TP_seg_MEDIMA_204_STUDENTS_2025_abdennour_kerboua

February 4, 2025

1 IMA204 Practical Session - Segmentation of medical images

1.1 ADD your name(s) here:

Abdennour Kerboua (worked alone)

2 Introduction

Images to use are provided in a zip file called **data.zip** to install in your local directory from where you are running this notebook.

The goal of this practical session is to push K-means segmentation method as a pre-segmentation tool on different types of medical images: CT scans for the segmentation of kidneys and tumors, MRI for the corpus callosum in the brain , temporal sequences of MRI images for segmentation of the myocardium.

You are provided with pre-processing ideas. You will have to adjust a pipeline for **AT LEAST** one application (kidney/tumor, corpus callosum or heart).

You have to submit your code and comment your results.

Deadline: You will have to upload a single jupyter notebook .ipynb with your answers (code + text) before the deadline (please check on Ecampus/Moodle).

The uploaded file should be named 'TP_SegMedImage_YOURSURNAME.ipynb'.

RUN THE WHOLE NOTEBOOK FIRST TO GET A FULL OVERVIEW OF USE-CASES AND THE NOTION OF HYPERPARAMETERS.

[2]:
```python
import os
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap, LinearSegmentedColormap

# For ploting utilities of contours on images
from mpl_toolkits.axes_grid1 import AxesGrid
from mpl_toolkits.axes_grid1 import make_axes_locatable

# For listing files in local foler
import glob
```

```python
# For loading .mat files
import scipy
from scipy.io import loadmat

import skimage
from skimage.io import imread
from skimage import morphology
from skimage.segmentation import watershed
from skimage.filters import rank
from skimage.util import img_as_ubyte
from skimage.morphology import disk
from scipy import ndimage
from skimage.measure import find_contours
import skimage.morphology as morpho
from skimage import data
from skimage import color
from skimage import morphology
from skimage import segmentation
from skimage.filters import gaussian


# For Kmeans
import cv2


# PRINT VERSIONS
#print("os.__version__",os.__version__)
print("np.__version__",np.__version__)
print("matplotlib.__version__",matplotlib.__version__)
print("skimage.__version__",skimage.__version__)
print("scipy.__version__",scipy.__version__)
print("cv2.__version__",cv2.__version__)
#print("glob.__version__",glob.__version__)
```

```
np.__version__ 1.26.4
matplotlib.__version__ 3.9.2
skimage.__version__ 0.24.0
scipy.__version__ 1.14.1
cv2.__version__ 4.11.0
```

[3]:
```python
def my_kmeans(image,k):
    #k = number of clusters

    # Reshaping the image
    pixel_vals = image.reshape((-1,1))
    # Convert to float type only for supporting cv2.kmean
    pixel_vals = np.float32(pixel_vals)
    criteria   = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.
↪85) #criteria
    retval, labels, centers = cv2.kmeans(pixel_vals, k,
                None, criteria, 10, cv2.KMEANS_PP_CENTERS)
    centers = np.uint8(centers) # convert data into 8-bit values

    segmented_data   = centers[labels.flatten()] # Mapping labels to center␣
↪points( RGB Value)
    segmented_image  = segmented_data.reshape((image.shape)) # reshape data␣
↪into the original image dimensions

    segmented_labels = labels # Mapping labels to center points( RGB Value)
    segmented_labels = segmented_labels.reshape((image.shape)) # reshape data␣
↪into the original image dimensions

    return segmented_image,segmented_labels


def my_colormap_white_bkg(Colormap_name,numLabels):
    #numLabels = number of colors

    Colormap  = plt.get_cmap(Colormap_name, numLabels)
    newcolors = Colormap(np.linspace(0, 1, numLabels))
    bkg_color = np.array([256/256, 256/256, 256/256, 1])
    max_color = np.array([0/256, 0/256, 0/256, 1])
    newcolors[numLabels-1, :] = max_color
    newcolors[0, :] = bkg_color
    newcmp = ListedColormap(newcolors)
    return newcmp
```

2.1 Abdominal CT

You have at your disposal **6 abdominal CT scans** of different subjects. Subjects may have renal tumor. You also have the manual segmentations for both kidney and, when present, the tumor.

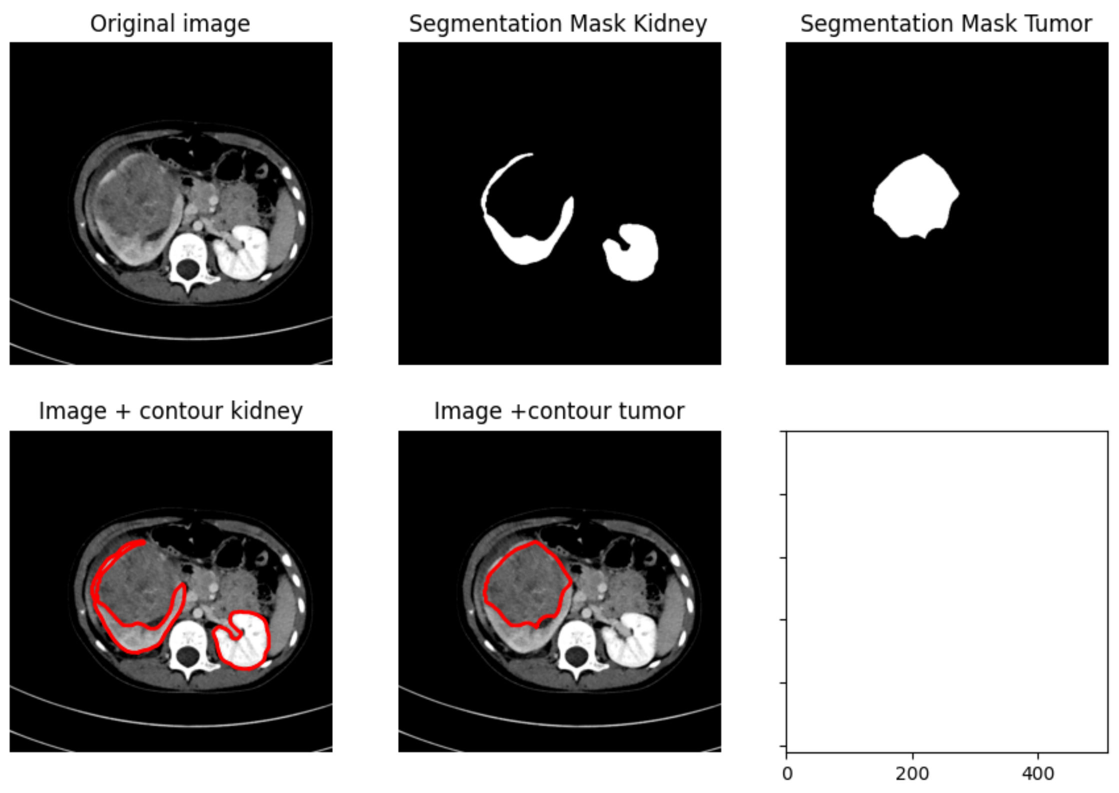
Variables defined: * abdominalCT_path * listImagesabdCT * Img_abdo_ex * Seg_abdo_kidney_ex, Seg_abdo_tumor_ex

