveaux de gris des images en ajustant leur histogramme, ce qui peut être plus simple que de donner à l'une l'histogramme de l'autre de manière manuelle.

pour égaliser l'histogramme d'une manière plus simple on peut le donner l'histogramme d'une image ayant un histogramme constant donc on génère un ndarray contenant toutes les valeurs à proportion égales et on refait la dernière étape au dessus. Le code permettant d'obtenir un ndarray ayant un histogramme constant et le même nombre total de pixels est le suivant :

```
In [18]: u_size=u.size  
u1=u.reshape((-1))  
occurrences =u_size//255  
v=np.zeros_like(u1)  
  
for i in range (u_size):  
    v[i]=i//occurrences  
plt.hist(v,255);
```



### 3.5 Dithering

```
In [6]: im = skio.imread("images\maison.tif")  
im1=quantize(im,25)  
im2=quantize(im,10)  
im3=quantize(im,2)  
viewimage(im)  
viewimage(im1)
```

```
viewimage(im2)  
viewimage(im3)
```

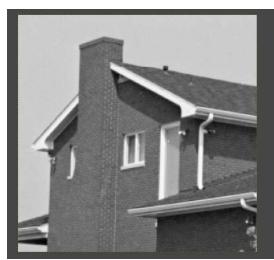


image originale



image avec 25  
niveau



image avec 10  
niveau

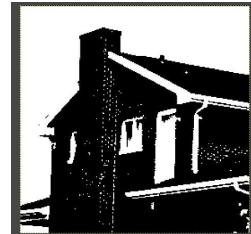


image avec 10  
niveau

```
In [12]: ims=seuil(im,90)  
viewimage(ims)
```

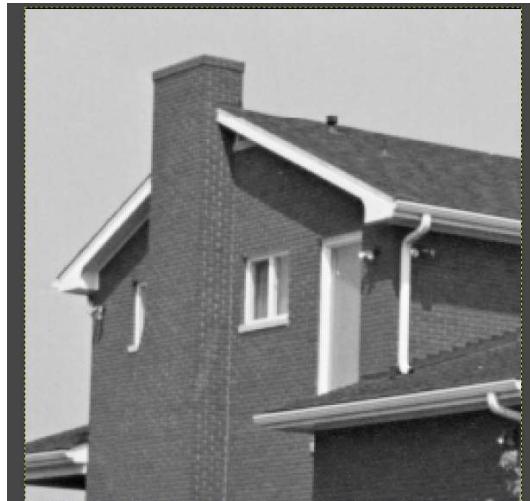


image originale



image avec seuil =90

```
In [13]: imbr=noise(im,20)  
viewimage(imbr)  
imbrs=seuil(imbr,90)  
viewimage(imbrs)
```

