TP4 - Graphcut part 2

February 1, 2024

1 Practical work on graph-cut optimization (part 2, multilevel)

The objective of this PW is the implementation of the α-expansion and α-β swap approaches for grayscale image denoising.

The PyMaxFlow library is used to compute the graph-cut.

[ ]:
```
import math
import os
import platform
import random
import ssl
import tempfile

import matplotlib.pyplot as plt
import numpy as np
from bokeh.io import output_notebook
from bokeh.plotting import figure, output_file
from bokeh.plotting import show
from bokeh.plotting import show as showbokeh
from scipy import ndimage as ndi
from skimage import io
```

[ ]:
```
ssl._create_default_https_context = ssl._create_unverified_context
output_notebook()
```

[ ]:
```
def affiche_pour_colab(im, MINI=None, MAXI=None, titre=""):  # special colab, don't look
    def normalise_image_pour_bokeh(X, MINI, MAXI):
        if MAXI == None:
            MAXI = np.max(X)
        if MINI == None:
            MINI = np.min(X)
        imt = np.copy(X.copy())
        imt = np.clip(imt, MINI, MAXI) / (MAXI - MINI)
        imt[imt < 0] = 0
        imt[imt > 1] = 1
        imt *= 255
```

```
        sortie = np.empty((*imt.shape, 4), dtype=np.uint8)
        for k in range(3):
            sortie[:, :, k] = imt
        sortie[:, :, 3] = 255
        return sortie

    img = im
    img = normalise_image_pour_bokeh(np.flipud(im), MINI, MAXI)
    p = figure(
        tooltips=[("x", "$x"), ("y", "$y"), ("value", "@image")],
        y_range=[im.shape[0], 0],
        x_range=[0, im.shape[1]],
        title=titre,
    )
    # p.x_range.range_padding = p.y_range.range_padding = 0
    # must give a vector of images
    p.image(
        image=[im],
        x=0,
        y=0,
        dw=im.shape[1],
        dh=im.shape[0],
        palette="Greys9",
        level="image",
    )
    p.xgrid.visible = False
    p.ygrid.visible = False
    showbokeh(p)


def affiche(im, MINI=0.0, MAXI=None, titre="", printname=False):
    affiche_pour_colab(
        im, MINI=MINI, MAXI=MAXI, titre=titre
    )  # under google colab many options disappear


def display_segmentation_borders(image, bin):
    imagergb = np.copy(image)
    from skimage.morphology import binary_dilation, disk

    contour = binary_dilation(bin, disk(15)) ^ bin
    imagergb[contour == 1, 0] = 255
    imagergb[contour == 1, 1] = 0
    imagergb[contour == 1, 2] = 0
    return imagergb
```

1.1 2 Denoising a grayscale image

In this second part of the PW, we are interested in using Markovian methods to **denoise** images with different regularization potentials.

We are interested in denoising the images *Ibruitee.png* and *Ibruitee2.png* which correspond to the same scene perturbed by two noises of different nature.

We will complete programs that call the algorithm of alpha-expansions or Boykov's alpha-beta swap according to the Kolmogorov technique.

Q1: What are the respective expressions for the data attachment potentials in the case of noise following a Gaussian distribution (equation 1) and a Rayleigh distribution (equation 2)?

$$p(y_p|x_p) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(y_p - x_p)^2}{2\sigma^2}\right], \tag{1}$$

$$p(y_p|x_p) = 2\frac{y_p}{x_p^2} \exp\left[-\frac{y_p^2}{x_p^2}\right]. \tag{2}$$

Your answer

A1: Using the expression $D(y_p, x_p) = -\log p(y_p|x_p)$, we obtain:

1. Gaussian distribution:

$$\begin{aligned} D(y_p,\ x_p) &= -(\ -\log(\sqrt{2\pi}\sigma) - \frac{(y_p - x_p)^2}{2\sigma^2}) \\ &= \log(\sqrt{2\pi}\sigma) + \frac{(y_p - x_p)^2}{2\sigma^2} \end{aligned}$$

2. Rayleigh distribution:

$$\begin{aligned} D(y_p,\ x_p) &= -(\ \log(\frac{2y_p}{x_p^2}) - \frac{y_p^2}{x_p^2}) \\ &= 2\log(x_p) - \log(2y_p) + \frac{y_p^2}{x_p^2} \end{aligned}$$

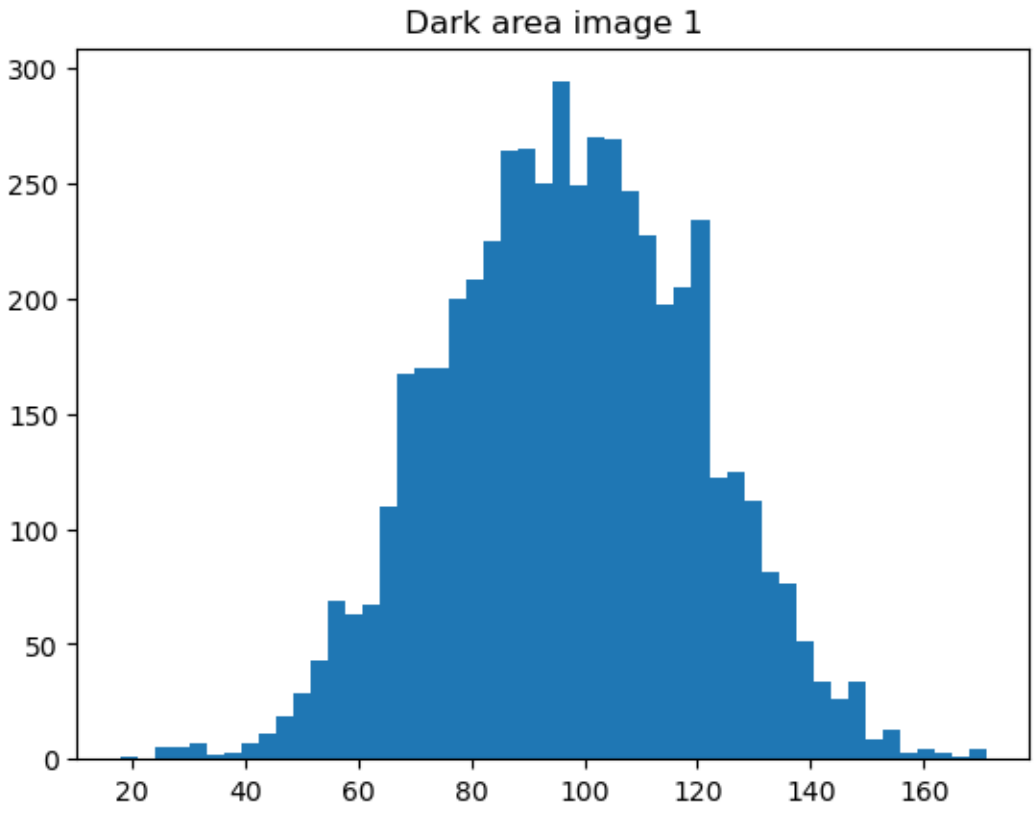
Q2: By studying the histogram of a homogeneous area, indicate which type of noise is present in which image.