[4]:
```
abdominalCT_path = './data/abdominalCT_axial'
os.listdir(abdominalCT_path)
listImagesabdCT=glob.glob(abdominalCT_path + '/*-seg.tiff')
print('There are', len(listImagesabdCT),  'abdomical CT images')

# Choose a figure and plot it with the ground truth segmentation
indexIm=3 # between 0 and 5
```

```
# Abdominal CT
filename_Segmentation = listImagesabdCT[indexIm]
Labels_abdo_ex    = imread(filename_Segmentation)
filename          = filename_Segmentation[:-9] + '.tiff'
Img_abdo_ex       = imread(filename)

print('Reading image ', filename)
print('Label values in gt mask image', np.unique(Labels_abdo_ex))

if Img_abdo_ex.shape != Labels_abdo_ex.shape:
  raise NameError('image and mask should have the same shape, problem...')

# In Labels_abdo_ex we may have two values: 127 is for kidney and 255 for renal␣
  ↪tumor
Seg_abdo_kidney_ex=Labels_abdo_ex==127
if np.sum(Seg_abdo_kidney_ex)==0:
  print('There is no kidney')
Cont_abdo_kidney_ex = find_contours(Seg_abdo_kidney_ex, 0.5)

Seg_abdo_tumor_ex=Labels_abdo_ex==255
if np.sum(Seg_abdo_tumor_ex)==0:
  print('There is no tumor')
Cont_abdo_tumor_ex = find_contours(Seg_abdo_tumor_ex, 0.5)

fig = plt.figure(figsize=(17, 7))
grid = AxesGrid(fig, 111,
                nrows_ncols = (2, 3),
                axes_pad = 0.5)
grid[0].imshow(Img_abdo_ex, cmap='gray')
grid[0].axis('off')
grid[0].set_title("Original image")
grid[1].imshow(Seg_abdo_kidney_ex,cmap='gray')
grid[1].axis('off')
grid[1].set_title("Segmentation Mask Kidney")
grid[2].imshow(Seg_abdo_tumor_ex,cmap='gray')
grid[2].axis('off')
grid[2].set_title("Segmentation Mask Tumor")
grid[3].imshow(Img_abdo_ex, cmap='gray')
for contour in Cont_abdo_kidney_ex:
  grid[3].plot(contour[:, 1], contour[:, 0], linewidth=2, c='r')
grid[3].axis('off')
grid[3].set_title("Image + contour kidney")
grid[4].imshow(Img_abdo_ex, cmap='gray')
for contour in Cont_abdo_tumor_ex:
  grid[4].plot(contour[:, 1], contour[:, 0], linewidth=2, c='r')
grid[4].axis('off')
grid[4].set_title("Image +contour tumor")
```

```
There are 6 abdomical CT images
Reading image  ./data/abdominalCT_axial/CTs3.tiff
Label values in gt mask image [  0 127 255]
```

[4]:
```
Text(0.5, 1.0, 'Image +contour tumor')
```



3 Brain MRI

Here you can select medial slices of the brain of **4 different subjects**. You also have manual segmentations of the corpus callosum.

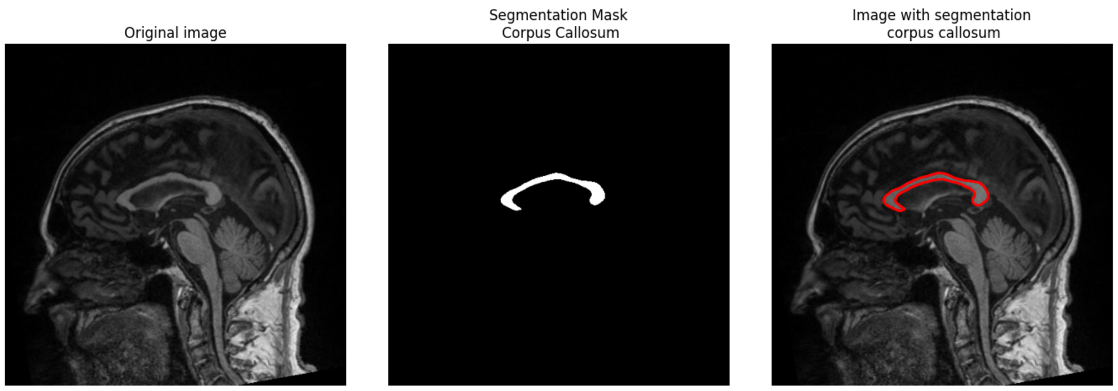
Variables defined: * brainMRI_path * listImagesbrainMRI * Seg_brain_ex * Img_brain_ex

[5]:
```python
brainMRI_path = './data/brainMRI'
os.listdir(brainMRI_path)
listImagesbrainMRI=glob.glob(brainMRI_path + '/*-seg.png')
print('There are', len(listImagesbrainMRI),  'brain MRI images')
print(listImagesbrainMRI)

# Choose a brain MRI and plot it with the ground truth segmentation
indexIm       = 3 # between 0 and 3
filename_seg = listImagesbrainMRI[indexIm]
```

```
Seg_brain_ex = imread(filename_seg)
filename     = filename_seg[:-8] + '.png'
Img_brain_ex = imread(filename)

print('Reading image ', filename)

if Img_brain_ex.shape != Seg_brain_ex.shape:
  raise NameError('image and mask should have the same shape, problem...')

# In Im Seg we have masks of the corpus callosum
maskCC      = Seg_brain_ex==255
contourMask = find_contours(maskCC, 0.5)

fig = plt.figure(figsize=(17, 7))
grid = AxesGrid(fig, 111,
                nrows_ncols = (1, 3),
                axes_pad = 0.5)
grid[0].imshow(Img_brain_ex, cmap='gray')
grid[0].axis('off')
grid[0].set_title("Original image")
grid[1].imshow(maskCC,cmap='gray')
grid[1].axis('off')
grid[1].set_title("Segmentation Mask\n Corpus Callosum");
grid[2].imshow(Img_brain_ex, cmap='gray')
for contour in contourMask:
  grid[2].plot(contour[:, 1], contour[:, 0], linewidth=2, c='r')
grid[2].axis('off')
grid[2].set_title("Image with segmentation\n corpus callosum")
```

```
There are 4 brain MRI images
['./data/brainMRI/MRIs1-seg.png', './data/brainMRI/MRIs2-seg.png',
'./data/brainMRI/MRIs3-seg.png', './data/brainMRI/MRIs4-seg.png']
Reading image  ./data/brainMRI/MRIs4.png
```

```
[5]: Text(0.5, 1.0, 'Image with segmentation\n corpus callosum')
```



4 Cardiac MRI

The last section is about MRI sequences of the heart. You are provided with **a single use case**. Your goal is to segment the left ventricule. Be careful, the segmentation is not a maks but a series of points (landmarks). To obtain a binary mask, you should first interpolate the points (using for instance a spline).

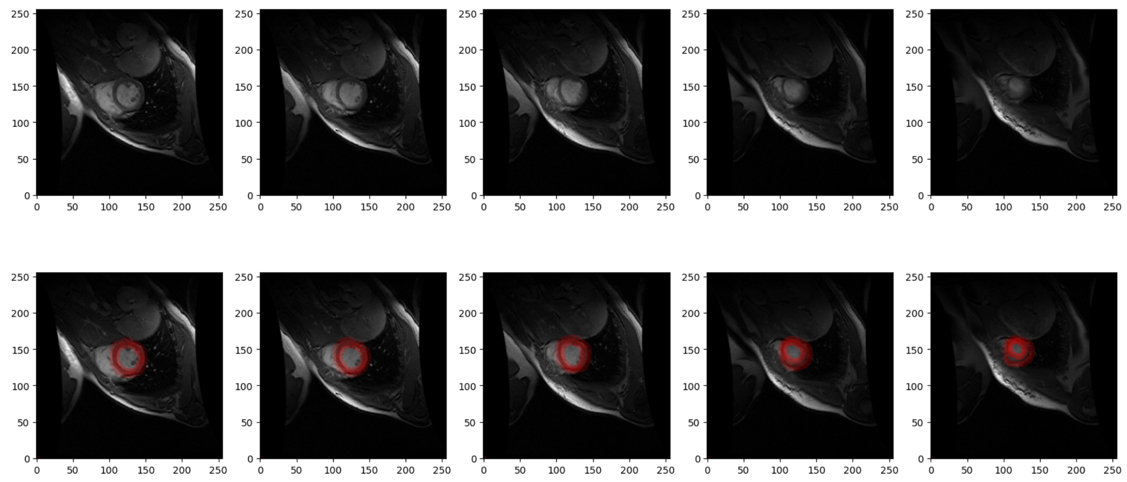
The structure of this image data is more complex and needs some coding to load single slices to segment.

