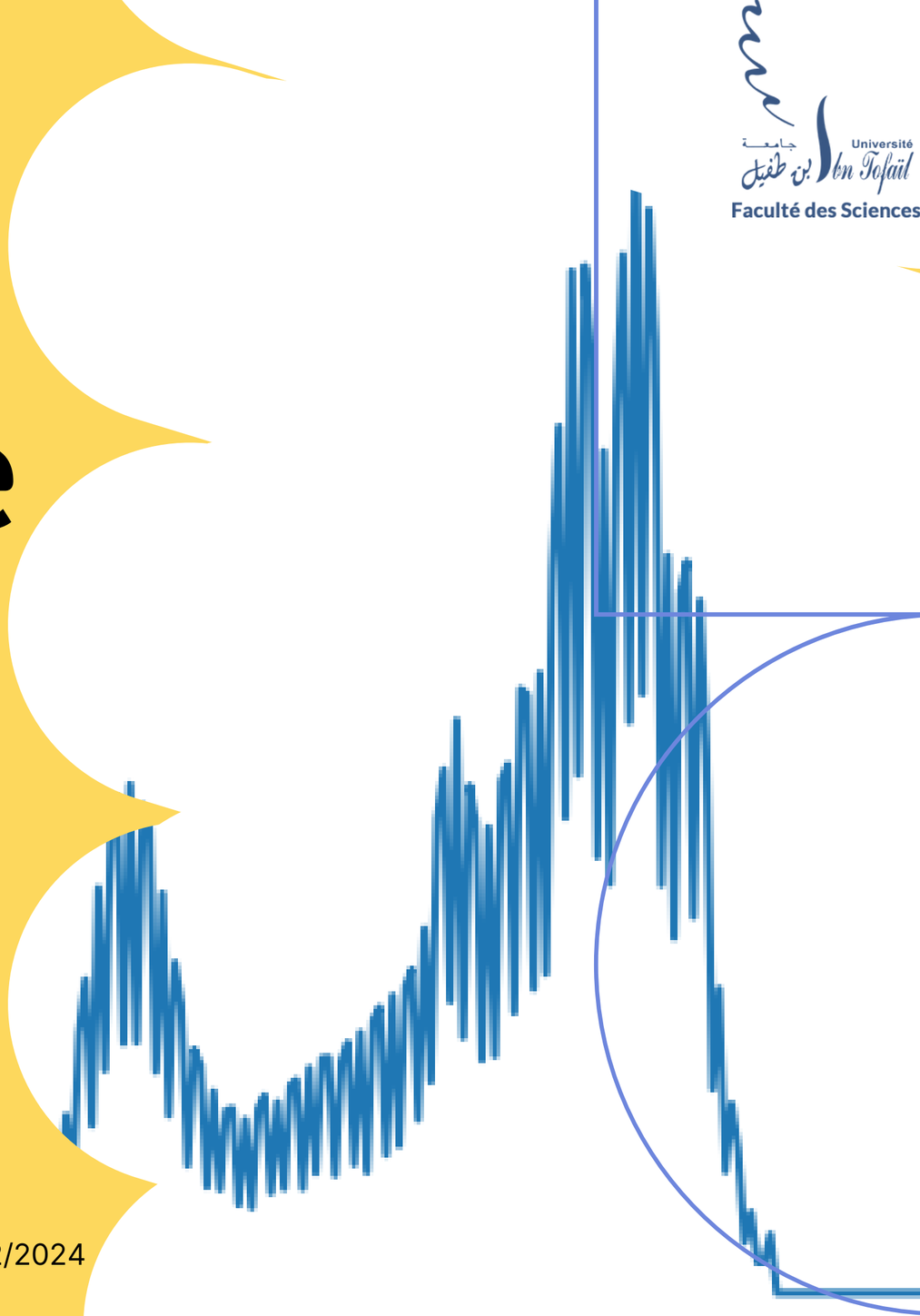
Traitement du signal et de l'image

L3 Intelligence Artificielle & Ingénierie des données



PR. ANASS NOURI
PHD STUDENT. ZAINEB IBORK

17/02/2024

TPs Order

TP1

Histogram construction/manipulation

TP2

Image Processing and Analysis

TP3

Fourier transform

TP1

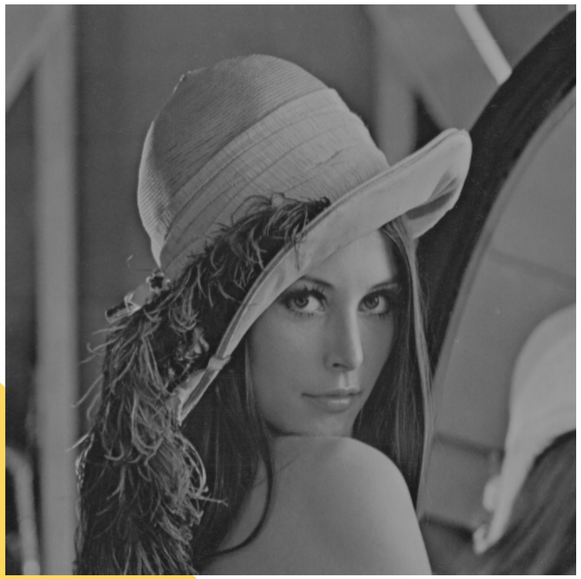
Histogram construction/manipulation

- For luminance correction
- For contrast correction
- For colored contrast correction

1. Luminance correction

Using OpenCV and Numpy libraries, load the dark_lena.png grayscale image from the ressources provided in this practical work.

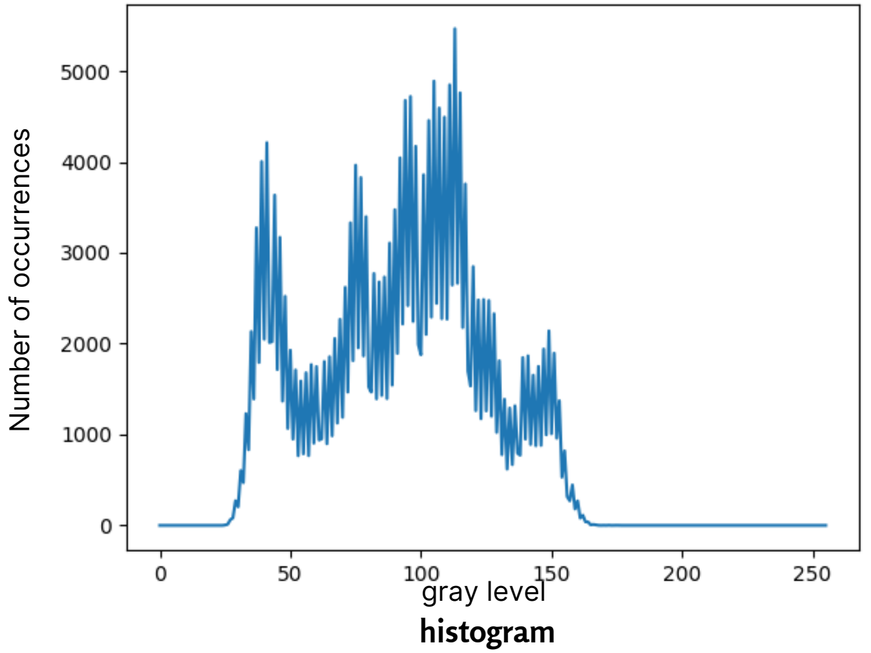
```
import matplotlib.pyplot as plt
import numpy as np
import cv2
image = cv2.imread("dark_lena.png",0)
cv2.imshow('image_orig', image)
```



image

Compute the histogram of the image and visualize it using matplolib.pyplot library.

```
# Calcule l'histogramme de l'image
hist = np.zeros(256, int)       # prépare un vecteur de 256 zéros (pour chaque gris)
for i in range(0,image.shape[0]):      # énumère les lignes
    for j in range(0,image.shape[1]):  # énumère les colonnes
        hist[image[i,j]] = hist[image[i,j]] + 1

#print(hist)
plt.plot(hist)
plt.show()
```



histogram

TP1

1. Luminance correction

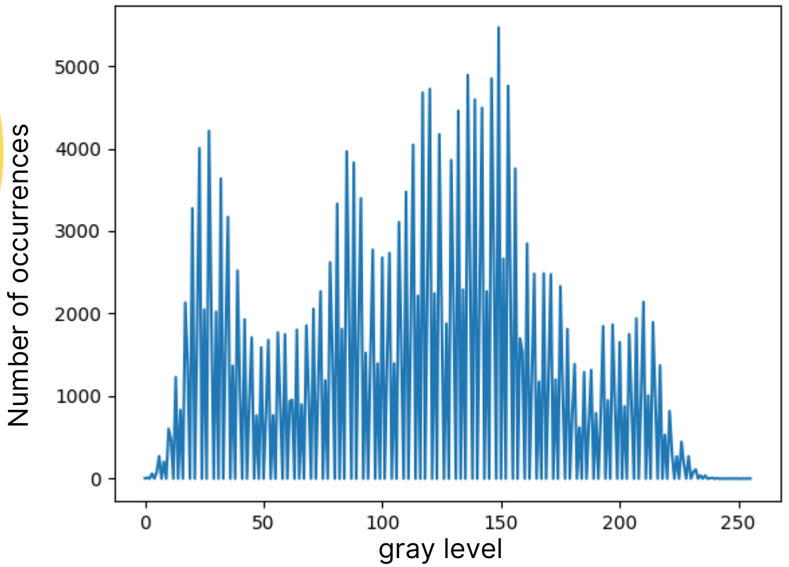
In order to correct the luminance of the image what can you do to the image to stretch the image ? Code it and visualize the resulted image.

```
#histogram stretching
max_gray_image = image.max()
min_gray_image = image.min()
max_gray = 255
min_gray = 0

res_image = (image - min_gray_image)/
          (max_gray_image - min_gray_image)*(max_gray - min_gray) + min_gray
res_image = np.uint8(res_image)

cv2.imshow('image_', res_image)
```

image

Compute the histogram of the resulted image. What do you remark

```
hist_stretch = np.zeros(256, int)       # prépare un vecteur de 256 zéros
for i in range(0,image.shape[0]):      # énumère les lignes
    for j in range(0,image.shape[1]):  # énumère les colonnes
        hist_stretch[res_image[i,j]] = hist_stretch[res_image[i,j]] + 1

plt.plot(hist_stretch)
plt.show()
```



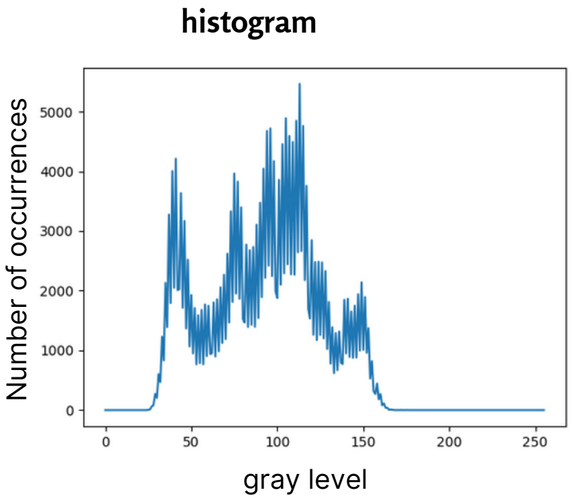
stretched histogram

Interprétation : l'histogramme est étiré et il couvre toute la dynamique des niveaux de gris de l'image. La majorité des niveaux de gris sont présents dans l'image. L'image résultante est plus lumineuse.

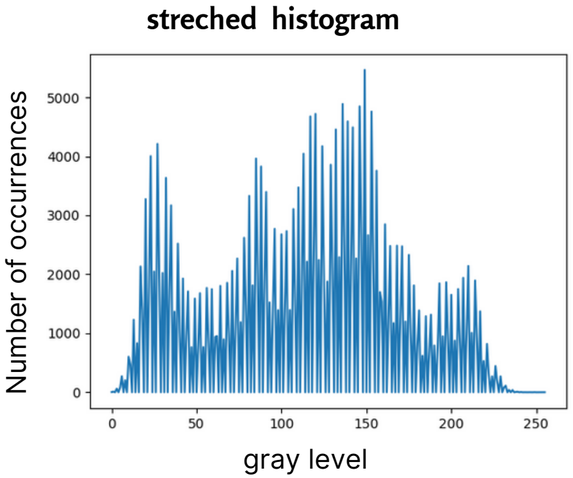
TP1

1. Luminance correction

Original image



Resulted image



TP1

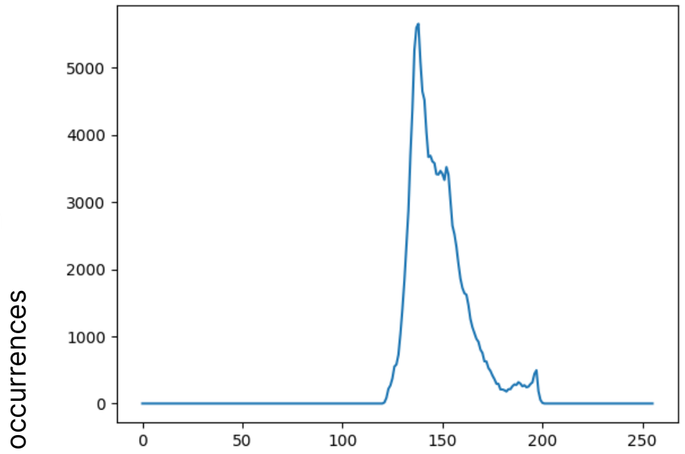
2. Contrast correction

Compute the histogram and cumulative histogram of the image and visualize them using matplolib.pyplot library.