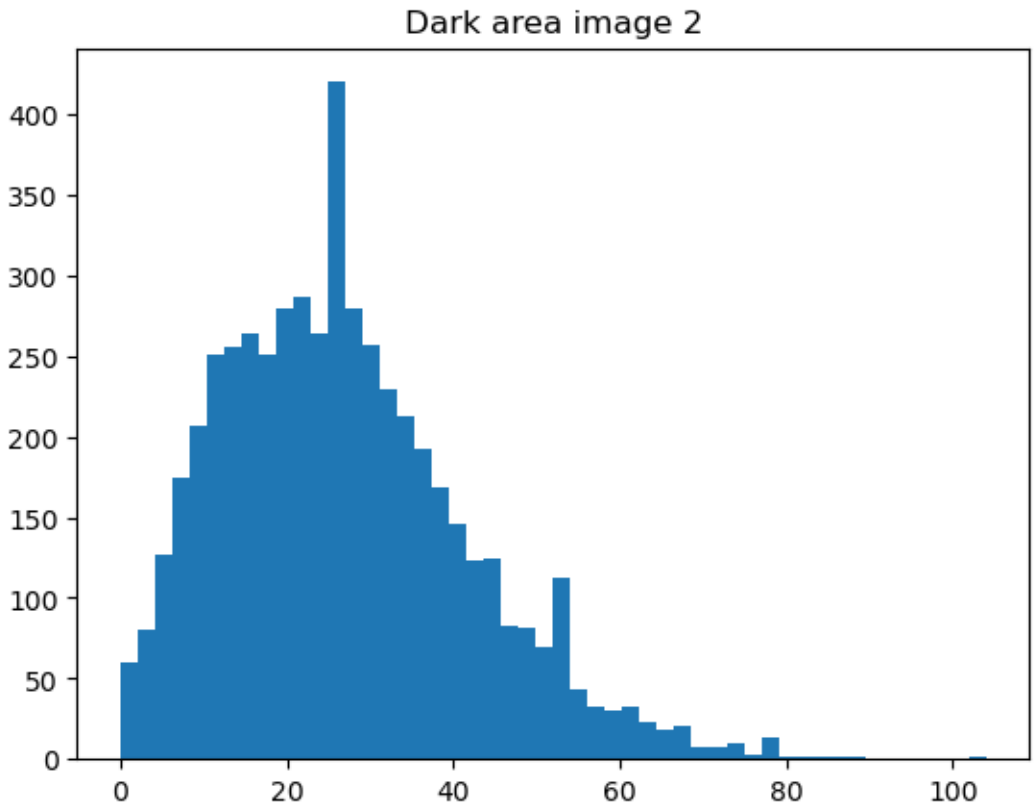
Your answer

A2: The noise on the first image seems to follow a Gaussian distribution, and the noise on the second image seems to follow a Rayleigh distribution.

[ ]:
```
im_obs = io.imread(
    "https://perso.telecom-paristech.fr/tupin/TPGRAPHCUT/OLD/Ibruitee.png"
)  # Observed image, noisy

im_obs2 = io.imread(
    "https://perso.telecom-paristech.fr/tupin/TPGRAPHCUT/OLD/Ibruitee2.png"
)  # Observed image, noisy

im_orig = io.imread(
    "https://perso.telecom-paristech.fr/tupin/TPGRAPHCUT/OLD/IoriginaleBW.png"
)  # Reference binary image, to evaluate the quality of the segmentation

I = im_obs

affiche(im_obs, MINI=0.0, MAXI=255.0, titre="Noisy image 1", printname=True)
affiche(im_obs2, MINI=0.0, MAXI=255.0, titre="Noisy image 2", printname=True)
affiche(im_orig, MINI=0.0, MAXI=255.0, titre="Original image", printname=True)
```

[ ]:
```
# select a homogeneous area to study the histogram
# we can use the area from 45 to 120 and from 175 to 245
affiche(
    im_obs[45:120, 175:245], MINI=0.0, MAXI=255.0, titre="Noisy image",␣
 ↪printname=True
)

plt.figure()
plt.hist(im_obs[45:120, 175:245].flatten(), 50)
plt.title("Dark area image 1")
plt.show()

plt.figure()
plt.hist(im_obs2[45:120, 175:245].flatten(), 50)
plt.title("Dark area image 2")
plt.show()
```


Dark area image 1
300
250
200
150
100
50
0
20
40
60
80
100
120
140
160



We will compare three *a priori* models: the Potts model $\delta_{x_p=x_q}$, the (discrete) total variation $|x_p - x_q|$, and the (quadratic) Gaussian model $(x_p - x_q)^2$.

Q3: Are they metrics or semi-metrics? What can we deduce about the optimization method to use?

Your answer

A3:

A model is considered semi-metric if $\forall \alpha, \beta \in E^2$:

$$V_c(\alpha,\ \beta) = V_c(\beta,\ \alpha) \geq 0 \tag{3}$$

$$V_c(\alpha,\ \beta) = 0 \Leftrightarrow \alpha = \beta \tag{4}$$

A model is considered metric if it also satisfies the triangular inequality $V_c(\alpha,\ \beta) \leq V_c(\alpha,\ \gamma) + V_c(\gamma,\ \beta)$.

Thus, analyzing in particular each of the *a priori* models, we obtain: - Potts model: - metric, if we consider the distance $1 - \delta_{x_p=x_q}$: expressions 1 and 2 are satisfied by definition, and the triangular inequality is also satisfied; - alpha-beta swap and alpha- expansion optimization methods can be used; - (discrete) total variation: - metric: expressions 1 and 2 are satisfied by definition, and the

triangular inequality is also satisfied; - alpha-beta swap and alpha- expansion optimization methods can be used; - (quadratic) Gaussian model: - semi-metric: expressions 1 and 2 are satisfied by definition, but the triangular inequality is not satisfied; - only the alpha-beta swap optimization method can be used, as for the alpha-expansion optimization method the model must be metric.

Q4: What are the differences between the alpha expansion method and the alpha-beta swap method?

Your answer

A4:

The main differences between the alpha expansion method and the alpha-beta swap method are:

- Approach to the graph construction:
  - alpha-beta swap: the graph is built using only the pixels labeled alpha or beta as nodes
  - alpha-expansion: the graph is build with all the pixels
- Run strategy:
  - alpha-beta swap: draws a pair of labels iteratively
  - alpha-expansion: run through all labels
- Constraint for the potential energy of regularization
  - alpha-beta swap: semi-metric
  - alpha-expansion: metric

In the following sections we will use the functions aexpansion_grid and abswap_grid which perform the alpha-expansion and alpha-beta swap respectively. These functions take as input two arguments for a number of levels L : - a tensor of the image size containing in the 3rd dimension the data attachment of each pixel for each considered level (unary term) - a matrix containing the values of the interaction terms between two levels l_1 and l_2 (depends on the chosen interaction potential)

1.1.1 2.1 Denoising in the Gaussian case (synthetic image)

[ ]:
```
from maxflow.fastmin import aexpansion_grid, abswap_grid

# Loading images
im_obs = io.imread(
    "https://perso.telecom-paristech.fr/tupin/TPGRAPHCUT/OLD/Ibruitee.png"
).astype(
    "float"
)  # Observed image, noisy
im_obs2 = io.imread(
    "https://perso.telecom-paristech.fr/tupin/TPGRAPHCUT/OLD/Ibruitee2.png"
).astype(
    "float"
)  # Observed image, noisy
im_orig = io.imread(
    "https://perso.telecom-paristech.fr/tupin/TPGRAPHCUT/OLD/IoriginaleBW.png"
).astype(
    "float"
)  # Reference binary image, to evaluate the quality of the segmentation
```

```
I = im_obs

I = I * 255 / I.max()
L = 30
# Generates L gray levels for nearsest prototype labeling
levs = np.arange(0, 255, 255 / L)
# Calculate data cost as the absolute difference between the label prototype␣
 ↪and the pixel value
D = np.double(np.abs((I.reshape(I.shape + (1,)) - levs.reshape((1, 1, -1))) **␣
 ↪2))

affiche(I, MINI=0.0, MAXI=255.0, titre="Noisy image", printname=True)

# Generate nearest prototype labeling
Id = np.argmin(D, 2)
affiche(
    Id * 255 / L,
    MINI=0.0,
    MAXI=255.0,
    titre="Maximum likelihood denoising",
    printname=True,
)

print(D.shape)
```

```
(323, 361, 30)
```

Q5: Here, what does the array D correspond to? Explain its dimension.