Variables defined: * MRIheart_path * Img_cardiac_ex * Seg_cardiac_ex

[6]:
```
# Read one case
MRIheart_path = './data/MRIheart/'
os.listdir(MRIheart_path)
data          = loadmat(MRIheart_path + 'dataMRIheart.mat')
data          = data['data']
seg           = loadmat(MRIheart_path + 'segMRIheart.mat')
seg           = seg['seg']

Ex_index_select = 6
Img_cardiac_ex = data[:,:,Ex_index_select,1] # can be index 4,5,6,...
Cont_cardiac_ex = seg[Ex_index_select,1][:]

print('MRI volume of the heart composed of', data.shape[2], 'slices along the z␣
 ↪axis and', data.shape[3],
'temporal frames. Each slice is an image ', data.shape[0], ' x ',  data.
 ↪shape[1])
print('For each slice and at each time frame we have a manual segmentation␣
 ↪composed of',seg[4,4].shape[0] , '2D landmarks')

print('Be careful, some slices do not contain the left ventricle myocardium and␣
 ↪the manual segmentation is not simply empty but it contains the value:',␣
 ↪seg[0,0] )

plt.figure(figsize=(20, 9))
plt.suptitle('Sample slices at a single time frame with the red manual␣
 ↪segmentation at the bottom')
plt.subplot(2, 5, 1)
plt.imshow(data[:,:,4,1],cmap="gray")
plt.gca().invert_yaxis()

plt.subplot(2, 5, 2)
plt.imshow(data[:,:,5,1],cmap="gray")
plt.gca().invert_yaxis()
plt.subplot(2, 5, 3)
plt.imshow(data[:,:,6,1],cmap="gray")
plt.gca().invert_yaxis()

plt.subplot(2, 5, 4)
plt.imshow(data[:,:,7,1],cmap="gray")
plt.gca().invert_yaxis()

plt.subplot(2, 5, 5)
plt.imshow(data[:,:,8,1],cmap="gray")
plt.gca().invert_yaxis()

plt.subplot(2, 5, 6)
plt.imshow(data[:,:,4,1],cmap="gray")
plt.gca().invert_yaxis()
plt.scatter(seg[4,1][:,0], seg[4,1][:,1], c='r',alpha=0.1)

plt.subplot(2, 5, 7)
plt.imshow(data[:,:,5,1],cmap="gray")
plt.gca().invert_yaxis()
plt.scatter(seg[5,1][:,0], seg[5,1][:,1], c='r',alpha=0.1)

plt.subplot(2, 5, 8)
plt.imshow(data[:,:,6,1],cmap="gray")
plt.gca().invert_yaxis()
plt.scatter(seg[6,1][:,0], seg[6,1][:,1], c='r',alpha=0.1)

plt.subplot(2, 5, 9)
plt.imshow(data[:,:,7,1],cmap="gray")
plt.gca().invert_yaxis()
plt.scatter(seg[7,1][:,0], seg[7,1][:,1], c='r',alpha=0.1)

plt.subplot(2, 5, 10)
plt.imshow(data[:,:,8,1],cmap="gray")
plt.gca().invert_yaxis()
plt.scatter(seg[8,1][:,0], seg[8,1][:,1], c='r',alpha=0.1);
```

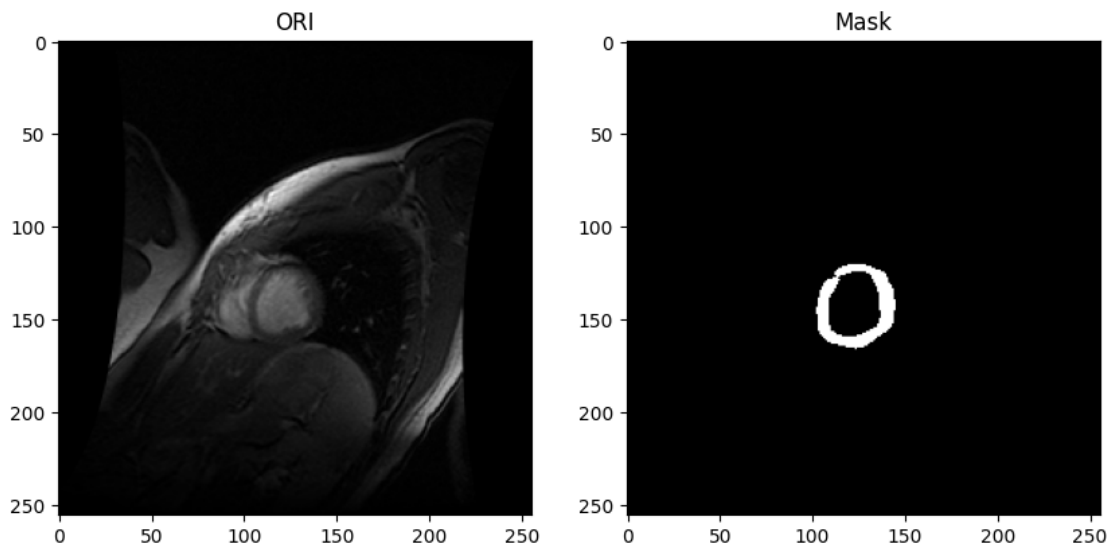
```
MRI volume of the heart composed of 11 slices along the z axis and 20 temporal
frames. Each slice is an image  256  x  256
For each slice and at each time frame we have a manual segmentation composed of
65 2D landmarks
Be careful, some slices do not contain the left ventricle myocardium and the
manual segmentation is not simply empty but it contains the value: [[-99999]]
```

Sample slices at a single time frame with the red manual segmentation at the bottom



4.1 For cardiac MRI: how to create a binary mask from the provided contours

[7]:
```python
from skimage.draw import polygon

# Remove [0 0] point in provided contour
Cont_cardiac_ex = Cont_cardiac_ex[~np.all(Cont_cardiac_ex == 0, axis=1)]

Seg_cardiac_ex    = np.zeros_like(Img_cardiac_ex, dtype='bool')
rr, cc = polygon(Cont_cardiac_ex[:, 1], Cont_cardiac_ex[:, 0], Img_cardiac_ex.
 ↪shape)
Seg_cardiac_ex[rr, cc] = 1

fig, axes          = plt.subplots(1,2, figsize=(10, 10))
ax                 = axes.ravel()
ax[0].imshow(Img_cardiac_ex,cmap='gray')
ax[0].set_title("ORI")
ax[1].imshow(Seg_cardiac_ex,cmap='gray')
ax[1].set_title("Mask")

plt.show()
```



5 Preliminaries

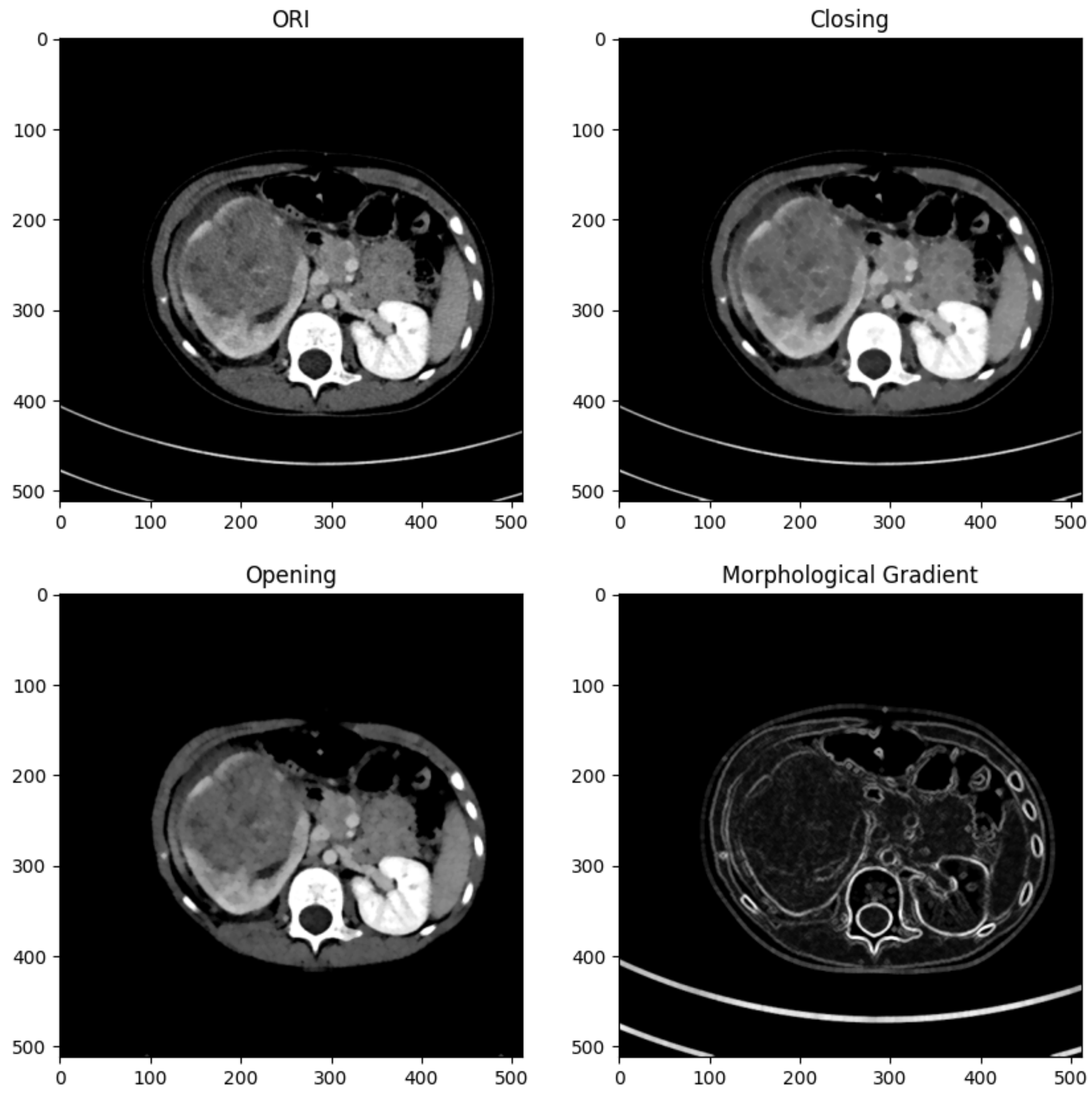
5.1 Mathematical Morphology

Incentives to use morphological operators seen during the previous lectures to segment the provided images.