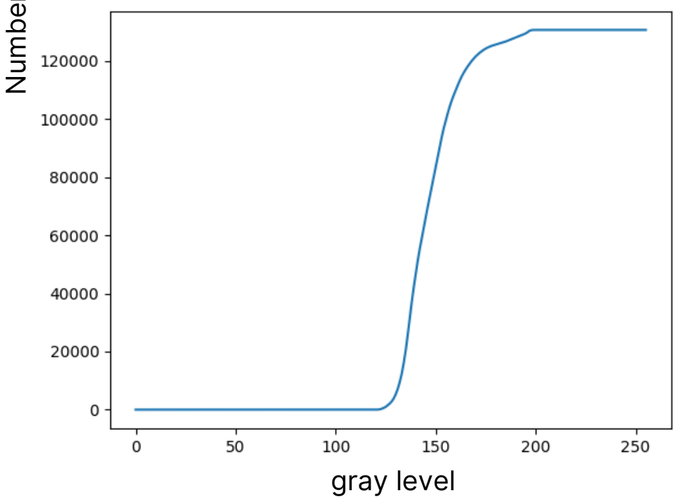
```
image = cv2.imread("weak_contrasted.png",0)

# Calcule l'histogramme de l'image
hist = np.zeros(256, int)       # prépare un vecteur de 256 zéros (pour chaque gris)
for i in range(0,image.shape[0]):      # énumère les lignes
    for j in range(0,image.shape[1]):  # énumère les colonnes
        hist[image[i,j]] = hist[image[i,j]] + 1

cv2.imshow('image_orig', image)

#print(hist)
plt.plot(hist)
plt.show()


# Calcule l'histogramme cumulé hc
hc = np.zeros(256, int)         # prépare un vecteur de 256 zéros
hc[0] = hist[0]
for i in range(1,256):
    hc[i] = hist[i] + hc[i-1]

plt.plot(hc)
plt.show()
```

Image

Histogram



Cumulated histogram



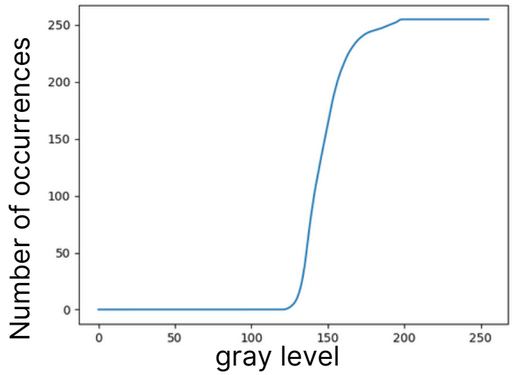
TP1

2. Contrast correction

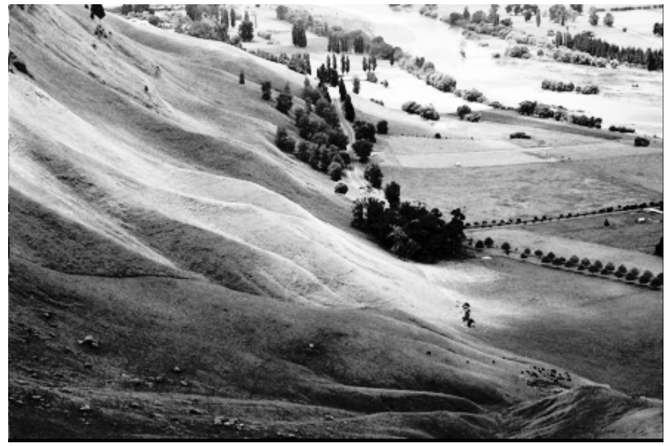
Normalize values of the cumulated histogram in order to bring them in [0 255] :divide each value of the cumulated histogram by the total number of pixels, then multiply the result by 255.

```
# normalize the cumulated histogram
print("normalize the cumulated histogram")
nbpixels = image.size
hc = hc / nbpixels * 255
#print(hc)
plt.plot(hc)
plt.show()

res=image.copy()
# Utilise hc comme table de conversion des niveaux de gris
for i in range(0,image.shape[0]):       # énumère les lignes
    for j in range(0,image.shape[1]):   # énumère les colonnes
        res[i,j] = hc[image[i,j]]
cv2.imshow("Luminance Y après égalisation", res)
```

Normalised cumulated histogram in range [0 , 255]



Original image



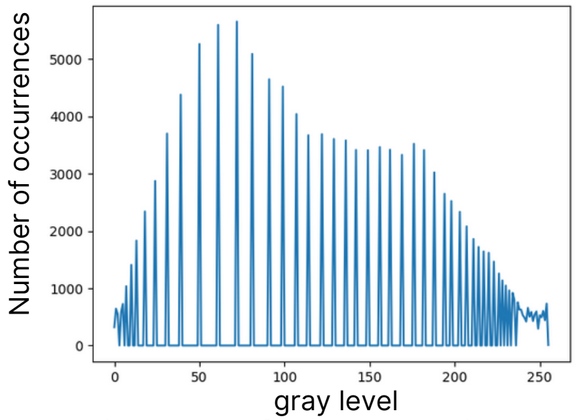
Resulted image after equalization

TP1

2. Contrast correction

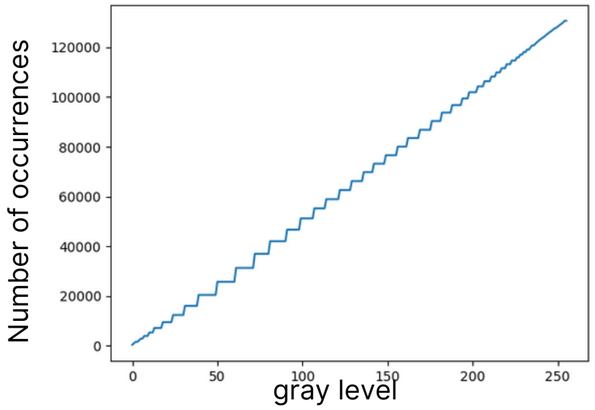
Plot the histogram of the resulted image. What happened to the histogram ? Can you explain more in details what happened when you consider the cumulative histogram ?

```
# Calcule l'histogramme de l'image res
print("Calcule l'histogramme de l'image res: Utilise hc comme table de conversion des niveaux de gris")
hist = np.zeros(256, int)       # prépare un vecteur de 256 zéros (pour chaque gris)
for i in range(0,image.shape[0]):      # énumère les lignes
    for j in range(0,image.shape[1]):  # énumère les colonnes
        hist[res[i,j]] = hist[res[i,j]] + 1

plt.plot(hist)
plt.show()

# Calcule l'histogramme cumulé hc de res
print("Calcule l'histogramme cumulé hc de res")
hc = np.zeros(256, int)         # prépare un vecteur de 256 zéros
hc[0] = hist[0]
for i in range(1,256):
    hc[i] = hist[i] + hc[i-1]

plt.plot(hc)
plt.show()
```



histogram of the resulted image

Interprétation : l'histogramme est mieux homogénéisé. Il n'y a pas de variation brusque entre le nombre d'occurrence de niveaux de gris (ou nombre de pixel).



Cumulated histogram of the resulted image

Interprétation : l'histogramme cumulé de l'image résultat est représenté par une ligne droite croissante à partir des niveaux de gris sombres jusqu'aux niveaux de gris blancs. Ça explique l'amélioration en contraste !

TP1

2. Contrast correction

2.6) If we perform both processes (histogram stretching then histogram equalization), does the resulting image will be better ?

```
image = cv2.imread("dark_lena.png",0)
cv2.imshow('image_orig', image)

print("Calcule l'histogramme de l'image")
hist = np.zeros(256, int)       # prépare un vecteur de 256 zéros (pour chaque
gris)
for i in range(0,image.shape[0]):      # énumère les lignes
    for j in range(0,image.shape[1]):  # énumère les colonnes
        hist[image[i,j]] = hist[image[i,j]] + 1

plt.plot(hist)
plt.show()

print("histogram stretching")
max_gray_image = image.max()
min_gray_image = image.min()
max_gray = 255
min_gray = 0

res_image = (image - min_gray_image)/(max_gray_image - min_gray_image)*(max_gray
- min_gray) + min_gray
res_image = np.uint8(res_image)

cv2.imshow('stretched image', res_image)

hist_stretch = np.zeros(256, int)       # prépare un vecteur de 256 zéros (pour
chaque gris)
for i in range(0,image.shape[0]):      # énumère les lignes
    for j in range(0,image.shape[1]):  # énumère les colonnes
        hist_stretch[res_image[i,j]] = hist_stretch[res_image[i,j]] + 1

plt.plot(hist_stretch)
plt.show()

# Calcule l'histogramme cumulé hc
print("Calcule l'histogramme cumulé hc")
hist= hist_stretch
hc = np.zeros(256, int)         # prépare un vecteur de 256 zéros
hc[0] = hist[0]
for i in range(1,256):
    hc[i] = hist[i] + hc[i-1]

plt.plot(hc)
plt.show()

# normalize the cumulated histogram
print("normalize the cumulated histogram")
nbpixels = res_image.size
hc = hc / nbpixels * 255
#print(hc)
plt.plot(hc)
plt.show()
```

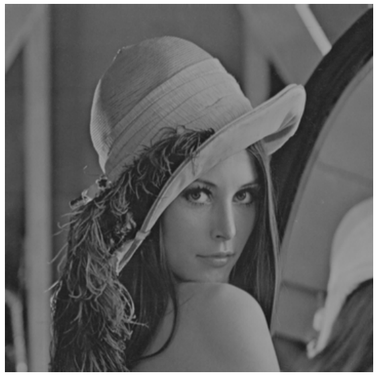
```
res=res_image.copy()

# Utilise hc comme table de conversion des niveaux de gris

for i in range(0,res_image.shape[0]):       # énumère les lignes
    for j in range(0,res_image.shape[1]):   # énumère les colonnes
        res[i,j] = hc[res_image[i,j]]
cv2.imshow("image après égalisation", res)
# Save the resulting image
cv2.imwrite("resulting_image_both.png", res)

#cv2.destroyAllWindows()

# Calcule l'histogramme de l'image res
print("Calcule l'histogramme de l'image res: Utilise hc comme table de conversion
des niveaux de gris")
hist = np.zeros(256, int)       # prépare un vecteur de 256 zéros (pour chaque
gris)
for i in range(0,res_image.shape[0]):      # énumère les lignes
    for j in range(0,res_image.shape[1]):  # énumère les colonnes
        hist[res[i,j]] = hist[res[i,j]] + 1

plt.plot(hist)
plt.show()

# Calcule l'histogramme cumulé hc de res
print("Calcule l'histogramme cumulé hc de res")
hc = np.zeros(256, int)         # prépare un vecteur de 256 zéros
hc[0] = hist[0]
for i in range(1,256):
    hc[i] = hist[i] + hc[i-1]

plt.plot(hc)
plt.show()
```

TP1

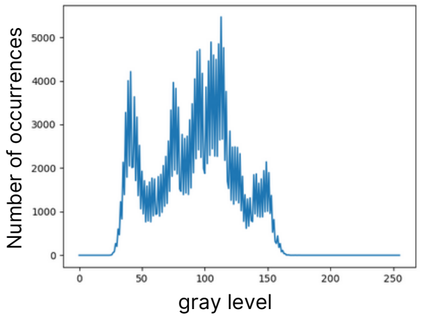
2. Contrast correction

2.6) If we perform both processes (histogram stretching then histogram equalization), does the resulting image will be better ?