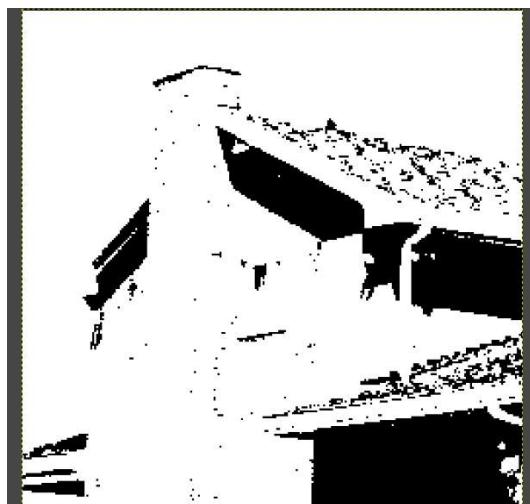


image originale avec seuil



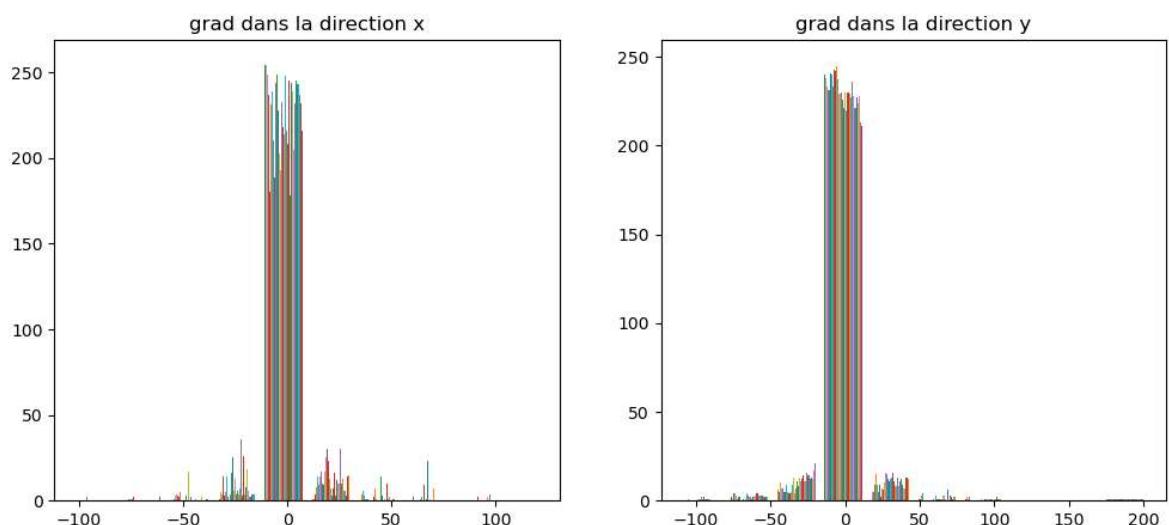
image bruité avec seuil =90

On constate qu'après avoir appliquer le bruit on parvient à retrouver plus facilement la forme de la maison et on peut capturer plus l'allure globale de l'image en utilisant 2 couleurs uniquement.

en ajoutant du bruit à l'images, on remarque que les points qui étais proches de seuil ont pu franchir cette barrière avec le bruit et vice versa, tandis que les points qui étaient loin ont une faible probabilité de changer de valeur après seuillage. Par conséquent plus une zone présente de la lumière plus il sera probable que cette zone dépasse le seuil après ajout de bruit plus de point on aura ce qui explique le résultat obtenu