Your answer

A5: The array D corresponds to the data attachment term. It contains $L = 30$ (number of gray levels) images of the same dimension as the noisy image (323×361).

Complete the programs below, and test each regularization model by determining an appropriate beta value each time.

[ ]:
```
# Potts model regularization
# beta values of several hundred
beta_Potts = 1800
# definition of the regularization matrix V(i,j) for the Potts model
V_Potts = np.double(
    beta_Potts * (np.abs(levs.reshape((-1, 1)) - levs.reshape((1, -1))) > 0)
)
affiche(V_Potts)

# Performs the alpha-expansion based on the data attachment D and the␣
 ↪regularization V
```

```
labels_Potts = aexpansion_grid(D, V_Potts)
affiche(
    labels_Potts * 255 / L,
    MINI=0.0,
    MAXI=255.0,
    titre="Graph Cut Denoising, Potts regularization, beta = " +␣
 ↪str(beta_Potts),
    printname=True,
)
```

[ ]:
```
# TV model regularization
# beta value of the order of tens
beta_TV = 50
# definition of the regularization matrix V(i,j) for the TV model
V_TV = np.double(beta_TV * (np.abs(levs.reshape((-1, 1)) - levs.reshape((1,␣
 ↪-1)))))
affiche(V_TV)

# Performs the alpha-expansion based on the data attachment D and the␣
 ↪regularization V
labels_TV = aexpansion_grid(D, V_TV)
print("Calcul TV terminé")
affiche(
    labels_TV * 255 / L,
    MINI=0.0,
    MAXI=255.0,
    titre="Graph Cut Denoising, TV regularization, beta = " + str(beta_TV),
    printname=True,
)
```

```
Calcul TV terminé
```

[ ]:
```
# Quadratic model regularization
# beta value of the order of tens
beta_quadratic = 3
# definition of the regularization matrix V(i,j) for the quadratic model
V_quadratic = np.double(
    beta_quadratic * (np.abs(levs.reshape((-1, 1)) - levs.reshape((1, -1)))) **␣
 ↪2
)
affiche(V_quadratic)

# Performs the alpha-beta swap based on the data attachment D and the␣
 ↪regularization V
labels_Quadratic = aexpansion_grid(D, V_quadratic)
affiche(
    labels_Quadratic * 255 / L,
```

```
    MINI=0.0,
    MAXI=255,
    titre="Graph Cut Denoising, TV regularization, beta = " +␣
 ↪str(beta_quadratic),
    printname=True,
)
```

Q6: Which regularization model do you think is best? Give the best regularization parameter visually found and comment on the results you get in each of the three cases.

Your answer

A6:

For the studied image, TV regularization quickly yields a better result. Potts model regularization also yields a comparable result, albeit a bit more slowly and less clear. The quadratic model regularization eliminate very well the noise, however it yields an even less accurate image and the calculations are quite extensive. The best regularization parameter obtained for the models are: 1800, 50 and 3, respectively.

1.1.2 2.2 Denoising in the case of speckle noise (synthetic image)

Modify the following cells to fit the model for denoising the im_obs2 image.

Q7: Which modifications are needed? (There are several!)

Your answer

A7:

It is necessary to change the calculation of the data attachment term (D).

[ ]:
```
from maxflow.fastmin import aexpansion_grid
from numpy import log

# Loading image
im_obs2 = io.imread(
    "https://perso.telecom-paristech.fr/tupin/TPGRAPHCUT/OLD/Ibruitee2.png"
).astype(
    "float"
)  # Observed image, noisy
I = im_obs2
I = I * 255 / I.max()
affiche(I, MINI=0.0, MAXI=255.0, titre="Noisy image 2", printname=True)

L = 30
# Generates L gray levels for nearsest prototype labeling
levs = np.arange(1 / L, 255, 255 / L)

# Calculate data cost as the absolute difference between the label prototype␣
 ↪and the pixel value
```

```
D = -log(2 * I.reshape(I.shape + (1,)) / levs.reshape((1, 1, -1)) ** 2) + (
    I.reshape(I.shape + (1,)) ** 2 / levs.reshape((1, 1, -1)) ** 2
)
print()
Id = np.argmin(D, 2)

# Generate nearest prototype labeling
Id = np.argmin(D, 2)
affiche(
    Id / L * 255,
    MINI=0.0,
    MAXI=255.0,
    titre="Maximum likelihood denoising",
    printname=True,
)
```

```
C:\Users\willf\AppData\Local\Temp\ipykernel_24476\1807008655.py:19:
RuntimeWarning: divide by zero encountered in log
  D = -log(2 * I.reshape(I.shape + (1,)) / levs.reshape((1, 1, -1)) ** 2) + (
```

[ ]:
```
# beta value in the order of tenths of a unit
beta_Potts = 0.5
# definition of the Potts potential matrix
V_Potts = np.double(
    beta_Potts * (np.abs(levs.reshape((-1, 1)) - levs.reshape((1, -1))) > 0)
)

# Performs the alpha-expansion based on the data attachment D and the␣
 ↪regularization V
labels_Potts = aexpansion_grid(D, V_Potts)
affiche(
    labels_Potts * 255 / L,
    MINI=0.0,
    MAXI=255,
    titre="Graph Cut Denoising, Potts regularization, beta = " +␣
 ↪str(beta_Potts),
    printname=True,
)
```

[ ]:
```
# beta value in hundredths of a unit
beta_TV = 0.02
# definition of the regularization matrix for TV
V_TV = np.double(beta_TV * np.abs(levs.reshape((-1, 1)) - levs.reshape((1,␣
 ↪-1))))
```

```
# Performs the alpha-expansion based on the data attachment D and the␣
 ↪regularization V
labels_TV = aexpansion_grid(D, V_TV)
print("TV computation completed")
affiche(
    labels_TV * 255 / L,
    MINI=0.0,
    MAXI=255,
    titre="Graph Cut Denoising, TV regularization, beta = " + str(beta_TV),
    printname=True,
)
```

```
TV computation completed
```

[ ]:
```
# beta value in the order of thousandths of a unit
beta_quadratic = 0.001
# definition of the regularization matrix for a quadratic potential
V_quadratic = np.double(
    beta_quadratic * (np.abs(levs.reshape((-1, 1)) - levs.reshape((1, -1)))) **␣
 ↪2
)


# Performs the alpha-expansion based on the data attachment D and the␣
 ↪regularization V
labels_Quadratic = aexpansion_grid(D, V_quadratic)
affiche(
    labels_Quadratic * 255 / L,
    MINI=0.0,
    MAXI=255.0,
    titre="Graph Cut Denoising, Quadratic regularization, beta = "
    + str(beta_quadratic),
    printname=True,
)
```

1.1.3 2.3 Denoising a natural image

Apply one of the methods used above to denoise the noisy cameraman image. Justify your choice.

Q8: Comment on the result obtained.

Your answer

A8:

As the noisy cameraman image is not piecewise, just as before, there seems to be a greater interest in the use of TV regularization. In particular as TV is metric, it is possible to use alpha-expansion.