Original image

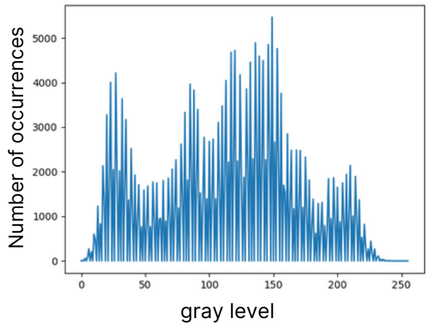


histogram

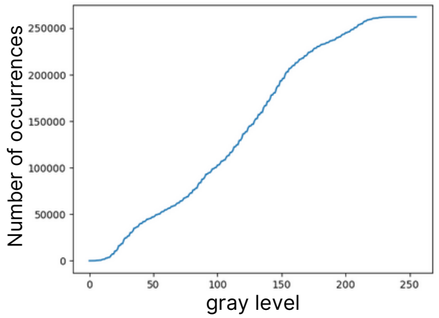


Stretched image

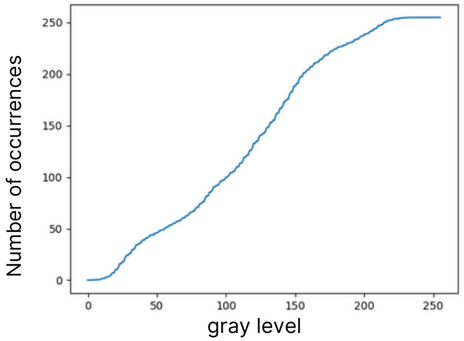
streched histogram



Cumulated histogram of the Stretched image

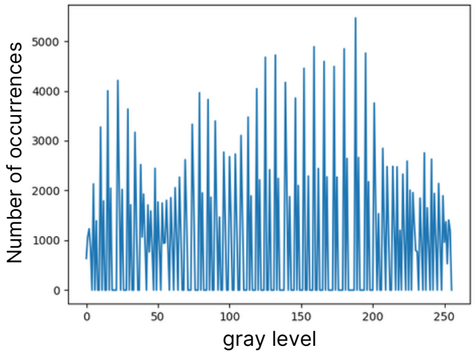


Normalized cumulated histogram of the Stretched image

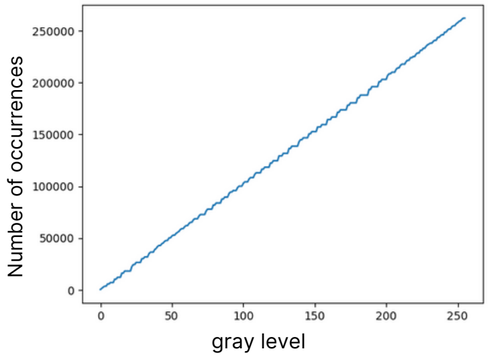


Stretched image after histogram equalization

histogram of the equalized image



Cumulated histogram of the equalized image



Interprétation : La correction de l'étirement de l'histogramme de l'image originale a amélioré sa luminosité, et en appliquant l'égalisation, nous avons également amélioré le contraste. La combinaison des deux processus produit une image plus nette.

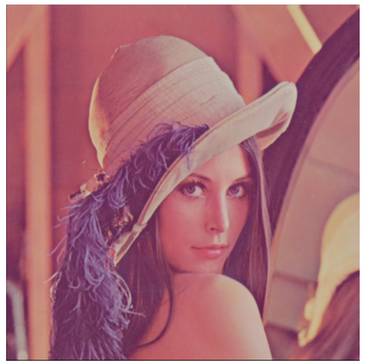
TP1

3. Colored contrast correction

Image weak_colored_contrast__old.png suffers from a lack of contrast. Doing the same process as above with some adjustments could correct this lack of contrast.

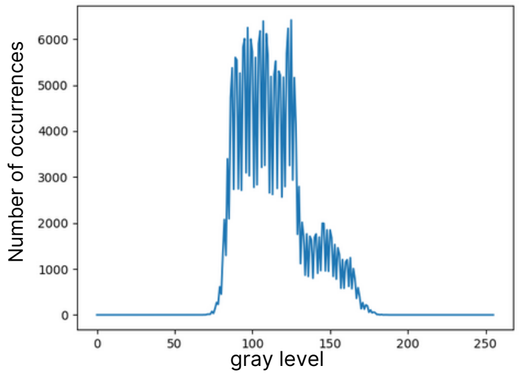
```
image = cv2.imread("./Lena_weak_v2.png")
B = image[:,:,0]
V = image[:,:,1]
R = image[:,:,2]

#pour B-----------------------|-------------------------------------------------------
hist = np.zeros(256, int)       # prépare un vecteur de 256 zéros (pour chaque gris)
for i in range(0,image.shape[0]):      # énumère les lignes
    for j in range(0,image.shape[1]):  # énumère les colonnes
        hist[B[i,j]] = hist[B[i,j]] + 1

print(hist)
plt.plot(hist)
plt.show()

# Calcule l'histogramme cumulé hc
hc = np.zeros(256, int)         # prépare un vecteur de 256 zéros
hc[0] = hist[0]
for i in range(1,256):
    hc[i] = hist[i] + hc[i-1]

plt.plot(hc)
plt.show()

# normalize the cumulated histogram
nbpixels = B.size
hc = hc / nbpixels * 255
print(hc)
plt.plot(hc)
plt.show()

res_B=B.copy()
# Utilise hc comme table de conversion des niveaux de gris
for i in range(0,B.shape[0]):       # énumère les lignes
    for j in range(0,B.shape[1]):   # énumère les colonnes
        res_B[i,j] = hc[B[i,j]]
cv2.imshow("B après égalisation", res_B)
cv2.imwrite("B_après_égalisation.png", res_B)
```

image

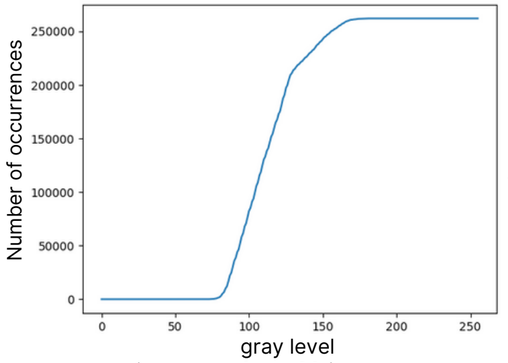


B channel image

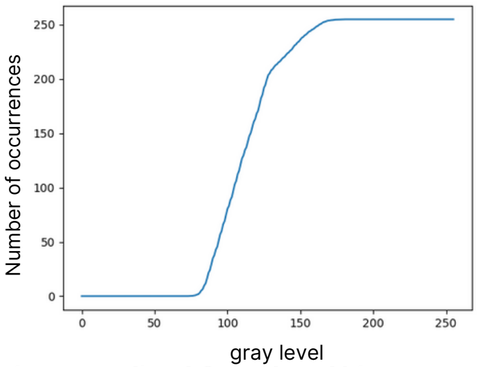
filtred B channel image



histogram of the filtred B channel image



its cumulated histogram



its normalized Cumulated histogram

TP1

3. Colored contrast correction

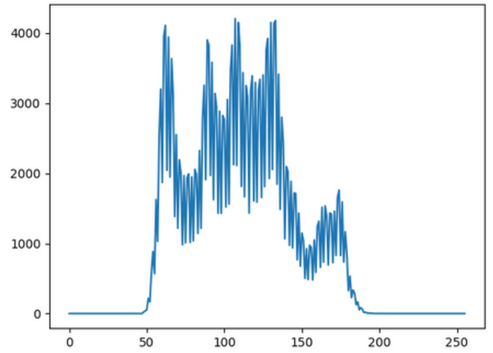
Image weak_colored_contrast__old.png suffers from a lack of contrast. Doing the same process as above with some adjustments could correct this lack of contrast.

```
#pour V-----------------------------------------------------------------------------
hist = np.zeros(256, int)       # prépare un vecteur de 256 zéros (pour chaque gris)
for i in range(0,image.shape[0]):      # énumère les lignes
    for j in range(0,image.shape[1]):  # énumère les colonnes
        hist[V[i,j]] = hist[V[i,j]] + 1

print(hist)
plt.plot(hist)
plt.show()

# Calcule l'histogramme cumulé hc
hc = np.zeros(256, int)         # prépare un vecteur de 256 zéros
hc[0] = hist[0]
for i in range(1,256):
    hc[i] = hist[i] + hc[i-1]

plt.plot(hc)
plt.show()

# normalize the cumulated histogram
nbpixels = V.size
hc = hc / nbpixels * 255
print(hc)
plt.plot(hc)
plt.show()

res_V=V.copy()
# Utilise hc comme table de conversion des niveaux de gris
for i in range(0,V.shape[0]):       # énumère les lignes
    for j in range(0,V.shape[1]):   # énumère les colonnes
        res_V[i,j] = hc[V[i,j]]

cv2.imshow("V après égalisation", res_V)
cv2.imwrite("V_après_égalisation.png", res_V)
```

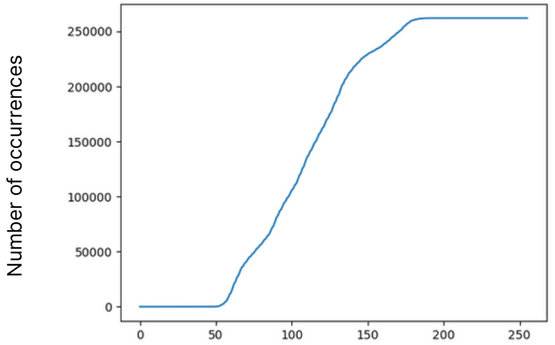
V channel image

filtred V channel image

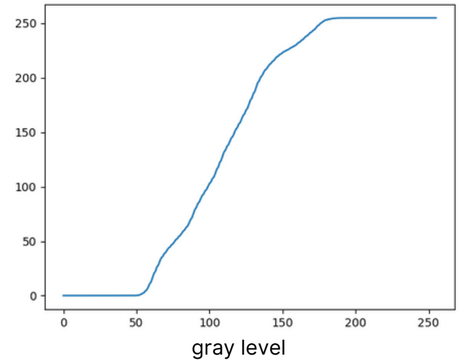
Histogram of the filtred V channel image



its cumulated histogram



its normalised cumulated histogram



TP1

3. Colored contrast correction

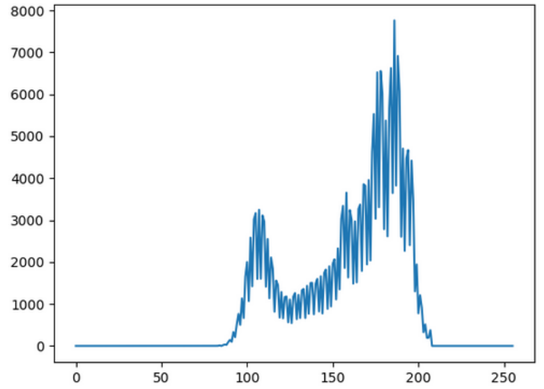
Image weak_colored_contrast__old.png suffers from a lack of contrast. Doing the same process as above with some adjustments could correct this lack of contrast.

```
#pour R-----------------------------------------------------------------------
hist = np.zeros(256, int)       # prépare un vecteur de 256 zéros (pour chaque gris)
for i in range(0,image.shape[0]):      # énumère les lignes
    for j in range(0,image.shape[1]):  # énumère les colonnes
        hist[R[i,j]] = hist[R[i,j]] + 1

print(hist)
plt.plot(hist)
plt.show()

# Calcule l'histogramme cumulé hc
hc = np.zeros(256, int)         # prépare un vecteur de 256 zéros
hc[0] = hist[0]
for i in range(1,256):
    hc[i] = hist[i] + hc[i-1]

plt.plot(hc)
plt.show()

# normalize the cumulated histogram
nbpixels = R.size
hc = hc / nbpixels * 255
print(hc)
plt.plot(hc)
plt.show()

res_R=R.copy()
# Utilise hc comme table de conversion des niveaux de gris
for i in range(0,R.shape[0]):       # énumère les lignes
    for j in range(0,R.shape[1]):   # énumère les colonnes
        res_R[i,j] = hc[R[i,j]]
cv2.imshow("R après égalisation", res_R)
cv2.imwrite("R_après_égalisation.png", res_R)
```

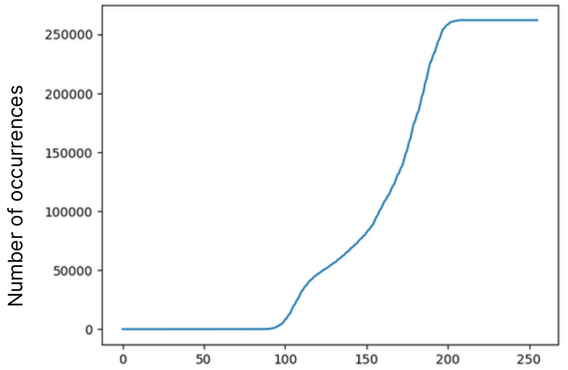
R channel image

filtred R channel image

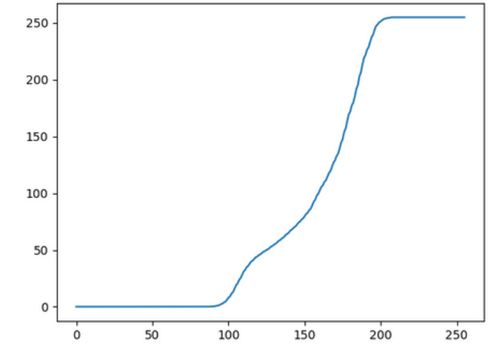
Histogram of the filtred R channel image



its cumulated histogram



its normalised cumulated histogram



TP1

3. Colored contrast correction

Image weak_colored_contrast__old.png suffers from a lack of contrast. Doing the same process as above with some adjustments could correct this lack of contrast.

```
#save
image_filt = image.copy()
image_filt[:,:,0] = res_B.copy()
image_filt[:,:,1] = res_V.copy()
image_filt[:,:,2] = res_R.copy()

cv2.imwrite('fusion_enhanced_rgb.png',image_filt)
cv2.imshow("fusion_enhanced_rgb", image_filt)
```

BVR channels fusion resulted image

Interprétation: la fusion des channels transformés (après égalisation) donne une image résultante avec des couleurs bizarres .