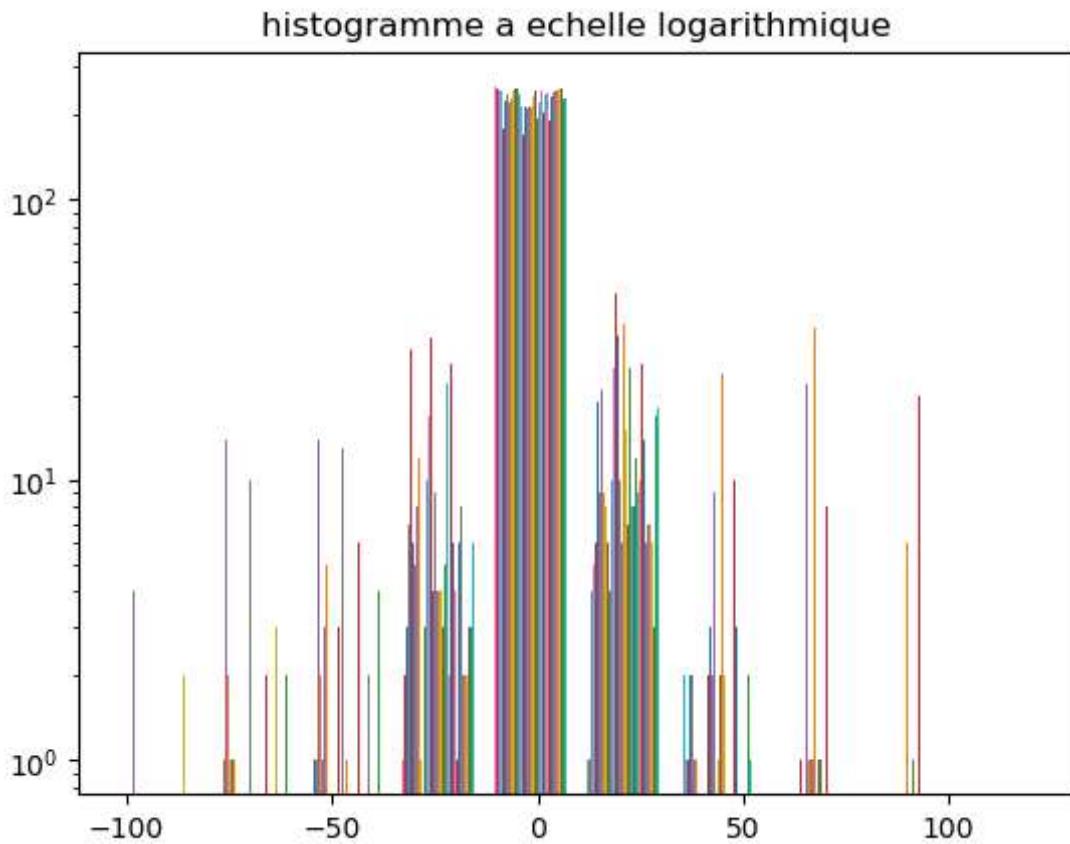
### 3.6 Différences de niveaux de gris voisins

```
In [21]: fig,axes = plt.subplots(1, 2, figsize=(12, 5))
axes[0].hist(gradx(im));
axes[0].set_title("grad dans la direction x")
axes[1].hist(grady(im));
axes[1].set_title("grad dans la direction y");
```



```
In [27]: plt.hist(gradx(im))
plt.yscale("log")
plt.title("histogramme à échelle logarithmique")
plt.plot()
```

```
Out[27]: []
```



oui, la distribution semble qu'elle parvienne d'une distribution gaussien. En fait sauf au boards d'un objet, les pixels voisins ont à peu près la memes valeur, ce qui explique le pics au voisinage de 0. En faisant la difference entre deux point éloigné on peut trouver des autre graphes mais il est fort probable que ces deux points sont incorréle l'un de l'autre et donc on aura un histogramme uniforme.

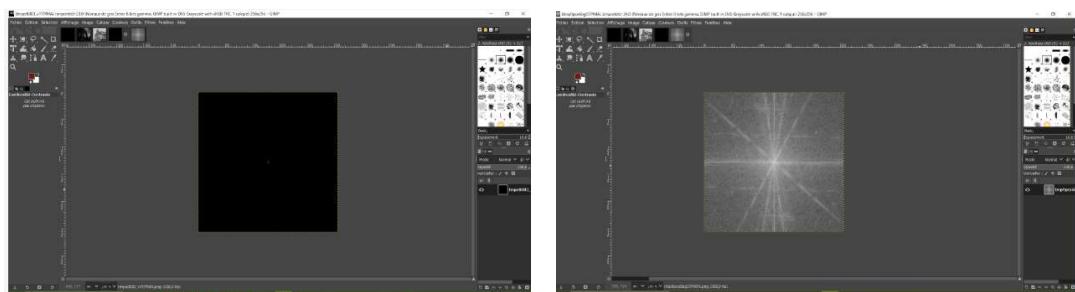
## 4 Spectre des images et transformation de Fourier

### 4.1 Visualisation de spectres

```
In [28]: im=skio.imread('images/maison.tif')
```

```
In [29]: viewimage(im)
```

```
In [34]: view_spectre(im,option=2)
```