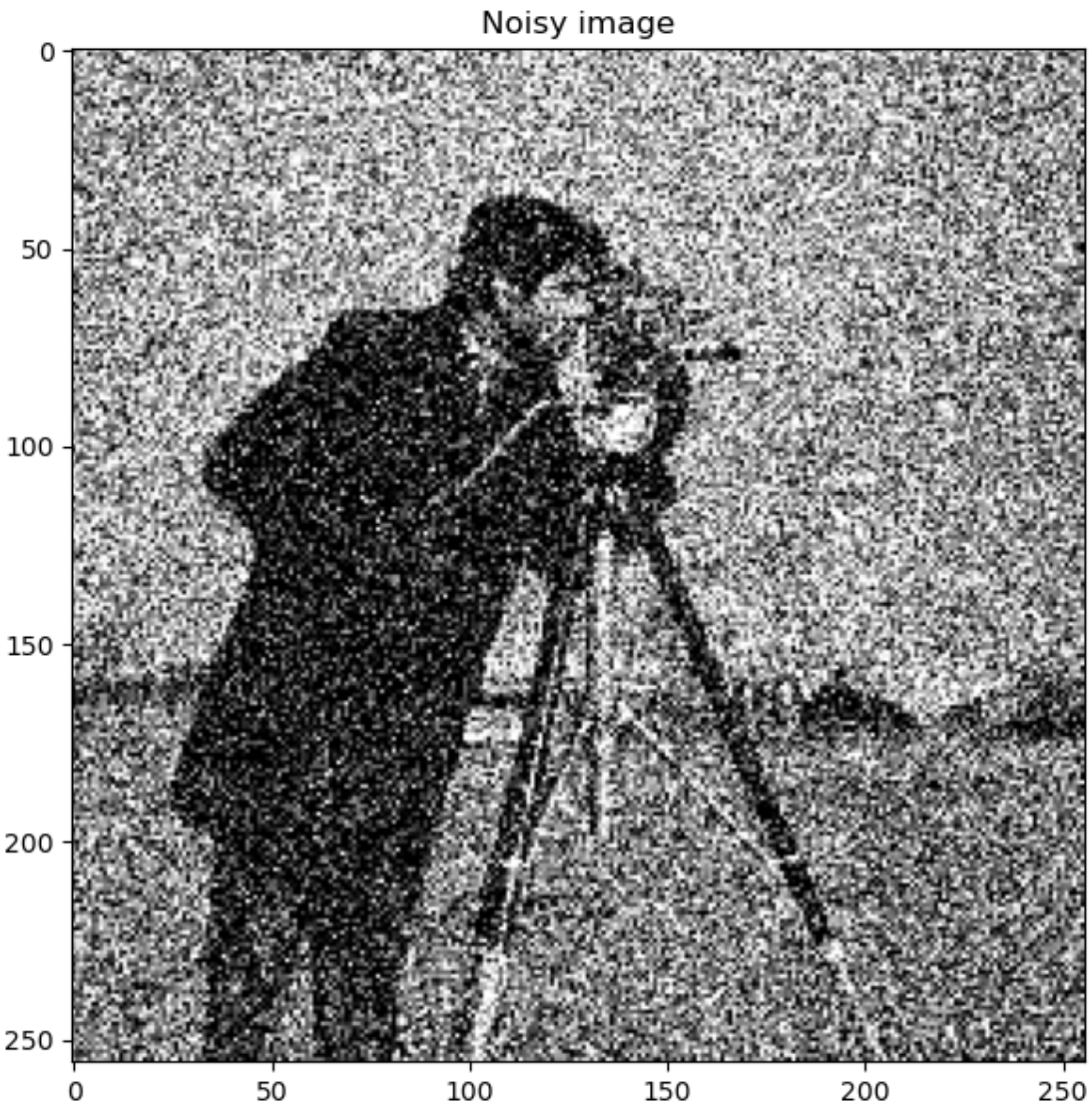
For the chosen beta, the obtained result is acceptable. It is possible to recover the outline of the objects on the foreground of the image (cameraman, camera), and a blurred outline of those on

the background of the image (buildings). In spite of that, it is important to highlight that, as the noise intensity is strong, to be able to recover important (foreground) information from the image, is already a relatively good result.

[ ]:
```
from skimage.color import rgb2gray
import imageio.v2 as imageio

I_cameraman = imageio.imread(
    "https://people.math.sc.edu/Burkardt/data/tif/cameraman.tif"
)
I_cameraman_bruit = I_cameraman + np.random.normal(0, 80, I_cameraman.shape)
I_cameraman_bruit[I_cameraman_bruit < 0] = 0
I_cameraman_bruit[I_cameraman_bruit > 255] = 255
print(I_cameraman.max())

plt.figure(figsize=(7, 7))
plt.imshow(I_cameraman, cmap="gray")
plt.title("Image without noise")
plt.show()

plt.figure(figsize=(7, 7))
plt.imshow(I_cameraman_bruit, cmap="gray")
plt.title("Noisy image")
plt.show()
```

```
253
```


Image without noise
0
50
100
150
200
250
0
50
100
150
200
250



[ ]:
```
I = 255 * I_cameraman_bruit / I.max()
L = 250
# Generates L gray levels for nearsest prototype labeling
levs = np.arange(0, 255, 255 / L)
# Calculate data cost as the absolute difference between the label prototype␣
 ↪and the pixel value
D = np.double(np.abs((I.reshape(I.shape + (1,)) - levs.reshape((1, 1, -1))) **␣
 ↪2))


# choose a regularization model and compute V matrix
Id = np.argmin(D, 2)
```

```
affiche(
    Id,
    MINI=0.0,
    MAXI=None,
    titre="Maximum likelihood denoising",
    printname=True,
)
beta_TV = 120
V_TV = np.double(beta_TV * (np.abs(levs.reshape((-1, 1)) - levs.reshape((1,␣
 ↪-1)))))

# compute the appropriate optimization
labels_TV = aexpansion_grid(D, V_TV)
# display the regularized image
affiche(
    labels_TV,
    MINI=0.0,
    MAXI=None,
    titre="Graph Cut Denoising, TV regularization, beta = " + str(beta_TV),
    printname=True,
)
```

1.1.4 2.3 SAR Image Denoising

SAR (Synthetic Aperture Radar) imagery is a radar-based remote sensing modality that provides images of the Earth in all light and weather conditions. A major drawback is the high speckle noise that affects them. The following cell loads an amplitude image acquired by the Sentinel 1-A satellite over the city of Des Moines in the USA. To limit the computation time, we will work on a small rectangle from the image provided.

Adapt one of the methods used previously to denoise the image provided. We will assume that the noise follows a Rayleigh distribution.

Q9: Comment on the result obtained.

We can compare the result with a denoising obtained by a more recent method (SAR2SAR), based on a Deep Learning approach (the code to display it is provided below).