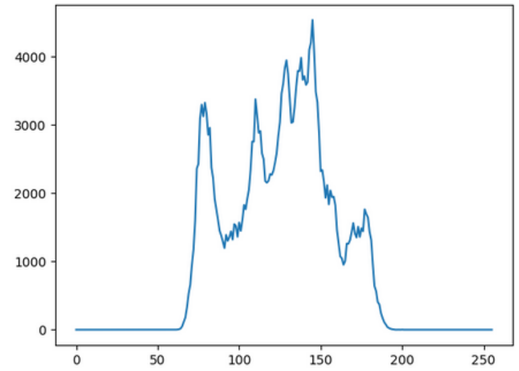
3. Colored contrast correction

Image weak_colored_contrast__old.png suffers from a lack of contrast. Doing the same process as above with some adjustments could correct this lack of contrast.

Solution: on va utiliser une autre stratégie cette fois pour remédier à ce problème d'apparition de couleurs bizarre. C'est le passage dans un espace couleur différent qui s'appelle **YCrCb** constitué d'une composante de luminance et de deux composantes couleurs. On va égaliser l'histogramme de la composante de luminance uniquement , car on a remarqué que si on touche aux composantes couleurs, alors ça affiche de nouvelles couleurs bizarres.

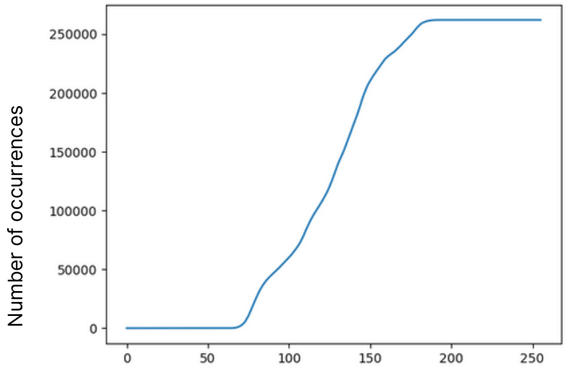
```
# en convertissant vers YCrCb
ycrcb_img = cv2.cvtColor(image, cv2.COLOR_BGR2YCrCb)
Y = ycrcb_img[:,:,0]
hist = np.zeros(256, int)       # prépare un vecteur de 256 zéros
for i in range(0,image.shape[0]):      # énumère les lignes
    for j in range(0,image.shape[1]):  # énumère les colonnes
        hist[Y[i,j]] = hist[Y[i,j]] + 1

print(hist)
plt.plot(hist)
plt.show()

# Calcule l'histogramme cumulé hc
hc = np.zeros(256, int)         # prépare un vecteur de 256 zéros
hc[0] = hist[0]
for i in range(1,256):
    hc[i] = hist[i] + hc[i-1]

plt.plot(hc)
plt.show()

# normalize the cumulated histogram
nbpixels = Y.size
hc = hc / nbpixels * 255
print(hc)
plt.plot(hc)
plt.show()

# Utilise hc comme table de conversion des niveaux de gris
for i in range(0,Y.shape[0]):       # énumère les lignes
    for j in range(0,Y.shape[1]):   # énumère les colonnes
        ycrcb_img[i,j,0] = hc[Y[i,j]]

image_filt = cv2.cvtColor(ycrcb_img, cv2.COLOR_YCrCb2BGR)
cv2.imwrite('enhanced_weak_color_lena_v2.png',image_filt)
```

Histogram of the Y channel of the image in YCrCb space

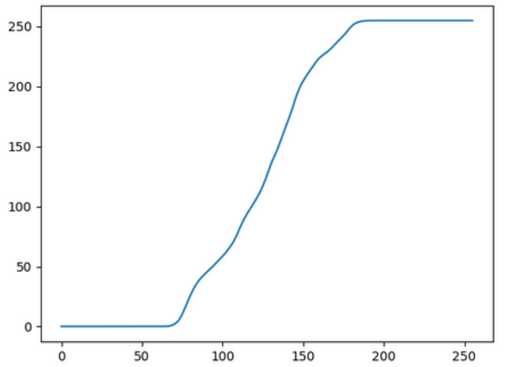


Resulted image after Y channel equalization

its cumulated histogram



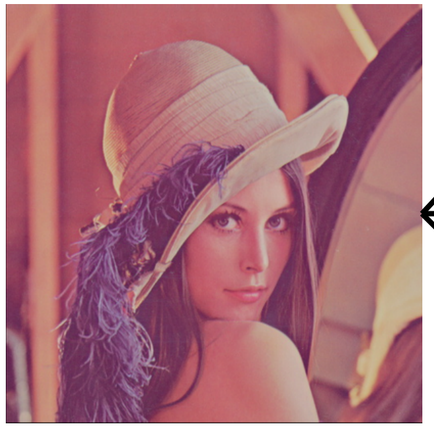
its normalised cumulated histogram



TP1

3. Colored contrast correction

To summarize:



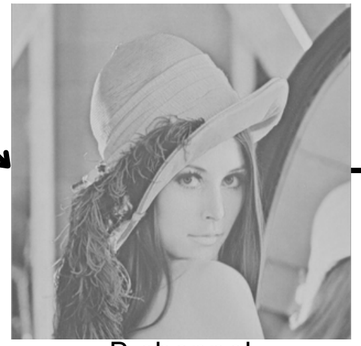
Original image

B channel

Filtred B channel

V channel

Filtred V channel



R channel

Filtred R channel

Fusion image

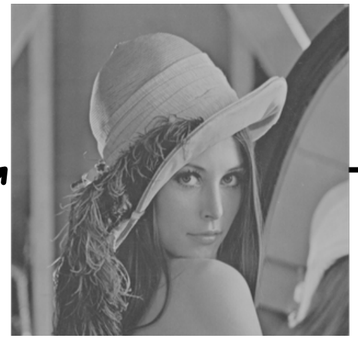
Interprétation: à partir de l'image originale on obtient 3 channels, ensuite on égalise leurs histogrammes pour obtenir un résultat pas bon !

TP1

3. Colored contrast correction

To summarize:

Original image in YCrCb space



Y channel

Y channel after equalization

Cr channel

Cb channel

Resulted image

Interprétation: On transforme l'image original en YCrCb et on montre les 3 composantes Y, Cr et Cb, puis on égalise uniquement l'histogramme de la composante de luminance. En modifiant la composante Y par la composante Y égalisée, on obtient une image avec un meilleur contraste

TP1

TP2: Notebook

link: https://drive.google.com/file/d/1RkFq1pMYBJR_17sUXJ4nhdFWL9Onw9gI/view?usp=sharing

Image Processing and Analysis

- Fundamental Concepts in Imaging
- Geometric Operations on the Image
- Histogram of an image
- Averaging and Median filters

TP2 BIS

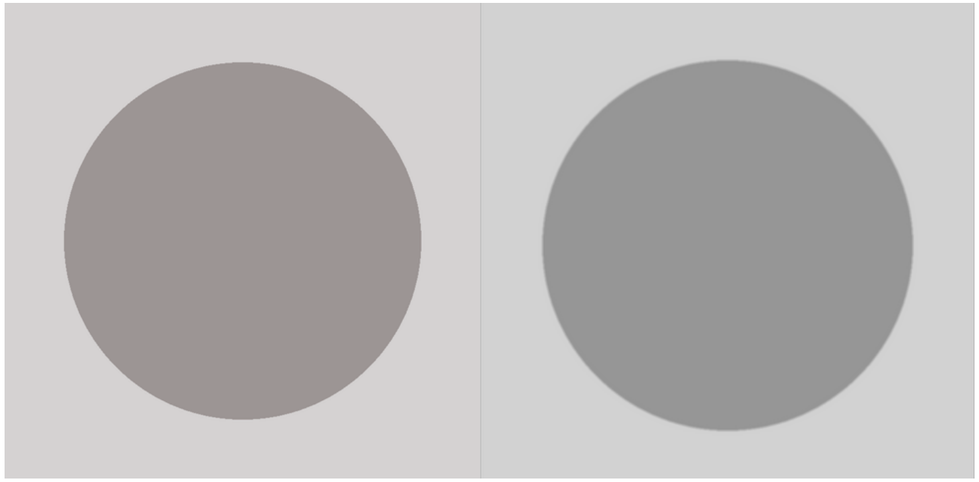
Image Processing and Analysis

- Averaging Filter
- Median Filter

1. Averaging Filter

1.1) Using OpenCV, load the gray_disc.png as a grayscale image into a variable named img from the ressources provided in this practical work.
1.2) Define two averaging filters h1 and h2 of size (5x5) and (11x11) respectively, and use them to filter the image using the OpenCV built-in functionfilter2D. Read its documentation in order to use it !

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load image
img = cv2.imread('./ressources/gray_disc.png', 0)

# Define kernels
h1 = 1/25 * np.ones((5, 5))
h2 = 1/121 * np.ones((11, 11))
h3 = 1/484 * np.ones((22, 22))

# Apply filters
res_h1 = cv2.filter2D(img, ddepth=-1, kernel=h1)
res_h2 = cv2.filter2D(img, ddepth=-1, kernel=h2)
res_h3 = cv2.filter2D(img, ddepth=-1, kernel=h3)

# Display the images
cv2.imshow("Image filtered 1", res_h1)
cv2.imshow("Image filtered 2", res_h2)
cv2.imshow("Image filtered 3", res_h3)