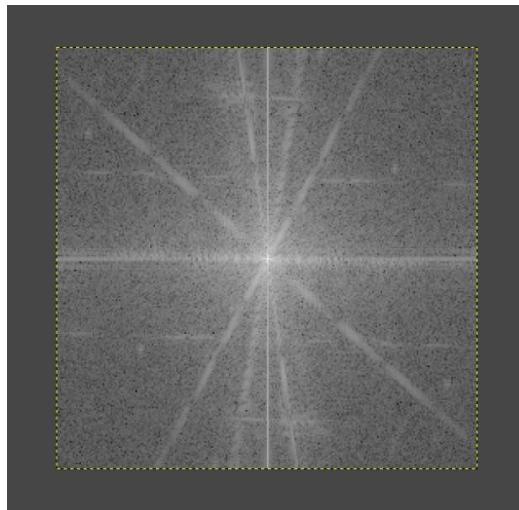


option 1

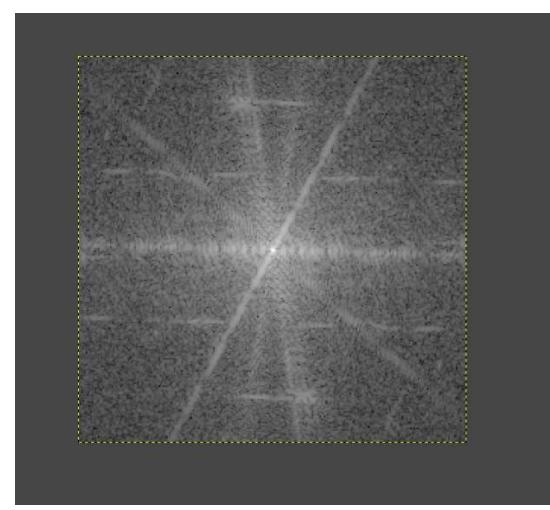
option 2

on constate que lorsqu'on prend l'echelle lineare le sepctre de l'image de la maison est presque invisible il est represente par un point blanc dans le milieu. ceci est du au fait que le point centrale contient la somme de toutes les valeurs ce qui donne une valeurs tres grandes par rapport au autres donc c'est la seule valeurs qui apparait pour l echelle logarithmique le spectre est plus claire.

```
In [33]: view_spectre(im,2,True)
```