Think about the structural elements and the hyper-parameters … We typically adapt their values to the image resolution and type of structures targeted in our segmentation (eg. brigth or dark).

[8]:
```
#Select input image
# Img_test          = Img_cardiac_ex
# Img_test          = Img_brain_ex
Img_test          = Img_abdo_ex ; # With: Seg_abdo_kidney_ex, Seg_abdo_tumor_ex

# Define Element
Radius            = 2
se                = disk(Radius)

# Morpho closing
Img_test_close   = morpho.closing(Img_test,se)

# Morpho Opening
Img_test_open     = morpho.opening(Img_test,se)

# Morpho Gradient
Img_test_grad   = morpho.dilation(Img_test,se)-morpho.erosion(Img_test,se)
```

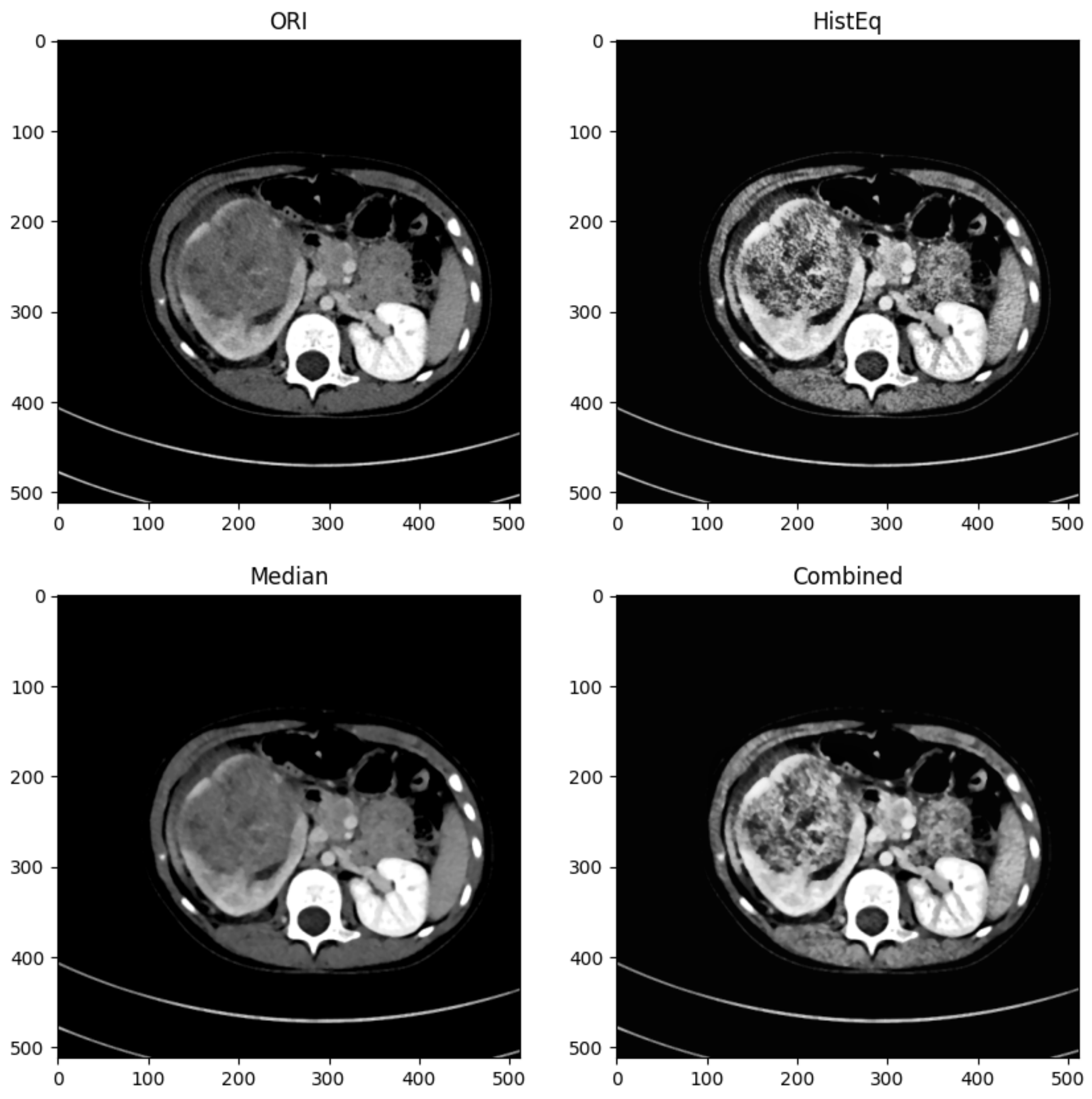
```
#Figure display
fig, axes        = plt.subplots(2,2, figsize=(10, 10))
ax               = axes.ravel()
ax[0].imshow(Img_test,cmap='gray')
ax[0].set_title("ORI")
ax[1].imshow(Img_test_close,cmap='gray')
ax[1].set_title("Closing")
ax[2].imshow(Img_test_open,cmap='gray')
ax[2].set_title("Opening")
ax[3].imshow(Img_test_grad,cmap='gray')
ax[3].set_title("Morphological Gradient")

plt.show()
```



6 Preliminaries

6.1 Regular filtering

[9]:
```
img_histeq    = skimage.exposure.equalize_adapthist(Img_test, clip_limit=0.03)
img_median    = ndimage.median_filter(Img_test, size=4)
img_histeqmed = ndimage.median_filter(img_histeq, size=4)

fig, axes        = plt.subplots(2,2, figsize=(10, 10))
ax               = axes.ravel()
ax[0].imshow(Img_test,cmap='gray')
ax[0].set_title("ORI")
ax[1].imshow(img_histeq,cmap='gray')
ax[1].set_title("HistEq")
ax[2].imshow(img_median,cmap='gray')
ax[2].set_title("Median")
ax[3].imshow(img_histeqmed,cmap='gray')
ax[3].set_title("Combined");
```



7 ASSIGNMENT DETAILS:

Overall task: Develop a segmentation pipeline building upon **kmeans** as proposed below for **at least one application** and report quality of segmentation results comparing your results to the provided ground truth on multiple images.

We want you to focus on **pre-processing** your image AND **post process** your segmentation result to extract **THE structure(s) of interest**. For the postprocessing you can rely on extraction of connected components and apply some criteria (size, shape, position,..) to extract the component(s) that most likely correspond(s) to the structure(s) of interest.

When processing medical images, and given that you are provided with several ground-truth (gt) segmentations you can push the exercice to consider: * **Cropping** the field of view to remove the background (a common issue in medical images) * Target a **range of intensity values** based on learning from the gt masks and corresponding images you are given. * Learn **priors on shape/intensity statistics** from the provided gt masks