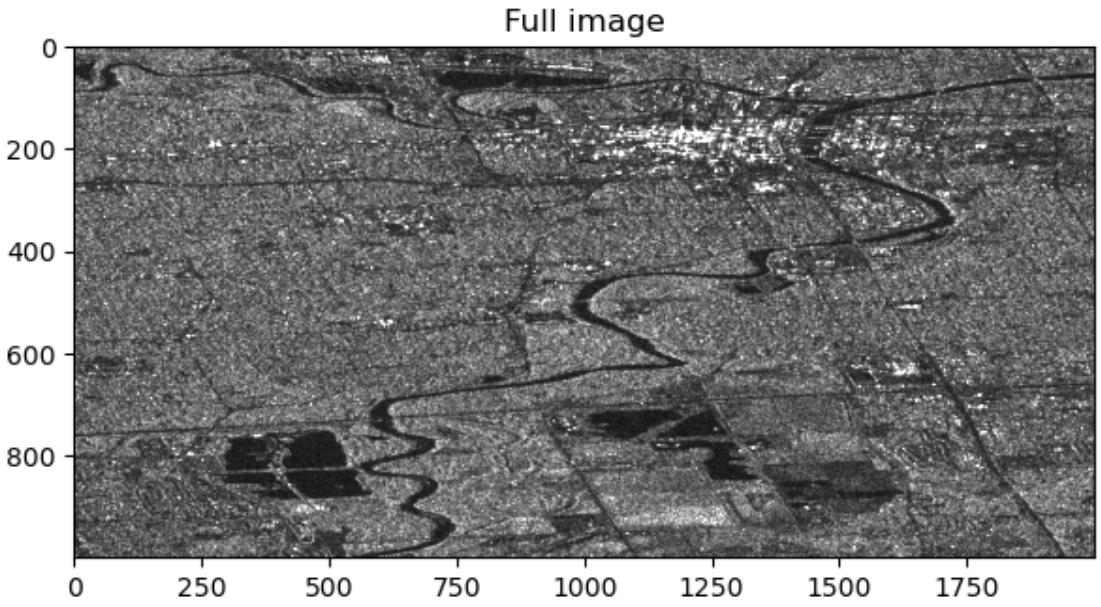
Your answer

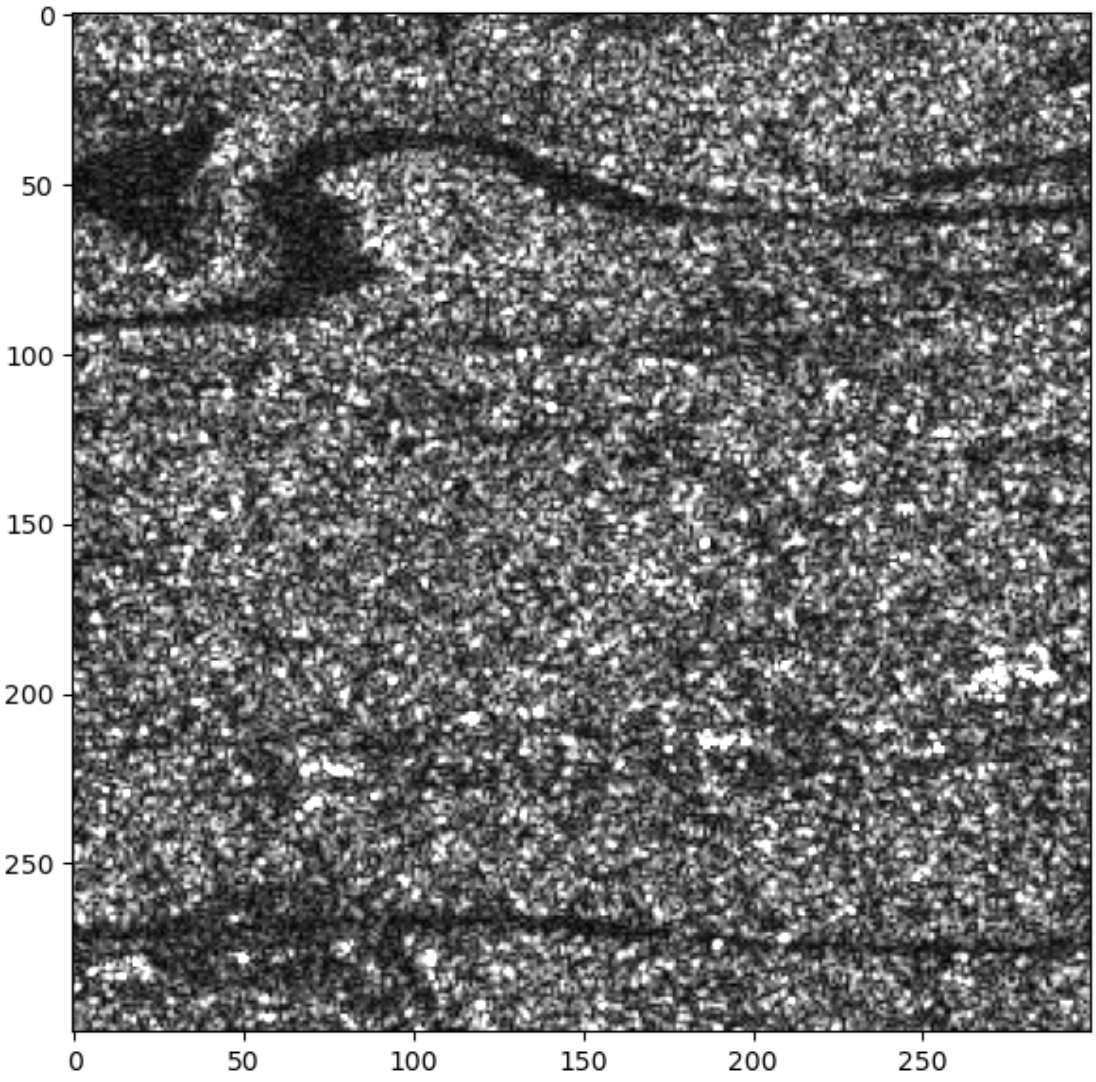
A9:

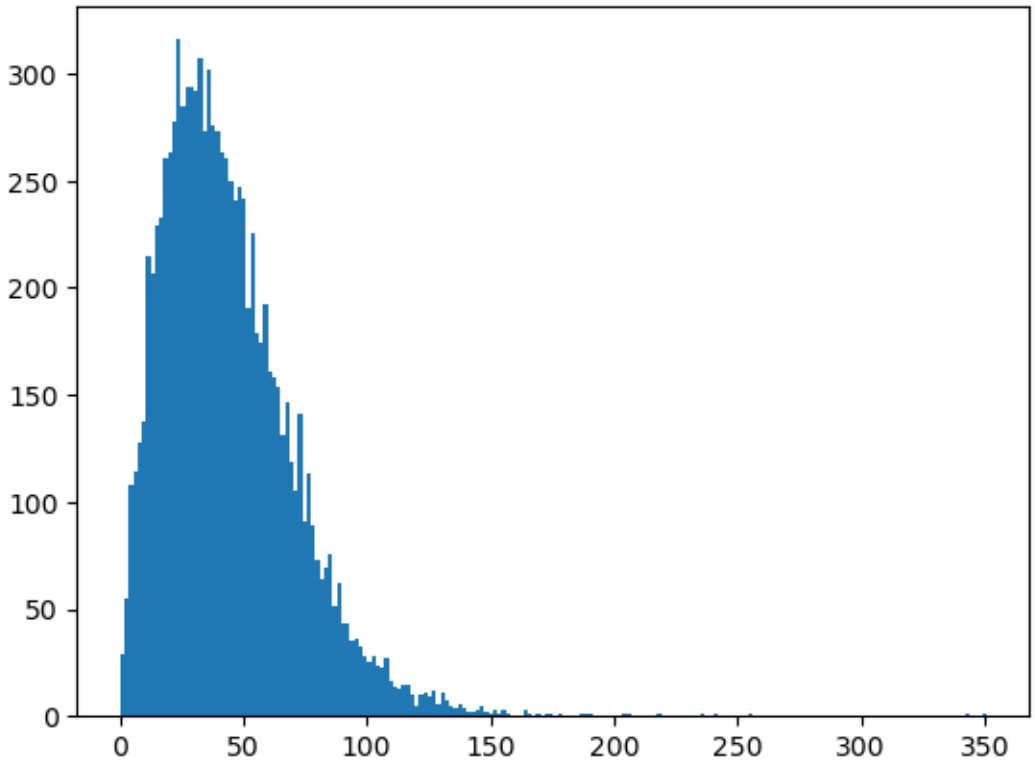
The obtained result is acceptable. It is possible to denoise the image and recover an approximate outline of the objects on the image (the river). In spite of that, it is important to highlight that, as the noise intensity is strong, to be able to recover important (foreground) information from the image, is already a relatively good result. The result yielded by SAR2SAR is generally superior.