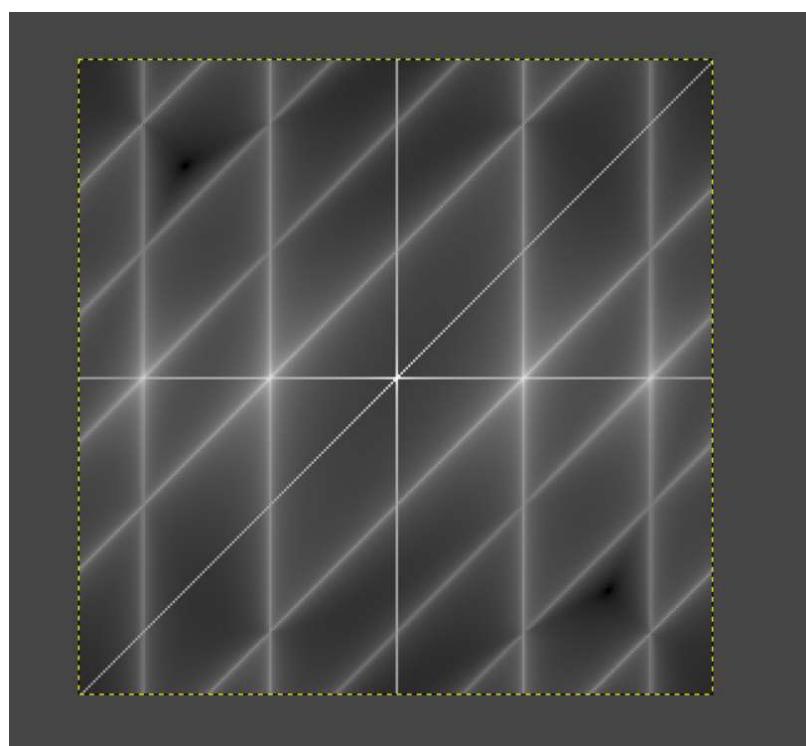


specbre avec option 2



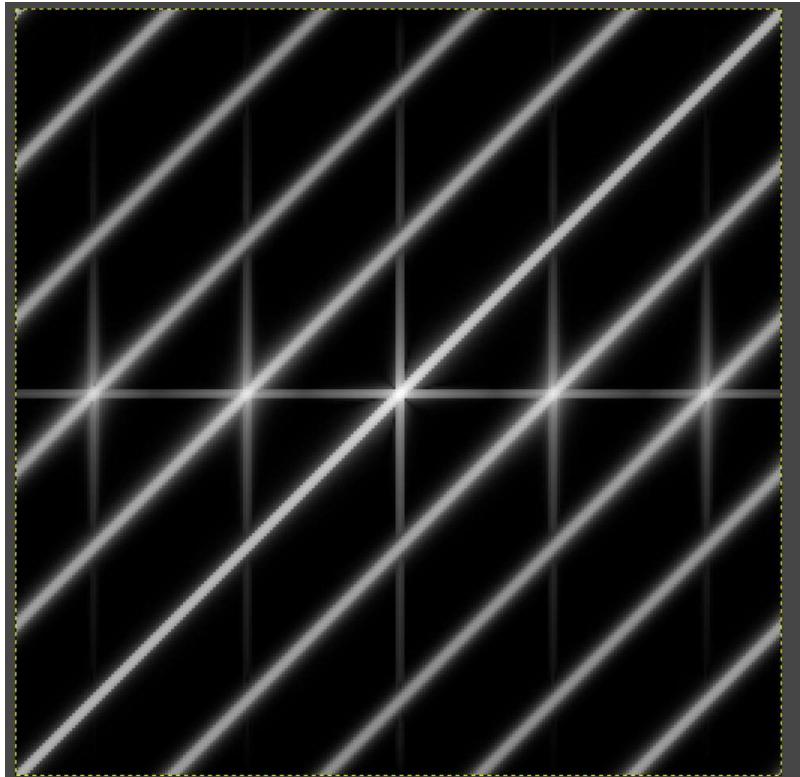
specbre avec option 2 et fenetre de hamming

```
In [3]: im1=skio.imread("images/rayures.tif")
view_spectre(im1,2)
```



specbre

```
In [5]: view_spectre(im1,2,True)
```



spectre

On constate que dans le spectre on a des lignes de direction orthogonal à celle des rayures, ce ceci est du au convolution entre le spectre de la même image mais sans rayures (image moitié blanc moitié noire) dont le spectre est une ligne orthogonale à celle de la transition du couleur et l'image qui contient just les rayures dont le spectre sera des point suivant la direction orthogonale des rayures. En faire l'image de rayures peut être vu comme la somme des deux images décrites précédemment d'où la convolution des spectres.

le filtre de hamming est un filtre qui atténue les discontinuité au bord en annulant sa valeur. donc on visualise le spectre de l'image après l'application du filtre de hamming on constate que les lignes horizontaux et verticaux sont énormément atténuer car on n'a plus de discontinuité au bord.

```
In [10]: im2=skio.imread("images/carte_nb.tif")
viewimage(im2)
# échantillonage en prenant un pixel sur deux dans les deux directions
im_ech=np.zeros((128,128))
im_ech=im2[::2,::2]
viewimage(im_ech)

view_spectre(im2,2)
view_spectre(im_ech,2)
```