7.1 Kmeans routine to pre-segment your images

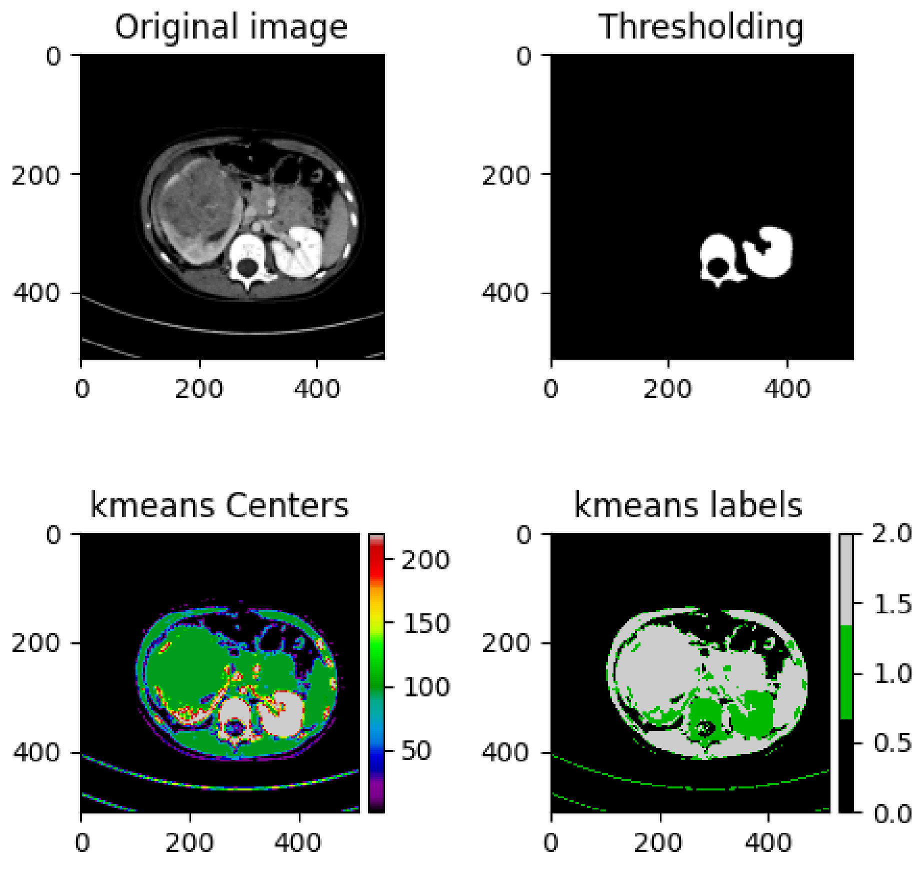
Kmeans is very often used as pre-segmentation to initialise a finer segmentation.

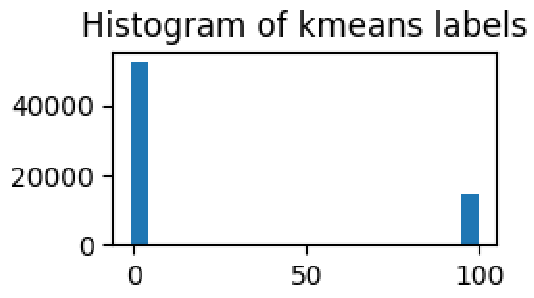
Provides a segmentation of the image using k-means clustering. Be careful: Kmeans uses random initialisation and is therefore different at each run and randomly assigns labels to clusters (0,1,2,…,K).

[10]:
```
# Image use-case and HYPER-PARAMETER values provided as a good start

Img_test          = Img_abdo_ex ; Img_seg_gt       = Seg_abdo_kidney_ex # With:␣
 ↪Seg_abdo_kidney_ex, Seg_abdo_tumor_ex
nber_clusters    = 3; Target_value_thresh = 180 # For abdo image
# Img_test              = Img_cardiac_ex; Img_seg_gt       = Seg_cardiac_ex
# nber_clusters         = 10; Target_value_thresh = 150 # For cardiac  image
# Img_test            = Img_brain_ex ; Img_seg_gt       = Seg_brain_ex
# nber_clusters       = 10; Target_value_thresh = 50 # For brain image




# [1] Example of segmentation via simple threshold
# Compute a mask thresholding above target value
Seg_thresh          = morphology.remove_small_holes(
                        morphology.remove_small_objects(
                          Img_test > Target_value_thresh,
                            min_size=500,connectivity=1),
                             area_threshold=500)
Seg_thresh = morphology.opening(Seg_thresh, morphology.disk(3))
Seg_thresh = Seg_thresh.astype('uint8')


# [2] Example of segmentation via kmeans


Seg_km,Seg_km_labels = my_kmeans(Img_test,nber_clusters)
# get discrete colormap to display results
Colormap  = plt.get_cmap('nipy_spectral', nber_clusters)

# Figure: Display data
fig, axes = plt.subplots(2,2, figsize=(5, 5))
ax        = axes.ravel()
ax[0].imshow(Img_test, cmap='gray')
ax[0].set_title('Original image')
```

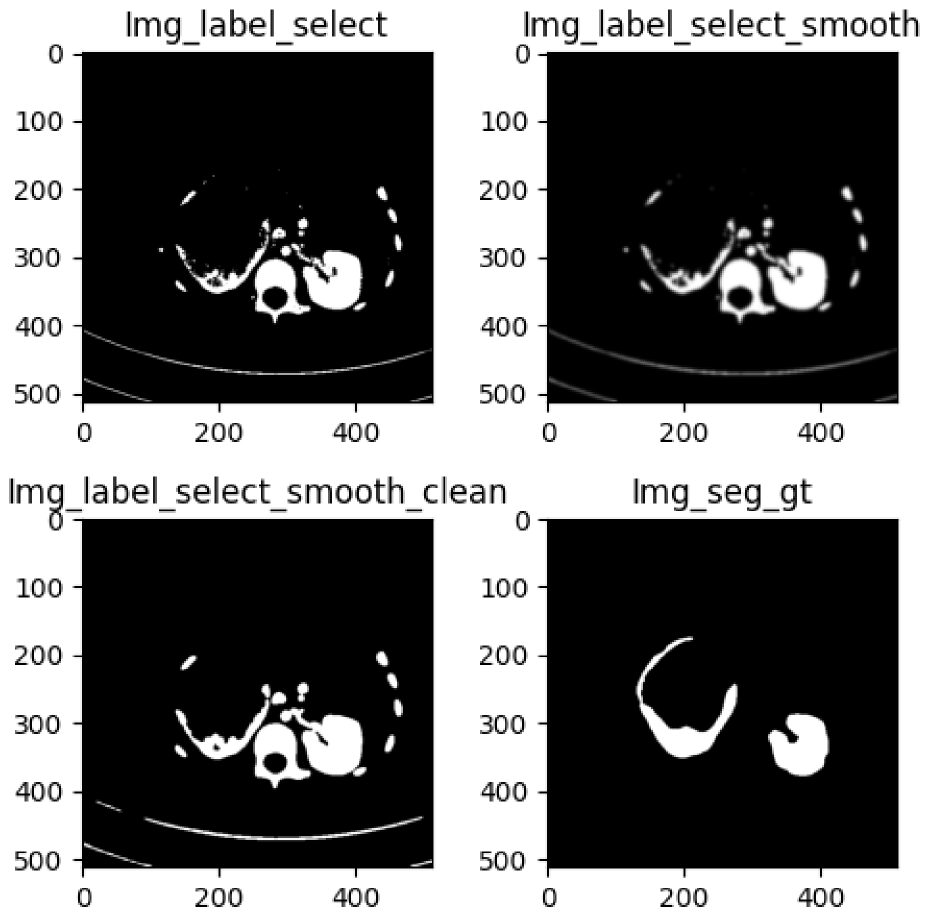
```python
ax[1].imshow(Seg_thresh, cmap='gray')
ax[1].set_title('Thresholding')
tmp       = ax[2].imshow(Seg_km, cmap='nipy_spectral')
divider   = make_axes_locatable(ax[2])
cax       = divider.append_axes('right', size='5%', pad=0.05)
ax[2].set_title('kmeans Centers')
fig.colorbar(tmp, cax=cax, orientation='vertical')
tmp     = ax[3].imshow(Seg_km_labels, cmap=Colormap)
divider = make_axes_locatable(ax[3])
cax     = divider.append_axes('right', size='5%', pad=0.05)
fig.colorbar(tmp, cax=cax, orientation='vertical')
ax[3].set_title('kmeans labels')
fig.tight_layout()
plt.show();

# Get mean pixel intensity values under all Kmeans labels
Img_label_means = np.empty(nber_clusters)
for i in range(0,nber_clusters):
    Img_label_mask = Seg_km_labels==i
    Img_label_mask = Img_label_mask.astype('uint8')
    tmp            = cv2.mean(Img_test, Img_label_mask)
    Img_label_means[i] = tmp[0]
#print(Img_label_means)

# Plot histograms of Kmeans clusters
Bins = np.sort(np.concatenate((0,Img_label_means), axis=None))
hist_kmeans, bins_kmeans  = np.histogram(Seg_km.flatten(),
                                         bins=Bins)
#Figure: plot histograms withins clusters from Kmeans
bar_width = 5
fig       = plt.figure(figsize=(2, 1))
ax        = fig.add_axes([0,0,1,1])
ax.bar(bins_kmeans[1:-1],(hist_kmeans[1:]),bar_width);
ax.set_title('Histogram of kmeans labels')
plt.show()

# Set a target intensity value and get the Kmeans label closest to it
Target_value = cv2.mean(Img_test, Seg_thresh)
Target_value = Target_value[0]
Diff         = np.absolute(Img_label_means-Target_value*np.ones(nber_clusters))
Label_select = np.argmin(Diff)

# Prints:
print('Nber of classes in Kmeans (input)= ', nber_clusters)
print('Your intensity Target_value = ', np.round(Target_value).astype(int))
print('Distance of your Target_value to Kmeans Centers = ', np.round(Diff).
 ↪astype(int))
```

```
print('Your selected label (starting with 0)= ', Label_select)



# Filter the selected label and get a binary segmentation mask
Img_label_select  = Seg_km_labels==Label_select
Img_label_select  = Img_label_select.astype('float64')


# Examples to refine your segmentation mask
Img_label_select_smooth = gaussian(Img_label_select, 2,
                            preserve_range=True)

Img_label_select_smooth_clean = morphology.remove_small_holes(
    morphology.remove_small_objects(
        Img_label_select_smooth>0.25, min_size=50,connectivity=1),
    area_threshold=50)


fig, axes = plt.subplots(2,2, figsize=(5, 5))
ax        = axes.ravel()
ax[0].imshow(Img_label_select, cmap='gray')
ax[0].set_title('Img_label_select')
ax[1].imshow(Img_label_select_smooth, cmap='gray')
ax[1].set_title('Img_label_select_smooth')
ax[2].imshow(Img_label_select_smooth_clean, cmap='gray')
ax[2].set_title('Img_label_select_smooth_clean')
ax[3].imshow(Img_seg_gt, cmap='gray')
ax[3].set_title('Img_seg_gt')

fig.tight_layout()
plt.show()
```





```
Nber of classes in Kmeans (input)=  3
Your intensity Target_value =  242
Distance of your Target_value to Kmeans Centers =  [241  22 145]
Your selected label (starting with 0)=  1
```