# Extract line 400 from the original and filtered images
line_original = img[400, :]
line_filtered = res_h2[400, :]
line_filtered_ = res_h3[400, :]
```



Original image

filter 1

filter 2

filter 3

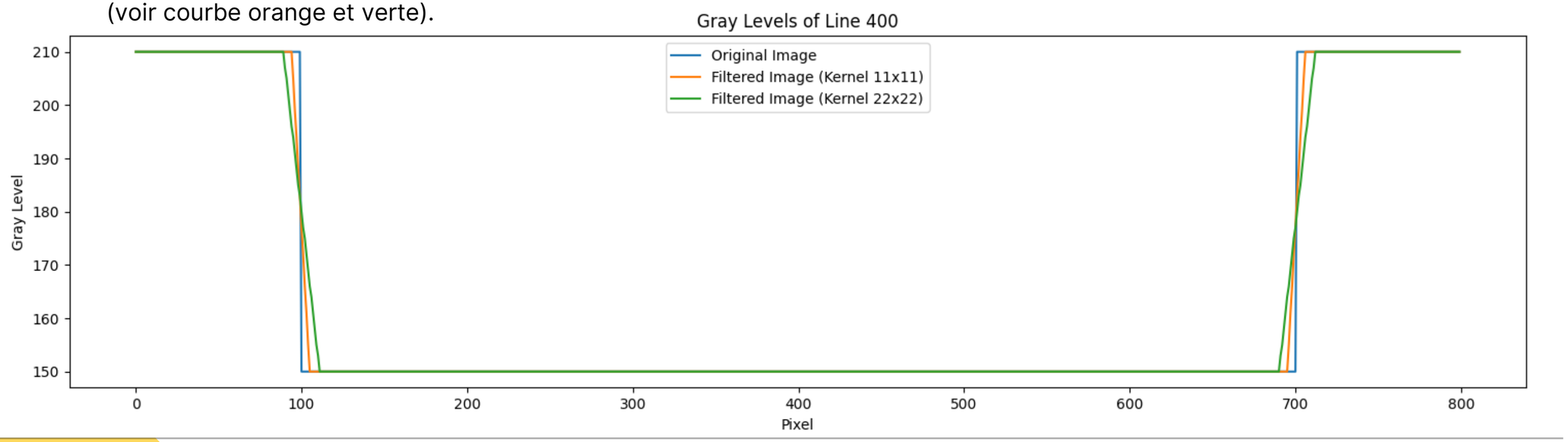
TP2

1. Averaging Filter

1.3) What can you notice from the obtained results related to the question 1.2 ?
1.4) Plot the gray levels of the line 400 of the original image and the filtered image with the kernel of size 11x11. What are your remarks? What can you deduce ?

Interprétation: 1.3) Nous remarquons que sur des images intactes, l'application d'un filtre moyenneur ajoute du floud sur l'image.
1.4) nous remarquons à partir des plots des lignes 400, que le filtrage moyenneur adoucit les variations brusque de niveaux de gris. Par exmeple, pour la la courbe bleue représentant les niveaux de gris de la ligne 400 de l'image de référence, nous remarques un violent changement de niveau de gris est bien visible aux alentours du pixel n° 100. Le niveau de gris était de 210 et passe brusquement à 150.
En revanche, pour les images filtrées, ce changement se fait doucement (voir courbe orange et verte).

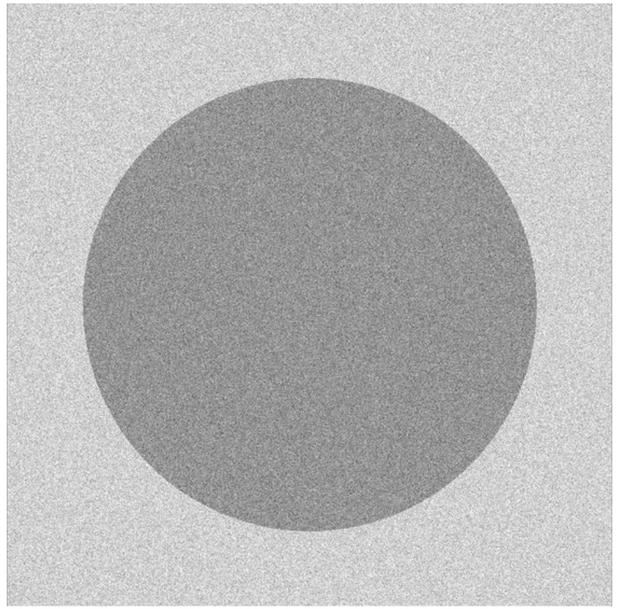
```
# Extract line 400 from the original and filtered images
line_original = img[400, :]
line_filtered = res_h2[400, :]
line_filtered_ = res_h3[400, :]

# Plot gray levels
plt.plot(line_original, label='Original Image')
plt.plot(line_filtered, label='Filtered Image (Kernel 11x11)')
plt.plot(line_filtered_, label='Filtered Image (Kernel 22x22)')
plt.xlabel('Pixel')
plt.ylabel('Gray Level')
plt.title('Gray Levels of Line 400')
plt.legend()
plt.show()
```



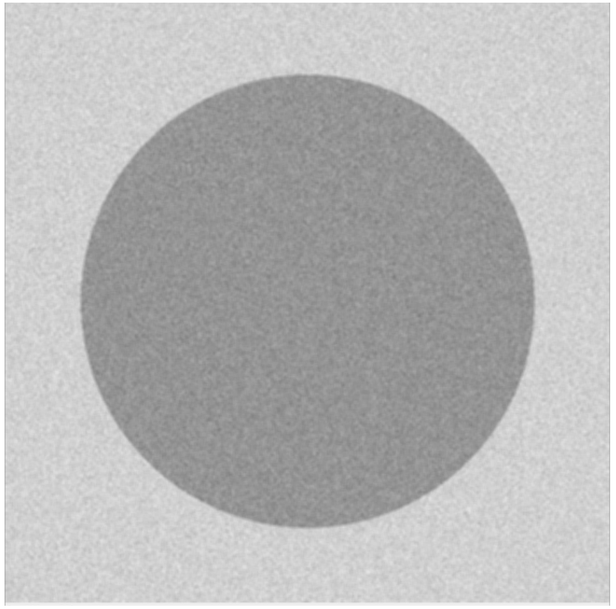
TP2

1. Averaging Filter

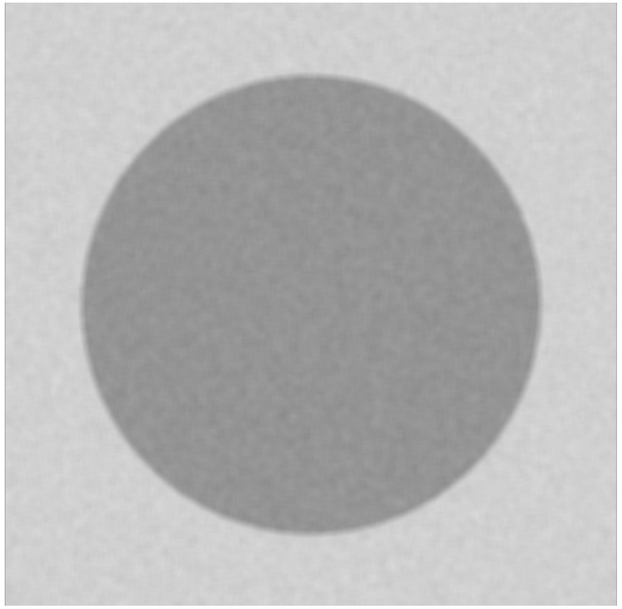
1.5) Load the noised_disc.png in a variable named noised_img and visualise it. What kind of noise is that?
1.6) Let's apply the two filters h1 and h2 defined in question 1.2. What can you notice from the resulting filtered images ? What can you deduce ?



noised_image



noised_image filter 1



noised_image filter 2

Interprétation: le bruit de l'image bruitée est un bruit de type Gaussien. Le filtrage moyenneur permet de réduire le bruit tout en rajoutant un peu de flou.

2. Median Filter

2.1) Using OpenCV, load the gray2_disc_noised_s&p.png as a grayscale image into a variable named noised_sp from the ressources provided in this practical work. What type of noise has affected this image?

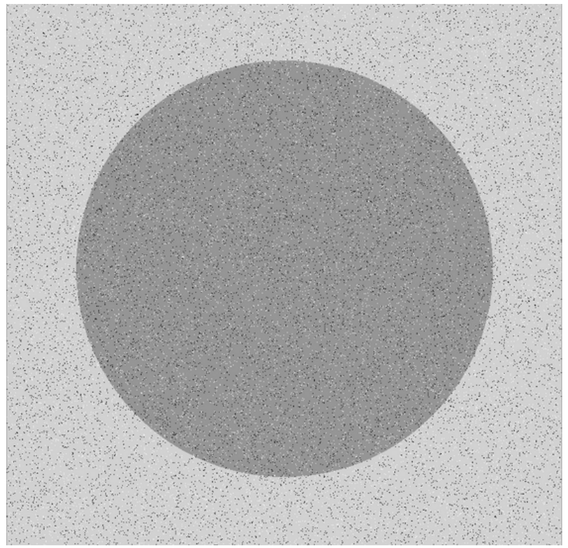
S: The type of the noise that affect this image is : Salt & Pepper

2.2) In order to denoise the latter, what is the appropriate filter to use?

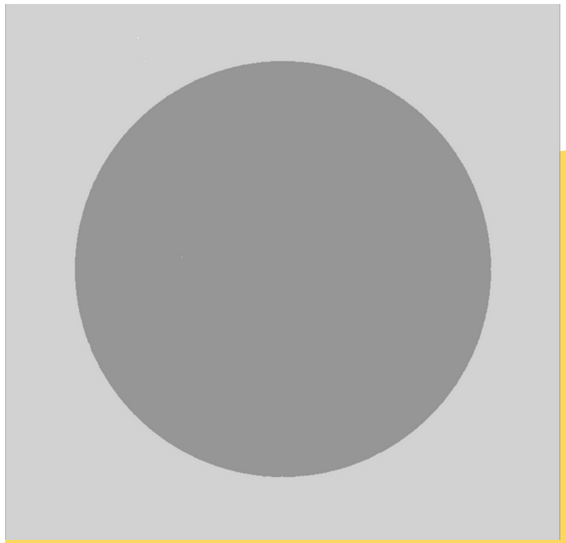
S: The appropriate filter is the Median Filter.

2.3) Let's try to denoise this image using the built-in OpenCV function medianBlur after you have read its documentation (try size 9). What can you notice?

```python
import cv2
import numpy as np

# Load image
noised_sp = cv2.imread('./ressources/gray_disc_noised_s&p.png', 0)
cv2.imshow("noised_image", noised_sp)

# Define kernels
res=cv2.medianBlur(noised_sp,3)

# Display the images
cv2.imshow("noised_image s&p", res)
```



noised_image s&p



filtred image

Interprétation: On remarque que le filtre médian permet une supression efficace du bruit Salt & Pepper.

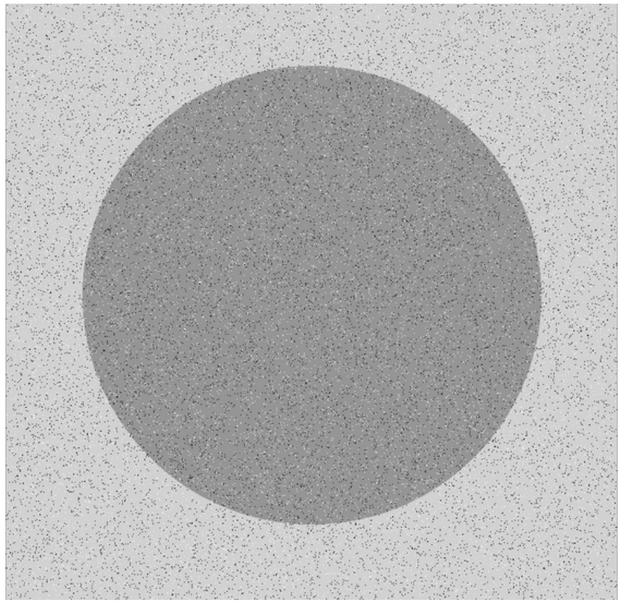
TP2

2. Median Filter

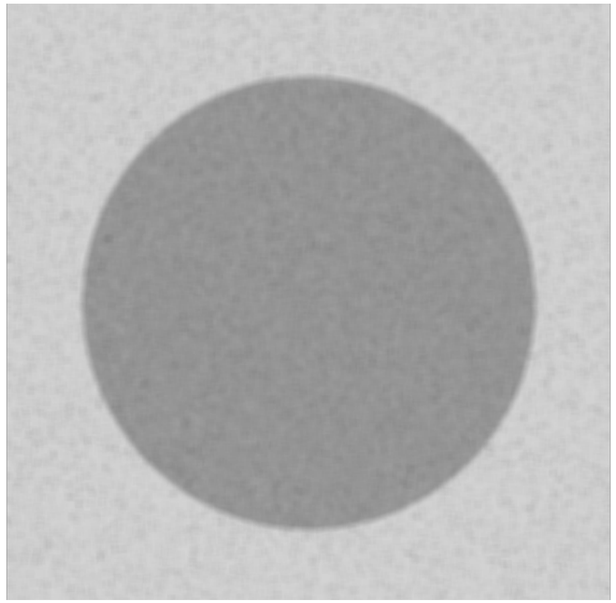
2.4) Let's try to denoise the image using the ilter h2 de ined in the question 1.2. What can you notice/deduce ?