[ ]:
```
try:
    I_SAR = np.load("noisy_DesMoines_dual_1_corrige_1_VV_AMPLITUDE.npy")
except:
```

```
    !wget "https://www.dropbox.com/s/7m2dw3irho8dpzj/
 ↪noisy_DesMoines_dual_1_corrige_1_VV_AMPLITUDE.npy?dl=1"
    I_SAR = np.load("noisy_DesMoines_dual_1_corrige_1_VV_AMPLITUDE.npy")

# affiche_(I_SAR, 0, 100, "Full image")
plt.figure(figsize=(7, 7))
plt.imshow(
    I_SAR, vmax=100, cmap="gray"
)
plt.title("Full image")
plt.show()

I_SAR = I_SAR[0:300, 0:300]
plt.figure(figsize=(7, 7))
plt.imshow(
    I_SAR, vmax=100, cmap="gray"
)  # The display is done by truncating the dynamic
plt.show()


plt.figure()
plt.hist(
    I_SAR[100:200, 100:200].flatten(), 200
)  # Display of the histogram on an almost homogeneous area
plt.show()
```

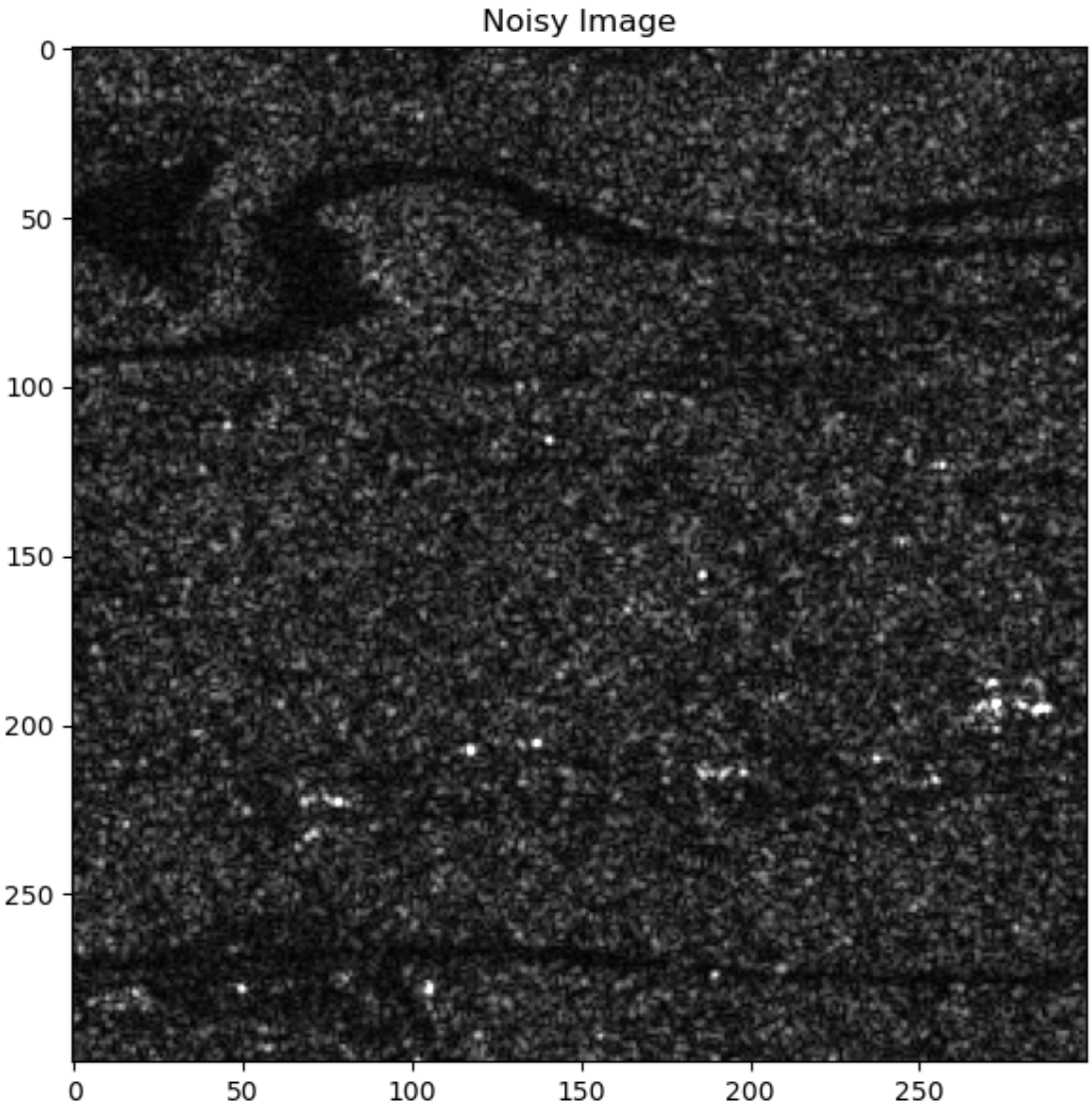


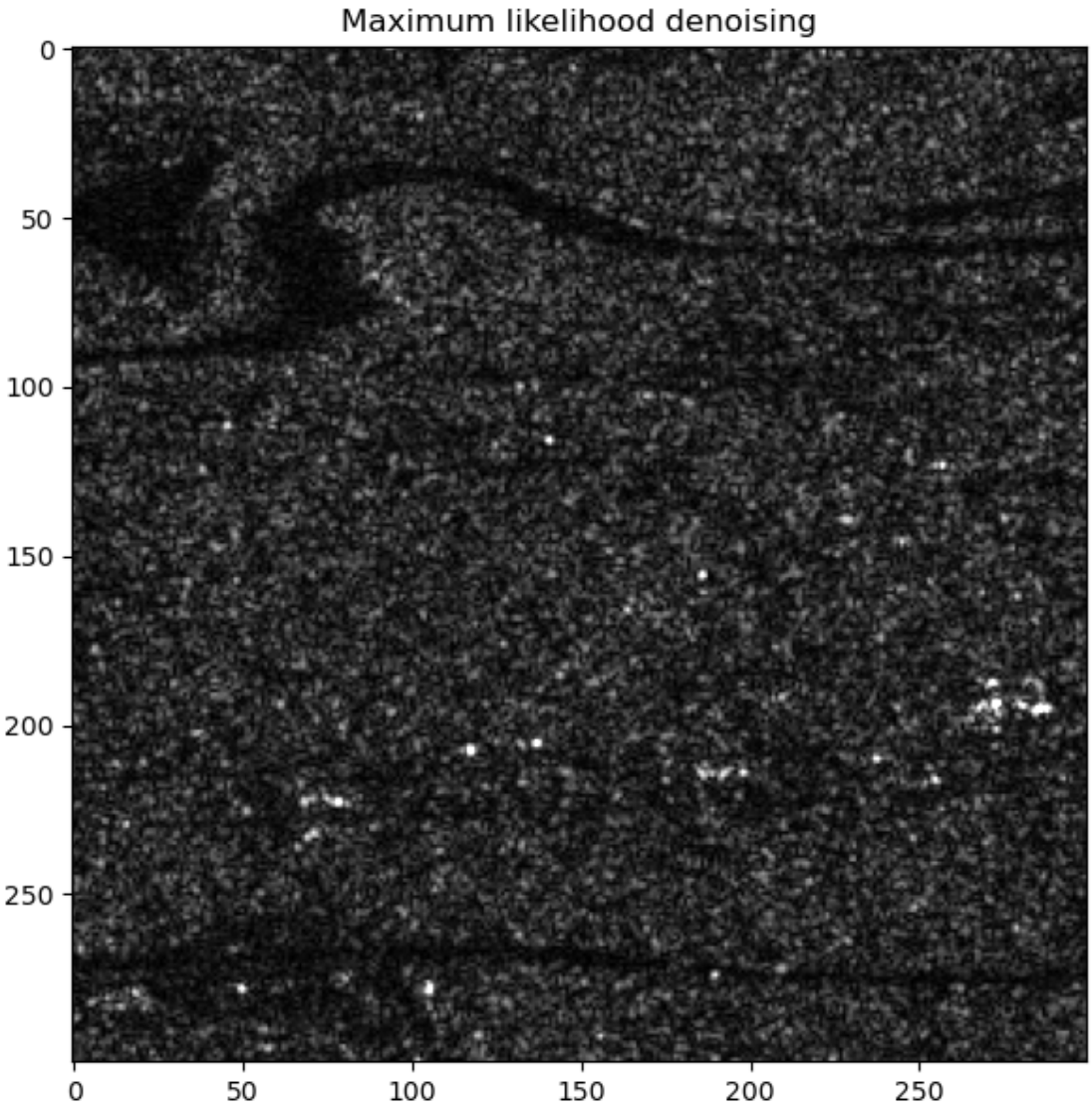

0
50
100
150
200
250
0
50
100
150
200
250

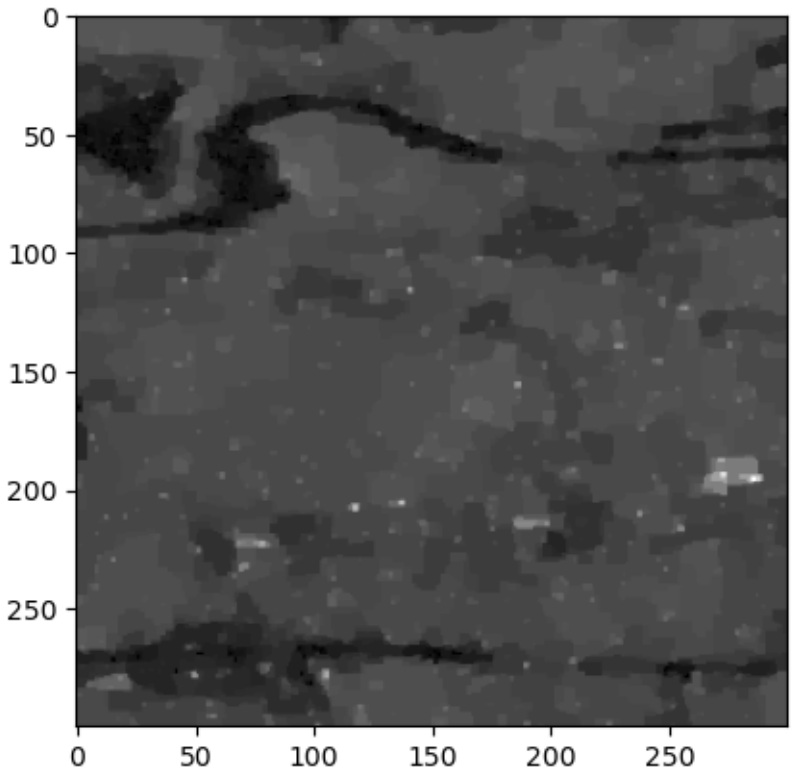


[ ]:
```
I = I_SAR * 255 / I_SAR.max()
L = 255
# Generates L gray levels for nearsest prototype labeling
levs = np.arange(1 / L, 255, 255 / L)