7.2 From Kmeans to Connected Components

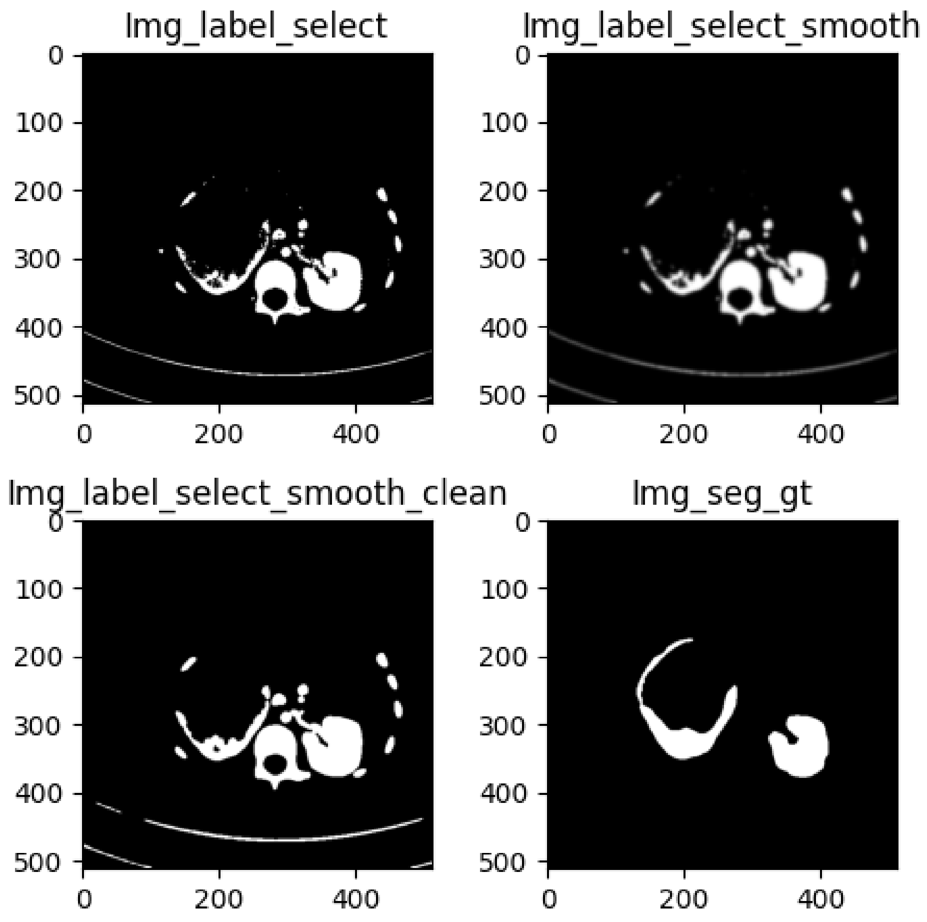
This is an example on how to exploit the results from Kmeans. Note that this example uses several hard-coded **hyperparameters** which is not appropriate to segment robustly several cases.

If you reuse this piece of code for the question below, propose some approaches to set some of the hyperparameter values automatically (eg using the known average size of the structure of interest from the gt segmentation you have)

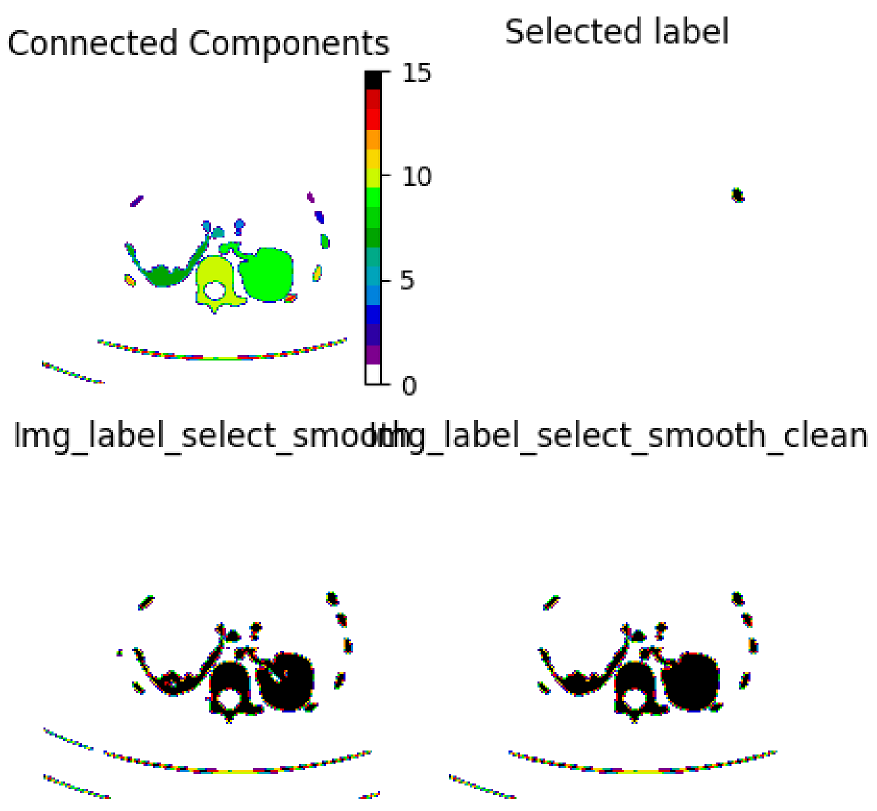
[11]:
```
# Examples to refine your segmentation mask
Img_label_select_smooth = gaussian(Img_label_select, 2,
                            preserve_range=True)

Img_label_select_smooth_clean = morphology.remove_small_holes(
    morphology.remove_small_objects(
        Img_label_select_smooth>0.25, min_size=50,connectivity=1),
    area_threshold=50)
```

```
fig, axes = plt.subplots(2,2, figsize=(5, 5))
ax        = axes.ravel()
ax[0].imshow(Img_label_select, cmap='gray')
ax[0].set_title('Img_label_select')
ax[1].imshow(Img_label_select_smooth, cmap='gray')
ax[1].set_title('Img_label_select_smooth')
ax[2].imshow(Img_label_select_smooth_clean, cmap='gray')
ax[2].set_title('Img_label_select_smooth_clean')
ax[3].imshow(Img_seg_gt, cmap='gray')
ax[3].set_title('Img_seg_gt')

fig.tight_layout()
plt.show()



Img_label_select_filter = gaussian(Img_label_select, 2,
                             preserve_range=True)
Img_label_select_smooth = Img_label_select_filter>0.25
Img_label_select_smooth_clean = morphology.remove_small_holes(
    morphology.remove_small_objects(
        Img_label_select_smooth, min_size=150,connectivity=1),
    area_threshold=150)

# Extract connected components
Thresh = Img_label_select_smooth_clean
output = cv2.connectedComponentsWithStats(Thresh.astype(np.uint8))
(numLabels, labels, stats, centroids) = output
print('Max value in labels = ',labels.max())
print('Number of connected components = ',numLabels)

# Extract 1 connected component
thresh = labels==1

# plots results
Colormap = my_colormap_white_bkg('nipy_spectral',numLabels)
fig, axes = plt.subplots(2,2, figsize=(5, 5))
ax        = axes.ravel()

tmp      = ax[0].imshow(labels, cmap=Colormap)
divider   = make_axes_locatable(ax[0])
cax       = divider.append_axes('right', size='5%', pad=0.05)
ax[0].set_title('Connected Components')
ax[0].axis('off')
fig.colorbar(tmp, cax=cax, orientation='vertical')

ax[1].imshow(thresh, cmap=Colormap)
```

```
ax[1].set_title('Selected label')
ax[1].axis('off')
ax[2].imshow(Img_label_select_smooth, cmap=Colormap)
ax[2].set_title('Img_label_select_smooth')
ax[2].axis('off')
ax[3].imshow(Img_label_select_smooth_clean, cmap=Colormap)
ax[3].set_title('Img_label_select_smooth_clean');
ax[3].axis('off');
```



```
Max value in labels =  15
Number of connected components =  16
```