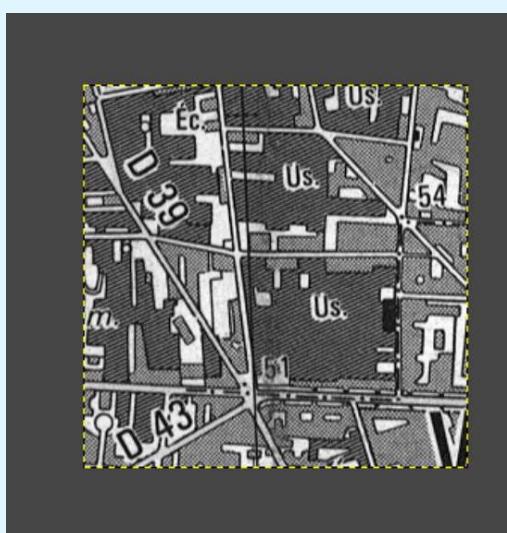


image originale

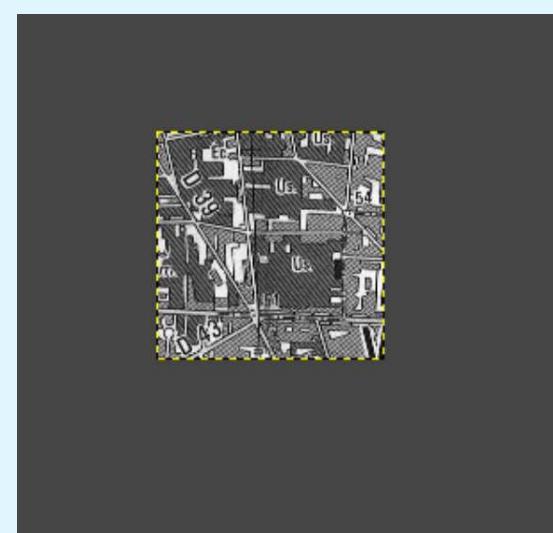
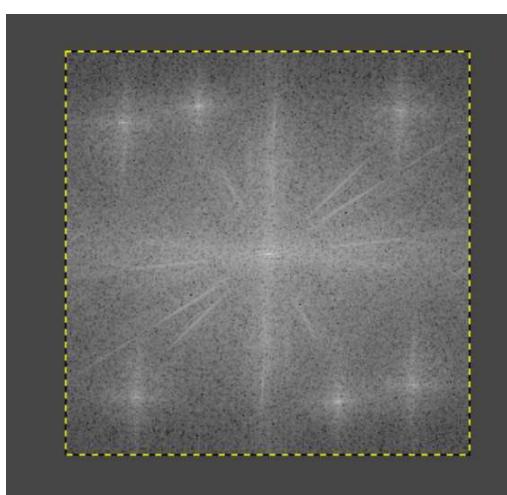
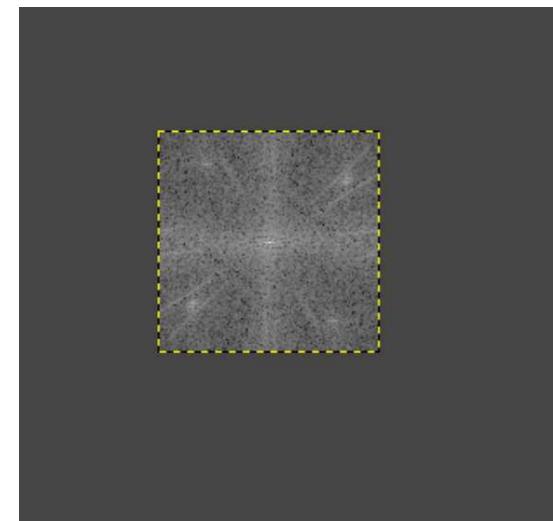


image échantillonnée avec un facteur 1/2



spectre de l'image originale



Spectre de l'image échantillonnée

Le spectre a une taille plus réduite que celui de l'image originale et présente l'effet de repliement et on a perdu de l'information

## 4.2 Ringing

```
In [4]: im2=skio.imread("images/carte_nb.tif")
viewimage(filterlow(im2))
# pour visualiser le filtre j'ai utilisé le code suivant et modifié le code de f
#im2=skio.imread("images/carte_nb.tif")
# mask=filterLow(im2)
# mask1=mask[:128,:,:128]
# mask2=mask[:128:,128::]
# mask3=mask[128:,:,128::]
# mask4=mask[128:,:,128:]
# mask5=np.copy(mask)
# mask5[:128,:,:128]=mask3
# mask5[:128:,128::]=mask4
# mask5[128:,:,128::]=mask1
# mask5[128:,:,128::]=mask2
```

```
# viewimage(mask5)
#
```