# Calculate data cost as the neg-log likelihood
D = -log(2 * I.reshape(I.shape + (1,)) / levs.reshape((1, 1, -1)) ** 2) + (
    I.reshape(I.shape + (1,)) ** 2 / levs.reshape((1, 1, -1)) ** 2
)
print()
# affiche(I, MINI=0.0, MAXI=100, titre="Noisy image", printname=True)
plt.figure(figsize=(7, 7))
plt.imshow(I, vmax=100, cmap="gray")
plt.title("Noisy Image")
plt.show()


# choose a regularization model and compute V matrix
Id = np.argmin(D, 2)
# affiche(
#     Id,
#     MINI=0.0,
#     MAXI=None,
```

```
#     titre="Maximum likelihood denoising",
#     printname=True,
# )
plt.figure(figsize=(7, 7))
plt.imshow(Id, vmax=100, cmap="gray")
plt.title("Maximum likelihood denoising")
plt.show()

beta_TV = 0.15
V_TV = np.double(beta_TV * (np.abs(levs.reshape((-1, 1)) - levs.reshape((1,␣
 ↪-1)))))

# compute the appropriate optimization
labels_TV = aexpansion_grid(D, V_TV)

# display the regularized image
affiche(
    labels_TV,
    MINI=0.0,
    MAXI=None,
    titre="Graph Cut Denoising, TV regularization, beta = " + str(beta_TV),
    printname=True,
)
plt.figure()
plt.imshow(labels_TV, cmap="gray")
plt.show()
```