7.3 Implement your full segmentation pipeline

7.3.1 TO DO:

- Implement a full pipeline to segment **TWO EXAMPLES** from a single use-case. You must print in the report that code which was run over the two cases. We need to see in your code that your were able to run the same pipeline on two cases without changing any hyperparameter or changing them based on image-specific information. So comment the code and list all hyperparameter values clearly at the start of the cell.
- Implement and compute a quality metric (eg Dice, overlap, relative area differences, errors between max diameters (something used to measure tumors), distances between contours) that compares your final segmentation with the ground-truth.
- Your final solution can involve registration or active contours from previous lectures. Just make sure to include the required functions and imports in your final notebook.
- **Extra point(s) if** :
  1. you provide results on more use-cases or more than 2 examples per use case.
  2. you test your segmentation pipeline on images degraded by noise for example.

7.3.2 If you use the cardiac MRI dataset:

- Extra point because extra difficulties to load and prepare the data.
- In this case the data preparation is part of your code below as you can only rely on the input

data provided as it is.
- Use any slice from any case you want as your "learning" ground truths on which you can learn object size or shape characteristics for example.
- Test your segmentation on any slice from any case, as long as the slices were not part of your "learning" ground-truth.
- See if you can get your solution to not detect anything on slices that don't contain the left ventricle myocardium.

7.3.3 Important instructions:

- You cannot use the ground truth of the images you segment to segment them! But you can use the ground-truths of other examples of the same use-case to learn a priori knowledge such as size or average pixel intensity.
- List all your hard-coded hyperparameters at the beginning of your code, indicating a variable name and its value. You will be penalised if you leave any hard-coded hyperparameter values inside your code.

Nb: Dice = np.sum([seg==gt])*2.0/(np.sum(seg)+np.sum(gt)) #seg is the segmentation and gt is the ground truth. Both are of same size

[12]:
```
brainMRI_path = './data/brainMRI'
os.listdir(brainMRI_path)
listImagesbrainMRI=glob.glob(brainMRI_path + '/*-seg.png')
print('There are', len(listImagesbrainMRI),  'brain MRI images')
print(listImagesbrainMRI)
```

```
There are 4 brain MRI images
['./data/brainMRI/MRIs1-seg.png', './data/brainMRI/MRIs2-seg.png',
'./data/brainMRI/MRIs3-seg.png', './data/brainMRI/MRIs4-seg.png']
```

7.3.4 Learning the size of the ground-truth segmentation

[13]:
```
indexIM_learning = 1
filename_seg = listImagesbrainMRI[indexIM_learning]
mask = imread(filename_seg)/255

size_segmentation = mask[mask==1].shape[0]
print(size_segmentation)
```

```
488
```

[14]:
```
# Hyper-parameters

scale = 100 # for the felzenszwalb algorithm
sigma = 1 # sigma for the gaussian filter
size_median = 4 # size of the median filter
```

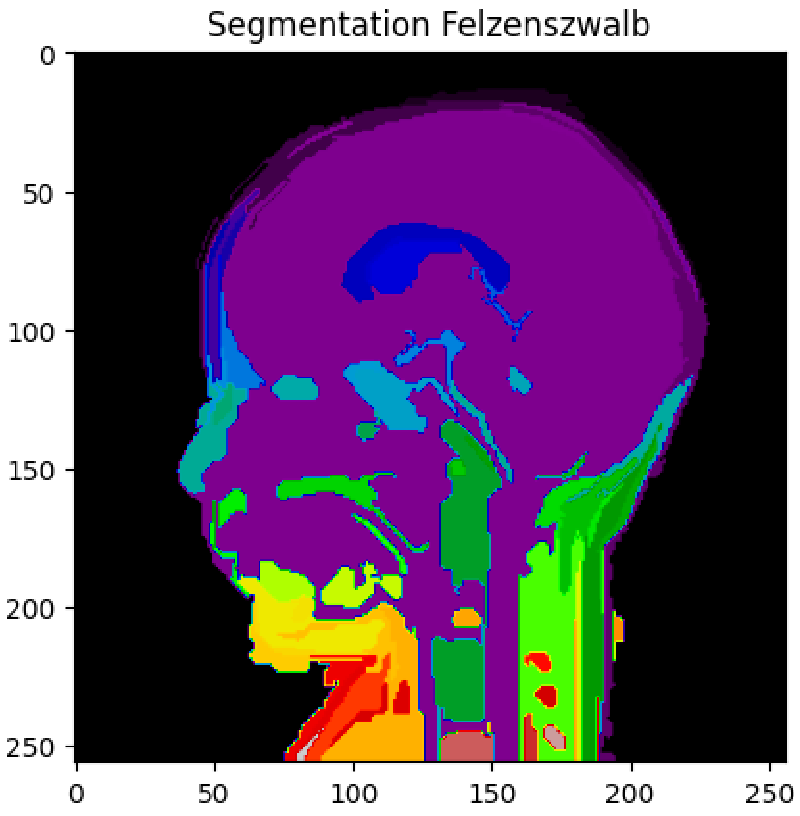
7.3.5 Using felzenszwalb segmentation

L'algorithme modélise l'image par un graphe $G = (V, E)$ tel que V représente les pixels de l'image et pour tout $v_i, v_j \in V$, $(v_i, v_j) \in E$ si et seulement si v_i et v_j sont voisins dans l'image (8-connexité). Enfin, chaque arête est affectée d'un poinds tels que $\omega(v_i, v_j) = |I_{v_j} - I_{v_i}|$. L'algorithme fusionne des pixels en superpixels si ces pixels sont reliés par une arête de poids est inférieure à un certain seuil défini par la variance interne potentielle du superpixel.

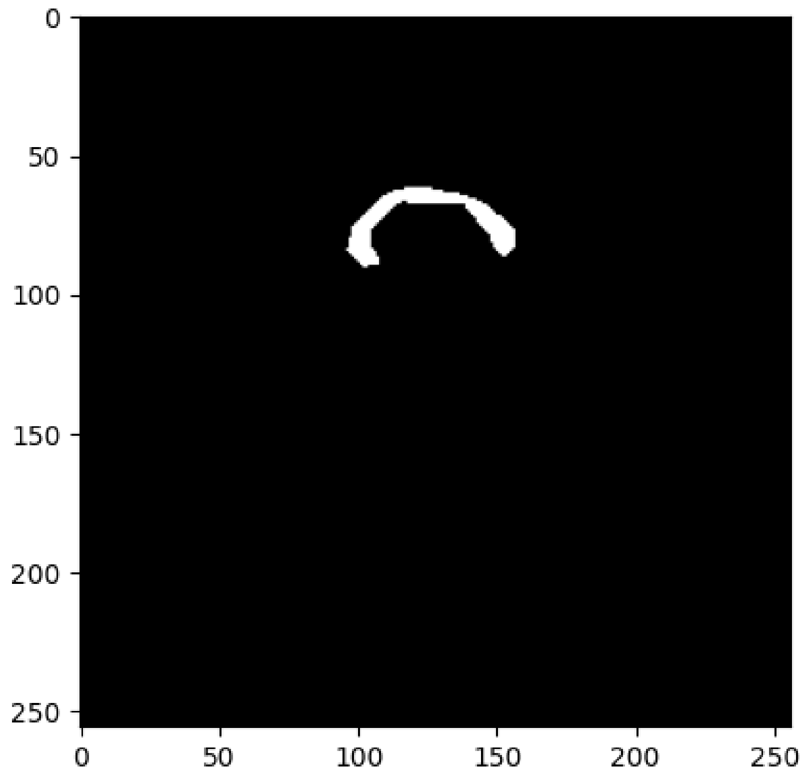
On se sert ensuite de la taille a-priori pour concentrer nos recherches de la zone d'intérêt sur un échantillon de segments restreints (10 images).