```
# Load image
noised_sp = cv2.imread('./ressources/gray_disc_noised_s&p.png', 0)
h2 = 1/121 * np.ones((11, 11))

# Apply filters
res_h2 = cv2.filter2D(noised_sp, ddepth=-1, kernel=h2)

cv2.imshow("noised_image filtered 2", res_h2)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Interprétation: On remarque que le filtre moyeneur n’est pas efficace pour la suppression du bruit Poivre et Sel



noised_image s&p



filtred image

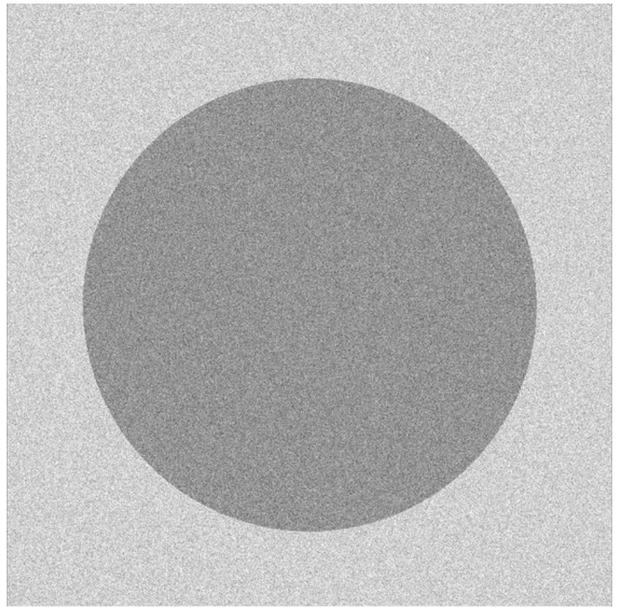
TP2

2. Median Filter

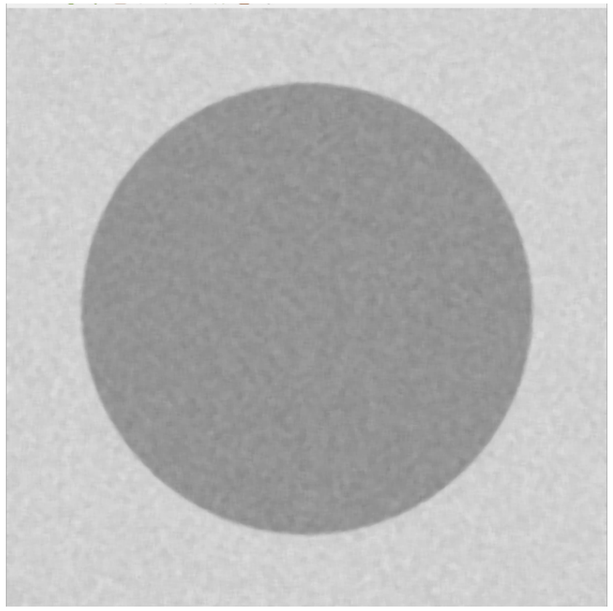
2.5) Let's try to denoise the image affected with the gaussian noise with a median filter of size 9. What can you notice/deduce?

```
# Load image
noised_image = cv2.imread('./ressources/gray_disc_noised_gaussian.png', 0)
cv2.imshow("noised_image", noised_image)

# Define kernels
res=cv2.medianBlur(noised_image,9)

# Display the images
cv2.imshow("filtred_image ", res)
```

Interprétation: Nous remarquons que le filtre Median n'est pas efficace pour la suprpession du bruit Salt & Pepper



noised_image



filtred image

TP3

Fourier transform

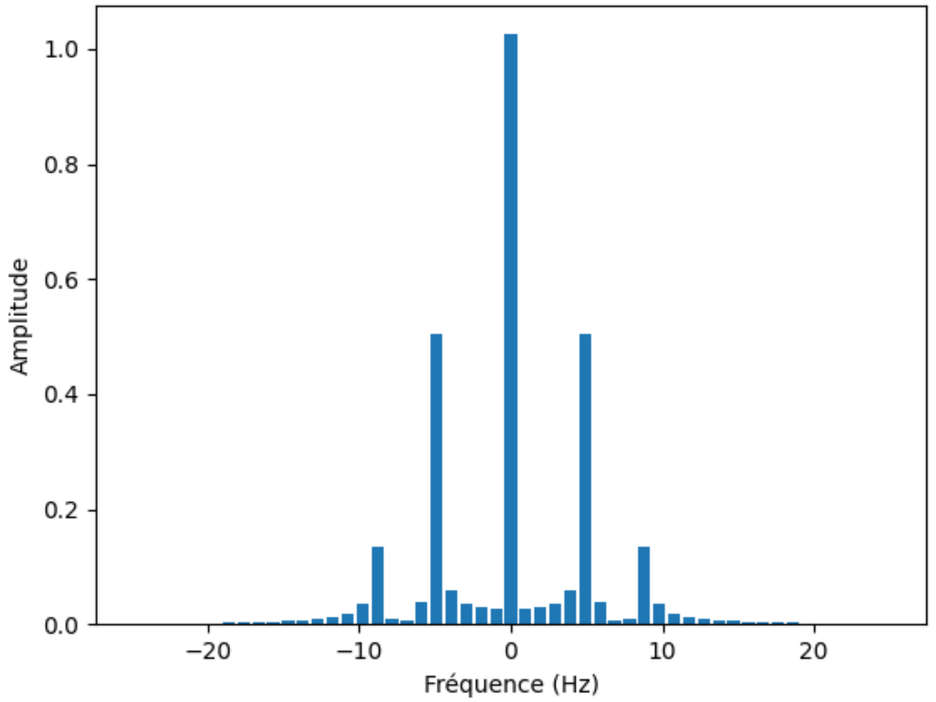
- On (1D) signal
- On sinusoidal images (2D)
- On synthetic images (2D)
- Filtering based (2D)

1. On (1D) signal

1.1) Construct the following signal characterized by a sampling frequency equal to SR=50 and constituted by the sum of cosine functions with frequencies equal to 5 and 9 respectively
1.2) Compute the 1D Fourier transform using the function provided in the file ressources.py. Visualize the results and interpret them.

```
from ressources import *
import numpy as np
import matplotlib.pyplot as plt
from scipy.ndimage import gaussian_filter
#1.1 et 1.2
fe=50
time = np.arange(0,1.02,1/fe)
s = 1+np.cos(2*np.pi*5*time) + 0.3*np.cos(2*np.pi*9*time)
fourier1D(s,fe)
```

Interprétation : Nous obtenons bien le spectre attendu, à savoir deux pics aux fréquences 5 Hz et 9 Hz ainsi qu'a leur opposés -5Hz et -9Hz. De plus, nous avons une raie à chaque fréquence d'échantillonnage. Mais nous remarquons que l'amplitude ne correspond pas à nos attentes. Ce qui est normale car comme nous le savons, dans la réalité les fréquences négatives n'existent pas, donc le résultat que nous trouvons par calcul est la transformée d'une somme de deux cosinus avec une amplitude de 0.5 et d'environ 0.15 pour le second cosinus. Ceux-ci correspondent respectivement à la moitié des amplitudes 1 et 0.3 que nous devrions obtenir.



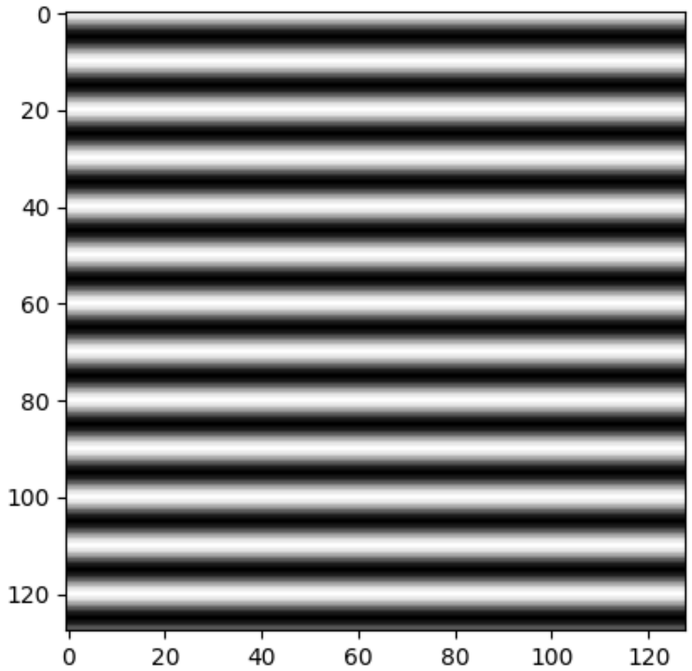
1D Fourier transform

TP3

2. On sinusoidal images (2D)

2.1) Using the wave function defined in the file wave.py, create a sinusoidal image of size 128x128 with oscillations of frequency 0.1 along the vertical axis only. Visualize the obtained image.

```
def wave(n, m, fy, fx):
        e1 = np.exp(-1j*2*np.pi*fx*np.arange(m))
        e2 = np.exp(1j*2*np.pi*fy*np.arange(n))
        img = np.outer(e2, e1).real
        return img

image= wave(128,128,0.1,0)
plt.imshow(image, cmap='gray')
plt.show()
```



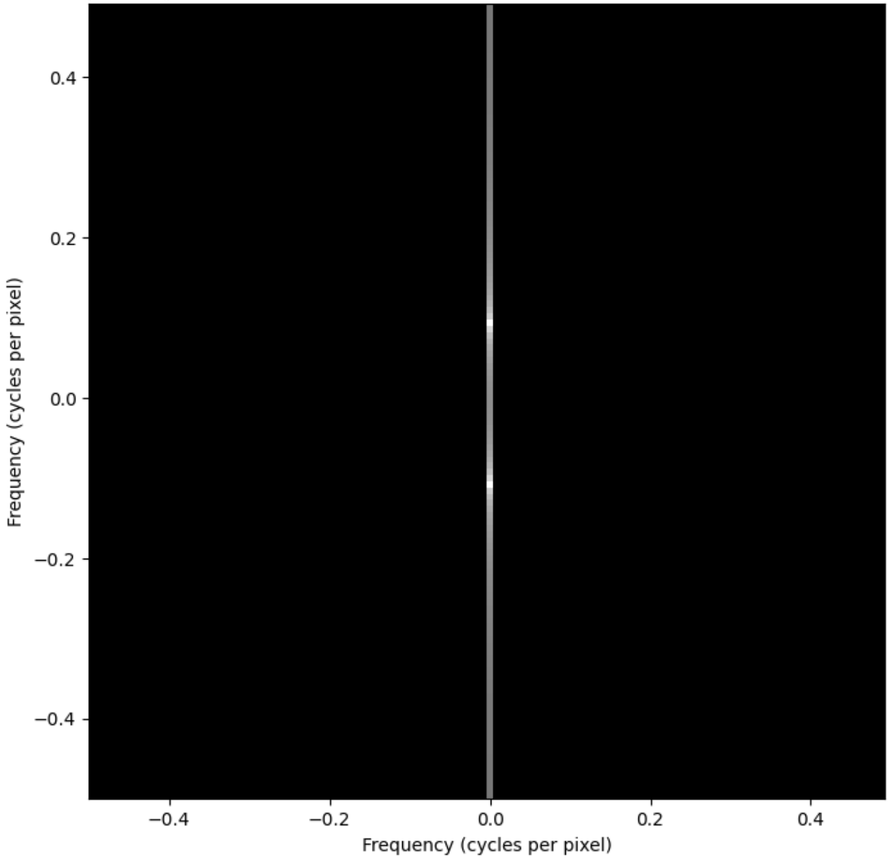
the sinusoidal image of size 128x128

TP3

2. On sinusoidal images (2D)

2.2) Using the function fourier_all, calculate the spectrum of Fourier of the image and visualize it. Interpret your results.

```
#Fonction modifié:
def fourier2d_single_frenquency(img):
    f = np.fft.fft2(img)
    fshift = np.fft.fftshift(f)
    magnitude_spectrum = 20*np.log(np.abs(fshift+1))
    rows, cols = img.shape
    freq_x = np.fft.fftfreq(cols)
    freq_y = np.fft.fftfreq(rows)
    plt.imshow(magnitude_spectrum, cmap='gray', extent=(freq_x.min(),
freq_x.max(), freq_y.min(), freq_y.max()))
    plt.xlabel('Frequency (cycles per pixel)')
    plt.ylabel('Frequency (cycles per pixel)')
    plt.show()

def fourier2d_all(img):
    # calcul de la transformée de fourrier 1d le long de l'axe x et y
    f_x=np.fft.fft(img, axis=0)
    f_y=np.fft.fft(img, axis=1)

    # Détection des fréqunces dominantes le long de x et y
    max_freq_x =np.argmax(np.abs(f_x))
    max_freq_y= np.argmax(np.abs(f_y))

    # Si l'image a une seule fréquence dominant
    if (max_freq_x== 0 and max_freq_y != 0) or (max_freq_x != 0 and max_freq_y ==
0):
        fourier2d_single_frenquency(img)
    else:
        fourier2d_many_frequencies(img)

image_ = fourier2d_all(image)
```