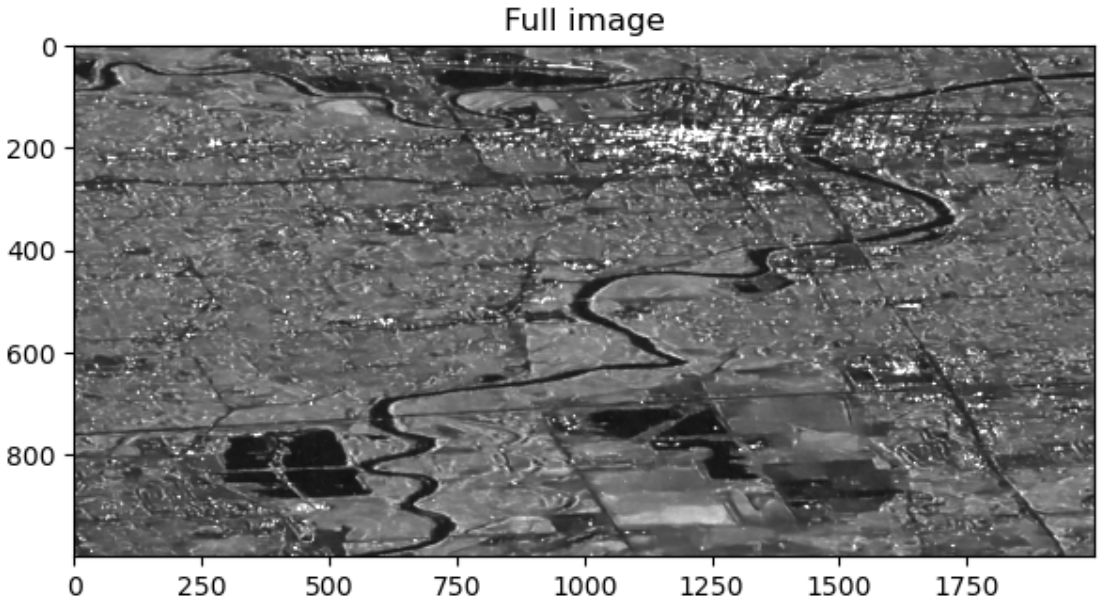

Noisy Image
0
50
100
150
200
250
0
50
100
150
200
250

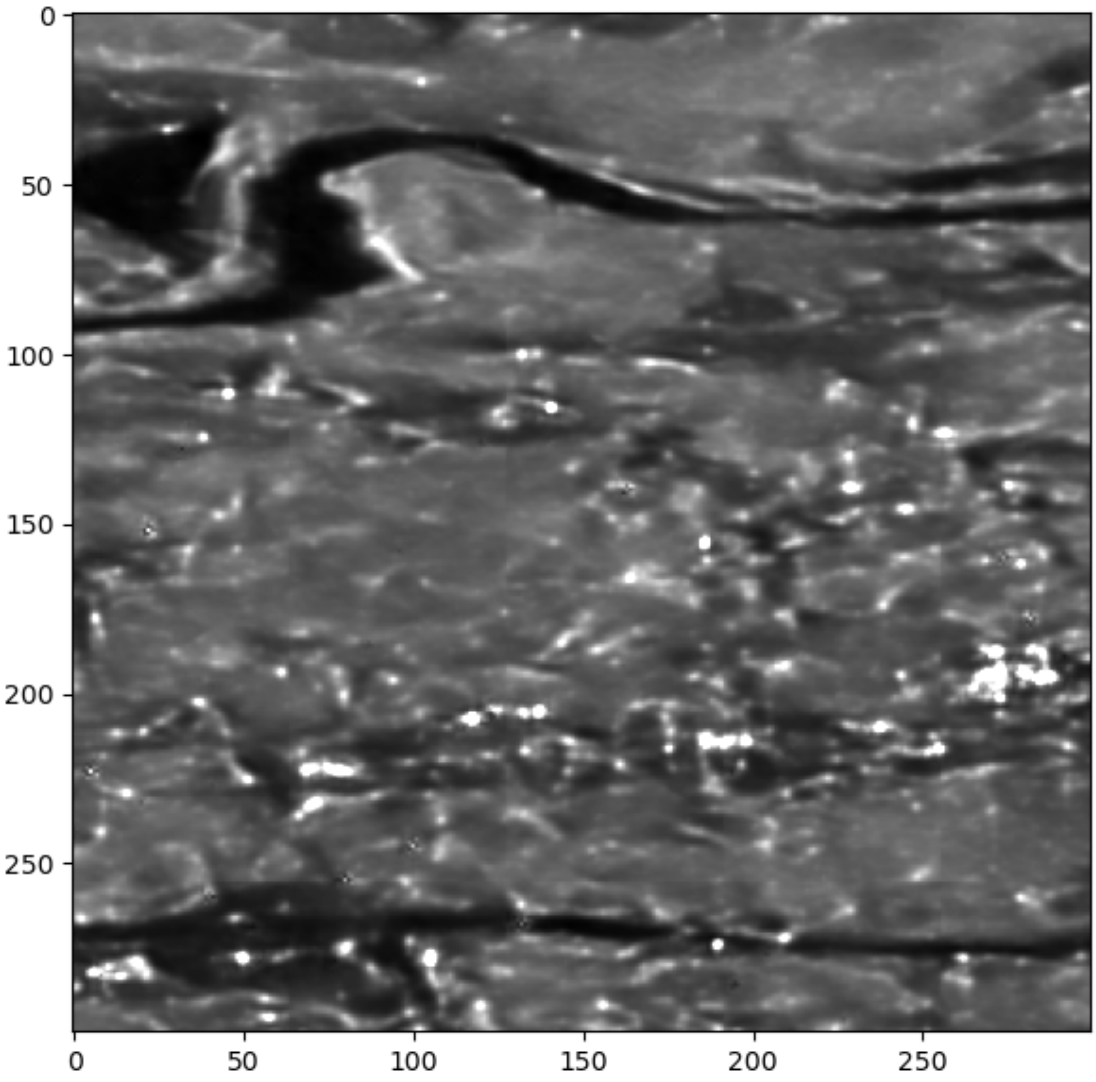

Maximum likelihood denoising
0
50
100
150
200
250
0
50
100
150
200
250

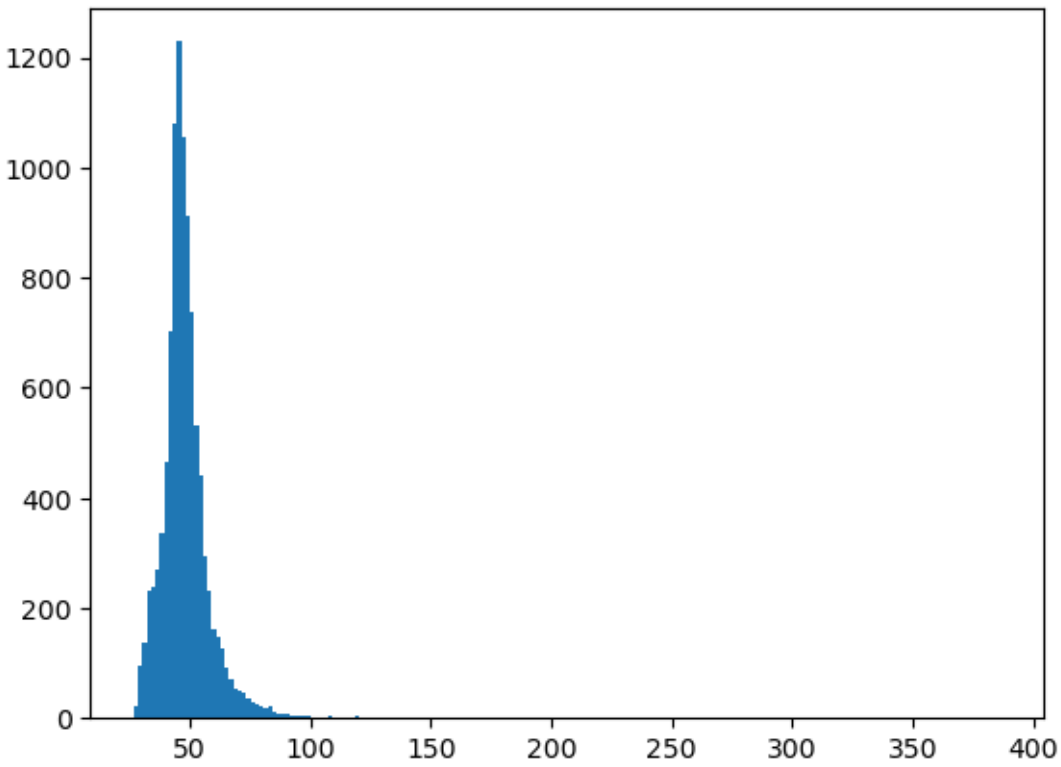


[ ]:
```
# Display of the denoised image by the SAR2SAR method

try:
    I_SAR = np.load("denoised_DesMoines_dual_1_corrige_1_VV_AMPLITUDE.npy")
except:
    !wget "https://www.dropbox.com/s/0f6l0qr6teck5bd/
 ↪denoised_DesMoines_dual_1_corrige_1_VV_AMPLITUDE.npy?dl=1"
    I_SAR = np.load("denoised_DesMoines_dual_1_corrige_1_VV_AMPLITUDE.npy")

# affiche(I_SAR, 0, 100, "Full image")
plt.figure(figsize=(7, 7))
plt.imshow(
    I_SAR, vmax=100, cmap="gray"
)
plt.title("Full image")
plt.show()

I_SAR = I_SAR[0:300, 0:300]
plt.figure(figsize=(7, 7))
plt.imshow(
    I_SAR, vmax=100, cmap="gray"
```

```
)  # The display is done by truncating the dynamic
plt.show()

plt.figure()
plt.hist(
    I_SAR[100:200, 100:200].flatten(), 200
)  # Display of the histogram on an almost homogeneous area
plt.show()
```




0
50
100
150
200
250
0
50
100
150
200
250


1200
1000
800
600
400
200
0
50
100
150
200
250
300
350
400