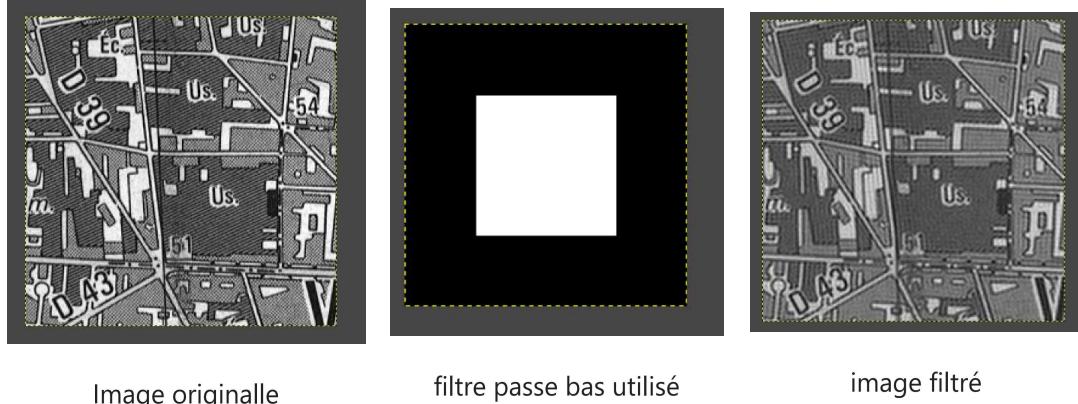


Image originale

filtre passe bas utilisé

image filtré

On remarque que on a perdu de la luminsité de l'image. Ceci est prévisible car on a annuler certaine region de domaine frequentielles donc on a diminué l'energie de l'image. Cependant l'image reste de bonne qualité et on peut reconnaître plus aspect

```
In [5]: viewimage(filterlow(im2))
```

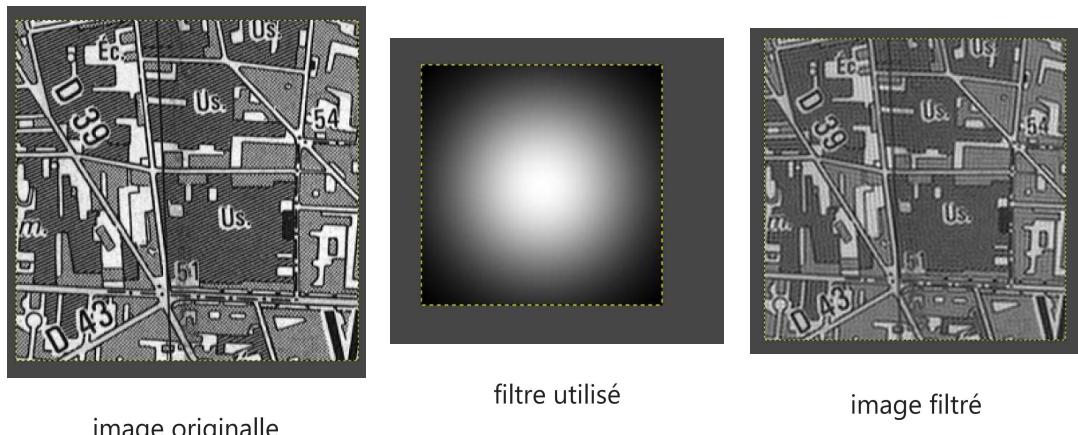


image originale

filtre utilisé

image filtré

en visualisant les deux filtre on remarque que le filtre passe bas parfait présente une grande discontinuité dans le domaine spectrale celui ci va induire une convolution avec une fonction à support infini dans le domaine spatial et puisque la transformé inverse de la fonction carré est un sinus cardinale alors on aura des rayures supplémentaires => c'est le phénomène de ringing. tandis que pour le filtre gaussien on a pas cette discontinuité ce qui explique que ce phénomène est d'autant plus atténué dans la dernière image