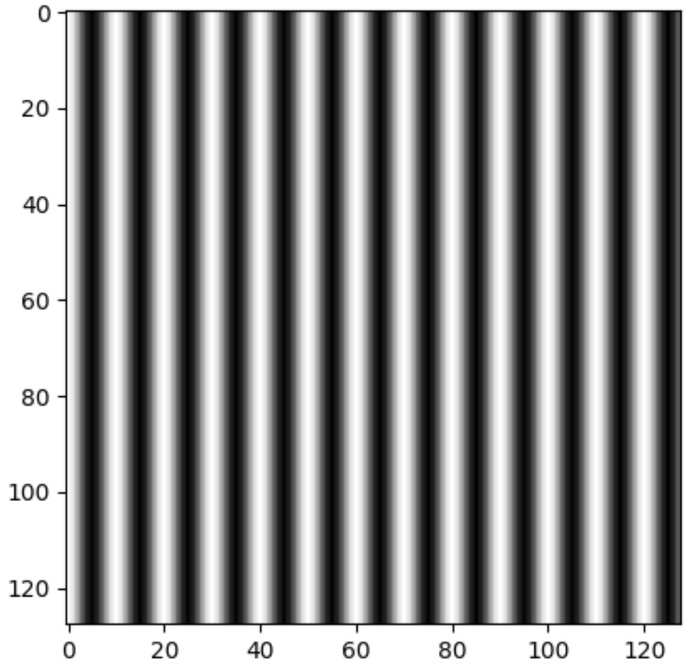
the spectrum of Fourier of the image

TP3

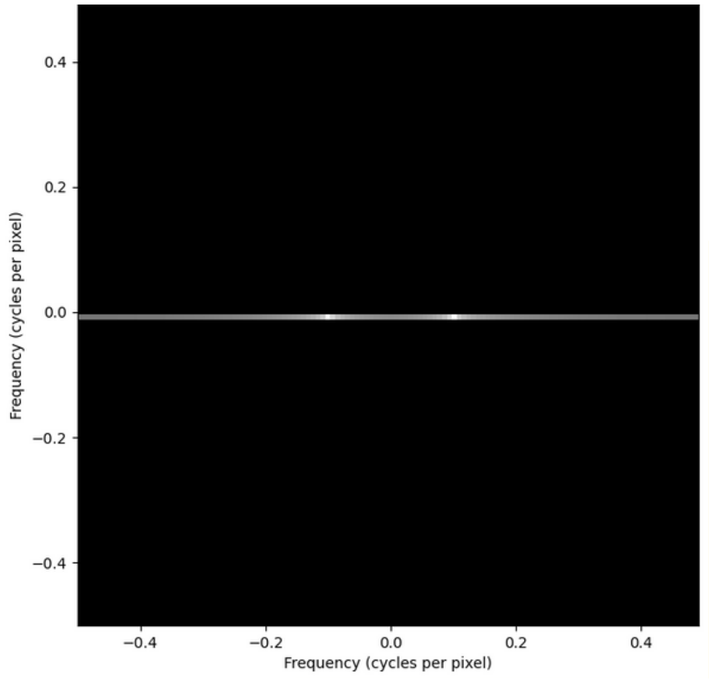
2. On sinusoidal images (2D)

2.3) try to generate oscillations of frequency 0.1 along the horizontal axe. Visualize the image. Compute its Fourier spectrum, visualize it and interpret it.

```
image_h= wave(128,128,0,0.1)
plt.imshow(image_h, cmap='gray')
plt.show()
f_h = fourier2d_all(image_h)
```



the sinusoidal image of size 128x128



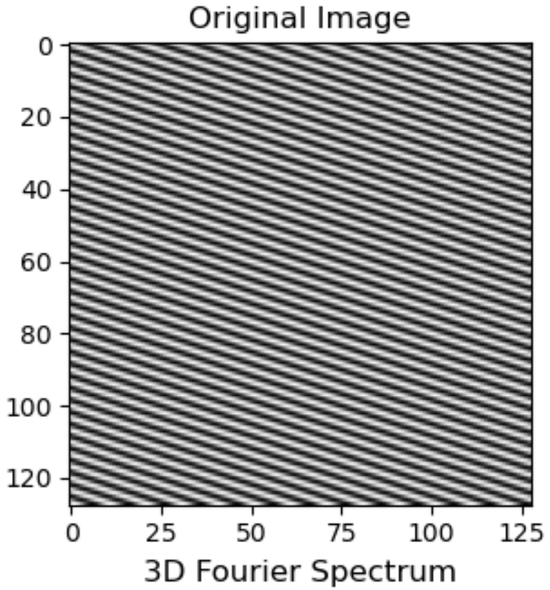
the spectrum of Fourier of the image

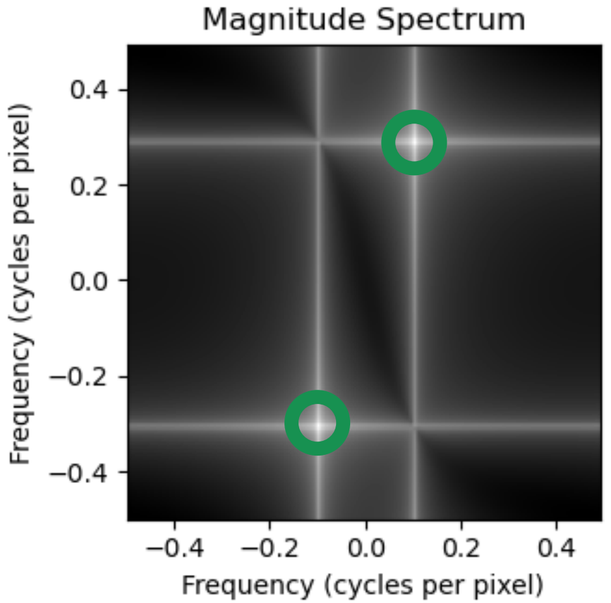
TP3

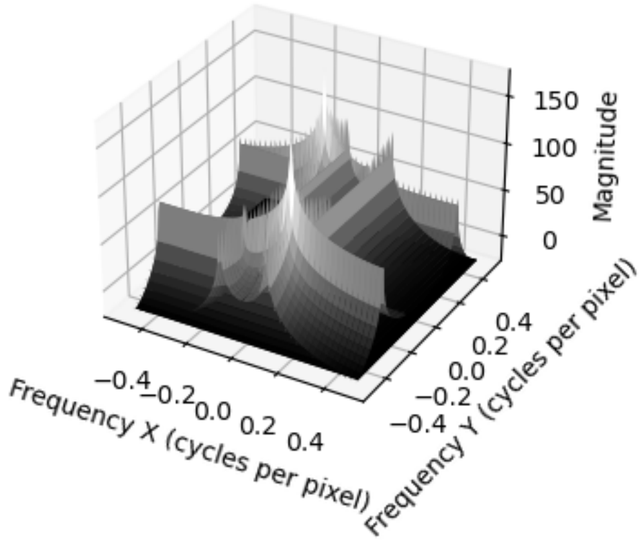
2. On sinusoidal images (2D)

2.4) same as 2.3, but with frequencies of 0.3 and 0.1 on both axes.

```
image_hv= wave(128,128,0.3,0.1)
plt.imshow(image_hv, cmap='gray')
plt.show()
f_hv = fourier2d_all(image_hv)

#Fonction modifié:
def fourier2d_many_frequencies(img):
    # transformation de fourier Rapide éD
    f = np.fft.fft2(img)
    fshift = np.fft.fftshift(f)
    magnitude_spectrum = 20*np.log(np.abs(fshift))

    # intervalle des fréquences sur x et
    rows, cols = img.shape
    freq_x = np.fft.fftfreq(cols)
    freq_y = np.fft.fftfreq(rows)

    # affichage de l'image originale
    plt.subplot(2, 2, 1)
    plt.imshow(img, cmap='gray')
    plt.title('Original Image')

    # Affichage du spectre de fourrier
    plt.subplot(2, 2, 2)
    plt.imshow(magnitude_spectrum, cmap='gray', extent=(freq_x.min(),
freq_x.max(), freq_y.min(), freq_y.max()))
    plt.title('Magnitude Spectrum')
    plt.xlabel('Frequency (cycles per pixel)')
    plt.ylabel('Frequency (cycles per pixel)')

    # Affichage du spectre de fourier 3D
    plt.subplot(2, 2, 3, projection='3d')
    X, Y = np.meshgrid(freq_x, freq_y)
    Z = 20*np.log(np.abs(f))
    ax = plt.gca() # sans patramètre
    ax.plot_surface(X, Y, Z, cmap='gray')
    ax.set_xlabel('Frequency X (cycles per pixel)')
    ax.set_ylabel('Frequency Y (cycles per pixel)')
    ax.set_zlabel('Magnitude')
    plt.title('3D Fourier Spectrum')
    plt.show()
```





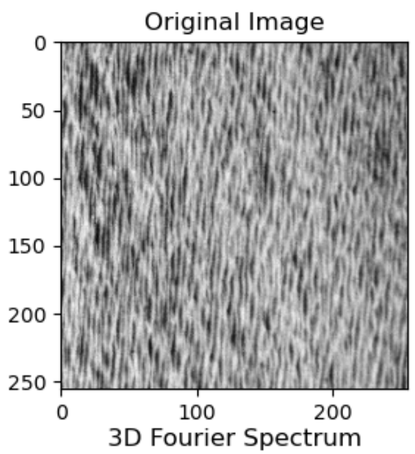


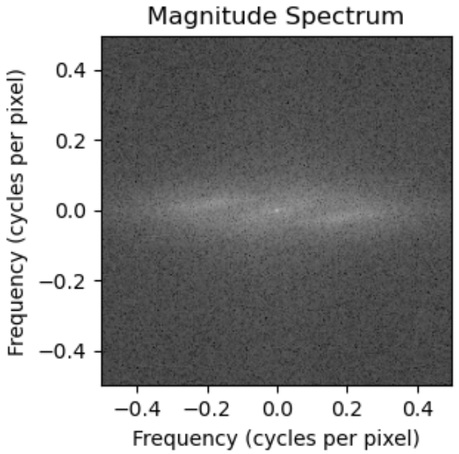
TP3

2. On synthetic images (2D)

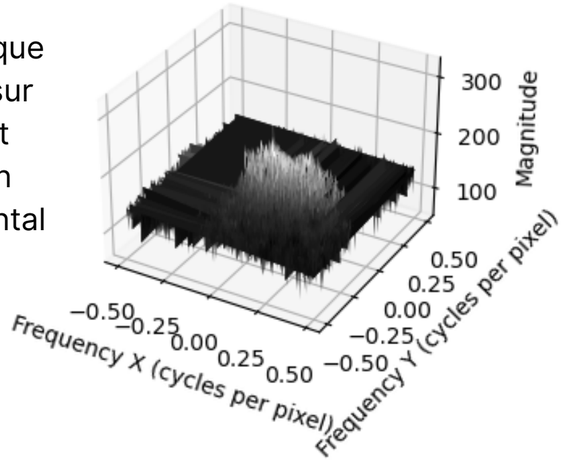
Let's do the same processing to the images located in the folder "Sytnhetic_images/" . Visualize, analyse and interpret your results.

```
image_synt = plt.imread('./synthetic_images/eau.bmp',0)
plt.imshow(image_synt, cmap='gray')
plt.show()
f_hv = fourier2d_all(image_synt)
```

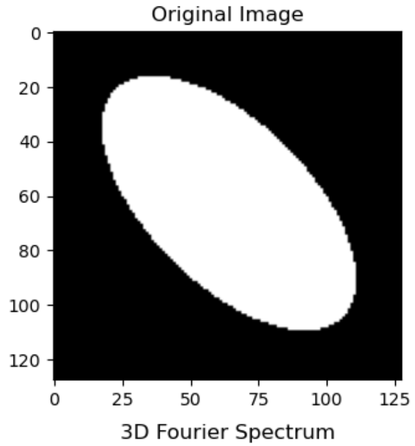


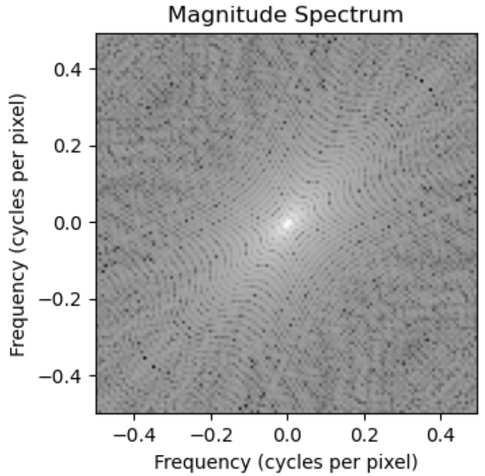


Interprétation: On remarque des gradients verticaux sur l'image originale et c'est pour ça qu'on obtient un spectre de Fourier horizontal

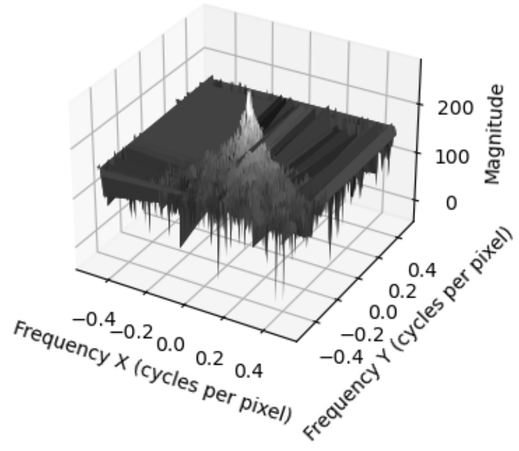



```
image_synt = plt.imread('./synthetic_images/oval5.bmp',0)
plt.imshow(image_synt, cmap='gray')
plt.show()
f_hv = fourier2d_all(image_synt)
```





Interprétation: Le spectre de Fourier a une orientation opposée aux gradients de l'image synthétique contenant une ellipse. En effet, les plus fortes et longues variations vont du coin bas gauche vers le coin haut droit. Le spectre de fourrier montre bien cela.

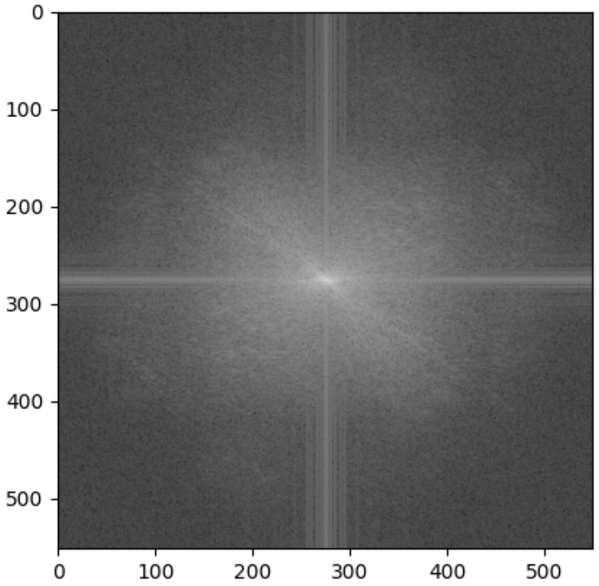


TP3

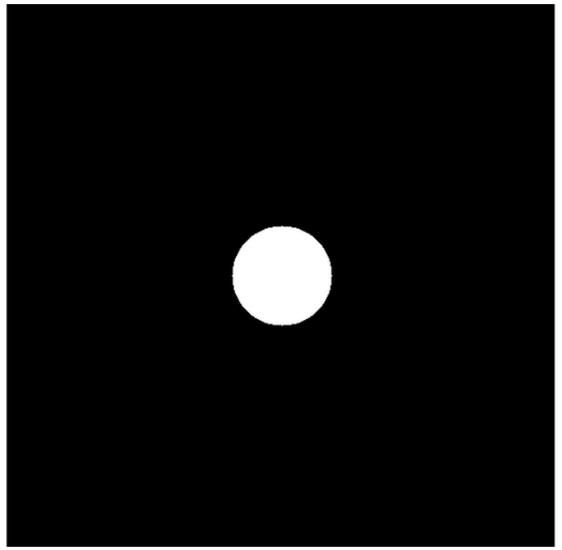
3. Filtering based (2D)

4.1) Having the Lena.png image, you are invited to implement a function that performs filtering similar to figure 1. The radius must be an entry of the function. The function should be able to plot the mask on the Fourier spectrum so you can visualize it. Try to visualize the masks and the results related to different radius of the mask.

```
from scipy.fft import fft2, ifft2, fftshift
import numpy as np
import matplotlib.pyplot as plt
from scipy.ndimage import gaussian_filter
import cv2

img = cv2.imread('/home/zaineb/Téléchargements/tp3-image-20240131T225845Z-001/
tp3-image/lena_nv.png',0)

f = fftshift(fft2(img)) #calcul du spectre de fourrier avec un shift

plt.imshow(np.log(np.abs(f)+1), cmap='gray')
plt.show()#affiche le spectre de fourrier.
```



the spectrum of Fourier of the image



Mask with radius = 50

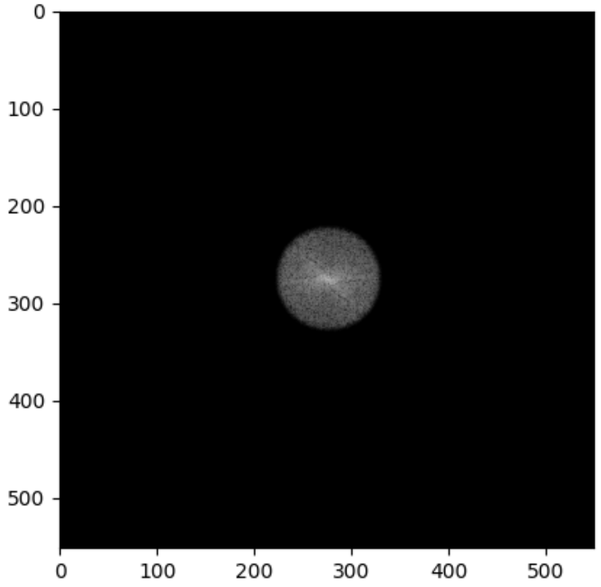
```
cx, cy = np.floor(np.array(f.shape) / 2).astype(int) #récupération des
coordonnées du centre de l'image (car le speectre et l'image ont le même centre)

center_coordinates_circle = (cx, cy)  # mise des coordonées du centre dans une
variable

img_circle = np.zeros((f.shape[0], f.shape[1]), dtype=np.uint8)

# Radius of the circle
radius = 50

# Color of the circle (B, G, R)
color = (255, 255, 255)  # White

# Thickness of -1 pixel (Fill the circle)
thickness = -1

# Using cv2.circle() method to draw a circle of white color with thickness -1 px
cv2.circle(img_circle, center_coordinates_circle, radius, color, thickness)

cv2.imshow('image_circle',img_circle) #affichage du cercle qui constituera notre
masque
#cv2.waitKey(0) #optionnel si jamais l'image du cercle ne s'affiche pas avec
cv2.imshow
```

TP3

3. Filtering based (2D)

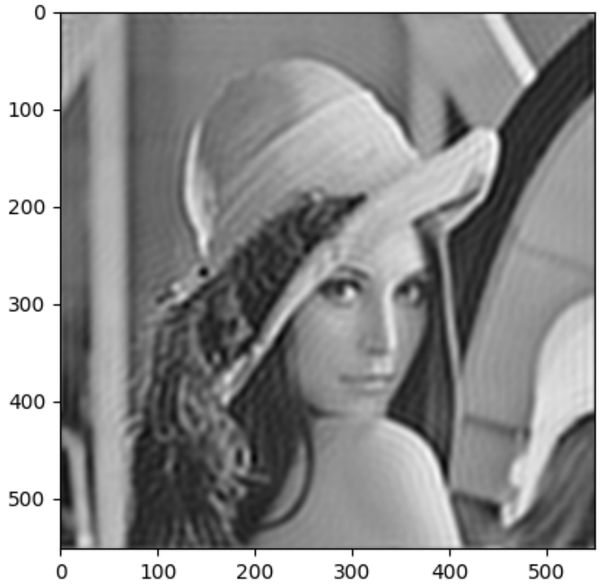
4.1) Having the Lena.png image, you are invited to implement a function that performs filtering similar to figure 1. The radius must be an entry of the function. The function should be able to plot the mask on the Fourier spectrum so you can visualize it. Try to visualize the masks and the results related to different radius of the mask.

```
#Multiplication du spectre par le cercle afin de ne garder que la partie du
spectre associée au blanc du cercle
spectre_crope =np.log(np.abs(f)+1)*img_circle #pour pouvoir cropé et visualiser
le spectre, il faut toujours le mettre dans la fonction log
plt.imshow(spectre_crope, cmap='gray')
plt.show()
```



Cropped spectre with the
Mask of radius = 50

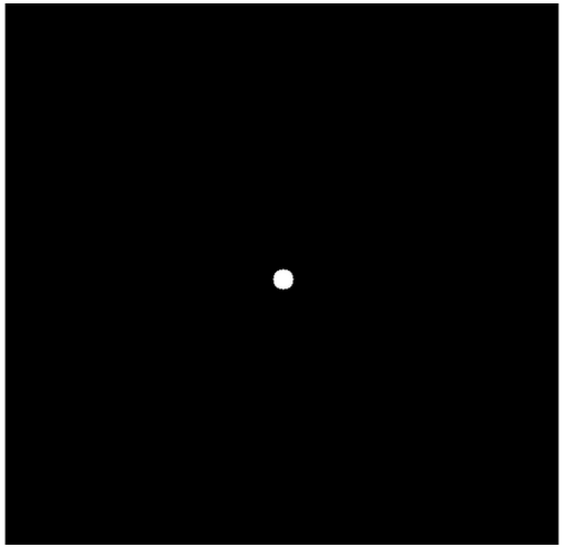
```
#reconstrution de l'image filtrée à partir de son spectre croppé
filtered_img = np.real(ifft2(fftshift(f*img_circle))) #attention, ici, on utilise
le spectre de fourrier directement, sans le mettre dans le log, car on récupère
par la suite sa partie réelle aveec np.real
plt.imshow(filtered_img, cmap='gray')
plt.show()
```



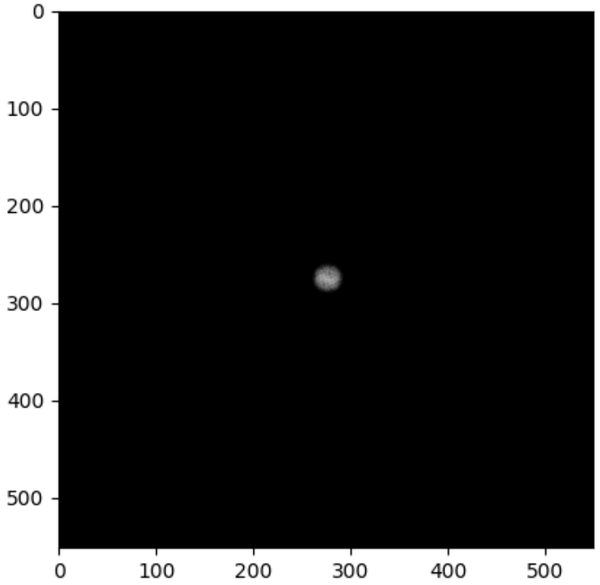
Reconstructed filtred image from
the cropped spectre

TP3

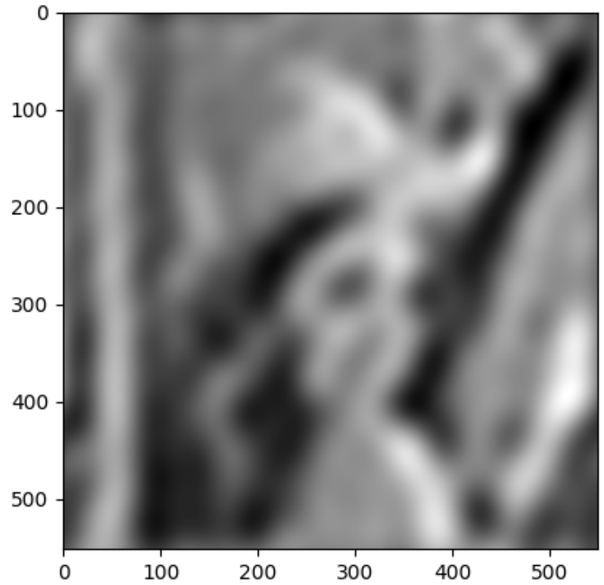
3. Filtering based (2D)



Mask with radius = 10



Cropped spectre with the Mask of radius = 10



Reconstructed filtred image from the cropped spectre

Interprétation: On remarque que les détails des images disparaissent beaucoup lorsqu'on réduit le rayon du rond blanc (du masque) car il n'y a que les basses fréquences qui sont maintenues. En effet les détails de l'image résident en dehors du centre du spectre. Plus nous supprimons les régions du spectre qui sont loin du centre du spectre, plus on supprime les détails de l'image

TP3

Questions ?

Bonne préparation