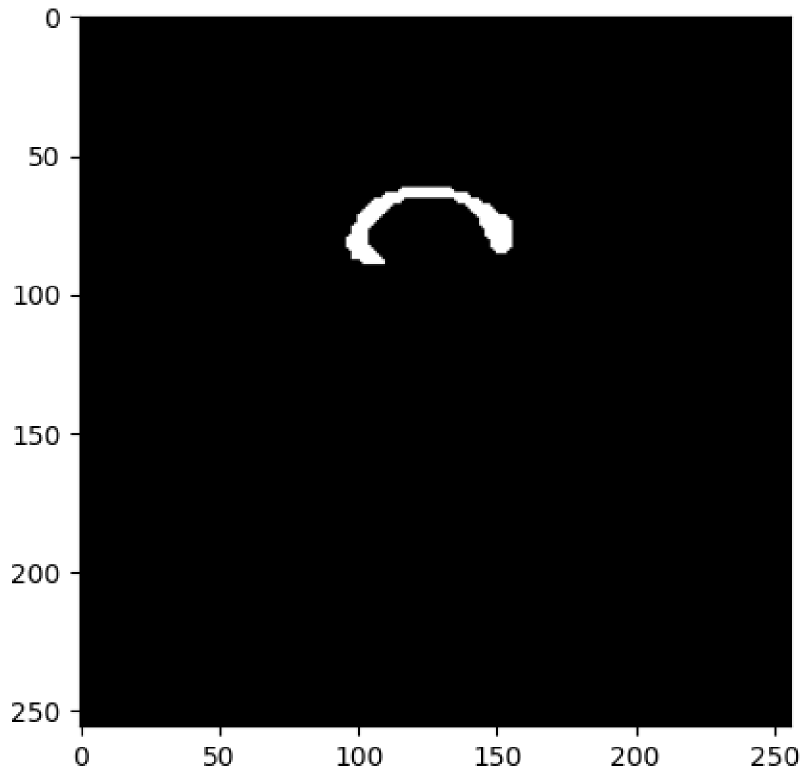
```
[15]: from skimage.segmentation import felzenszwalb

# Choose a brain MRI and plot it with the ground truth segmentation
indexIm      = 1 # between 0 and 3
filename_seg = listImagesbrainMRI[indexIm]
Seg_brain_ex = imread(filename_seg)/255
filename     = filename_seg[:-8] + '.png'
Img_brain_ex = imread(filename)

Img_test = ndimage.median_filter(Img_brain_ex, size=size_median)

segments = felzenszwalb(Img_test, sigma=sigma,scale=scale)
plt.imshow(segments, cmap="nipy_spectral")
plt.title("Segmentation Felzenszwalb")
plt.show()

# Trouver les labels uniques
# unique_labels = np.unique(segments)

# Dictionnaire pour stocker la différence de taille avec la taille apprise plus␣
 ↪haut
#diff_size = {label: np.abs(Img_test[segments == label].shape[0] -␣
 ↪size_segmentation) for label in unique_labels}

# Trier les segments par intensité moyenne (ordre croissant)
#sorted_labels = sorted(diff_size, key=diff_size.get)

#for i in range(10):
#    print(f"Label {sorted_labels[i]}: {diff_size[sorted_labels[i]]}")
#    print(Img_test[segments == sorted_labels[i]].shape[0])
#   plt.imshow(segments == sorted_labels[i], cmap="gray")
#  plt.figure()

plt.imshow(segments == 12, cmap="gray")
plt.figure()
plt.imshow(Seg_brain_ex, cmap="gray")
```

```
print(f"Error rate (using Sum Squared Difference) between Ground-Truth and␣
 ↪performed segmentation : {np.sum((segments==12-Seg_brain_ex)**2)/np.
 ↪sum(Seg_brain_ex**2)}")
```

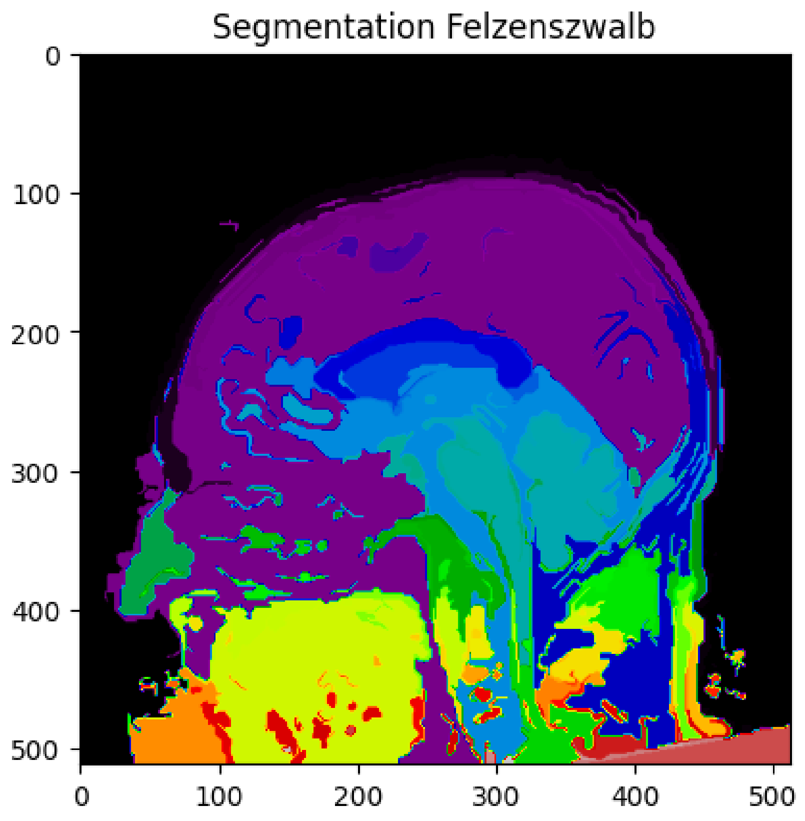


```
Error rate (using Sum Squared Difference) between Ground-Truth and performed
segmentation : 0.1557377049180328
```

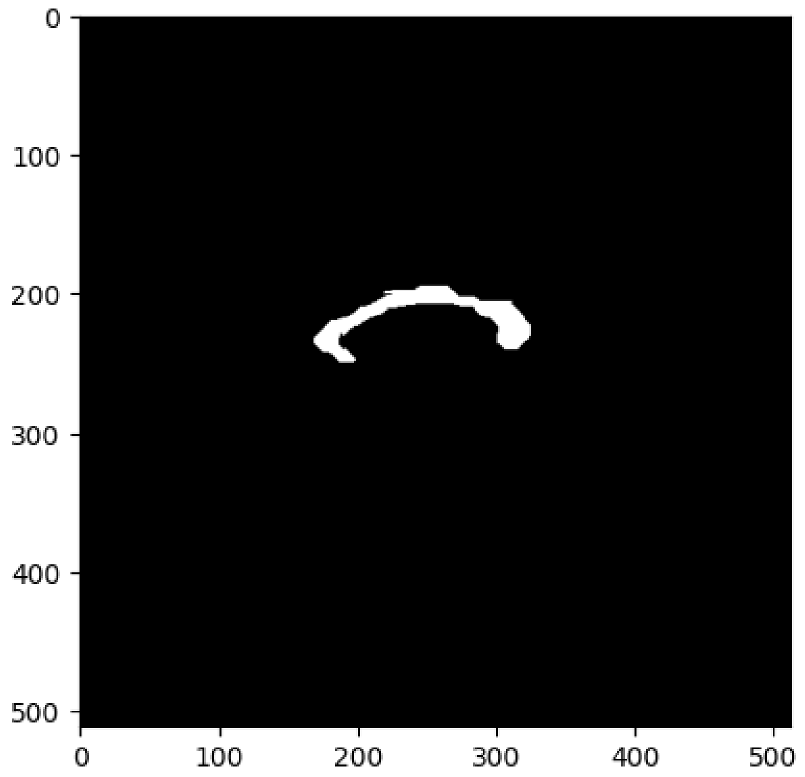

0
50
100
150
200
250
0
50
100
150
200
250

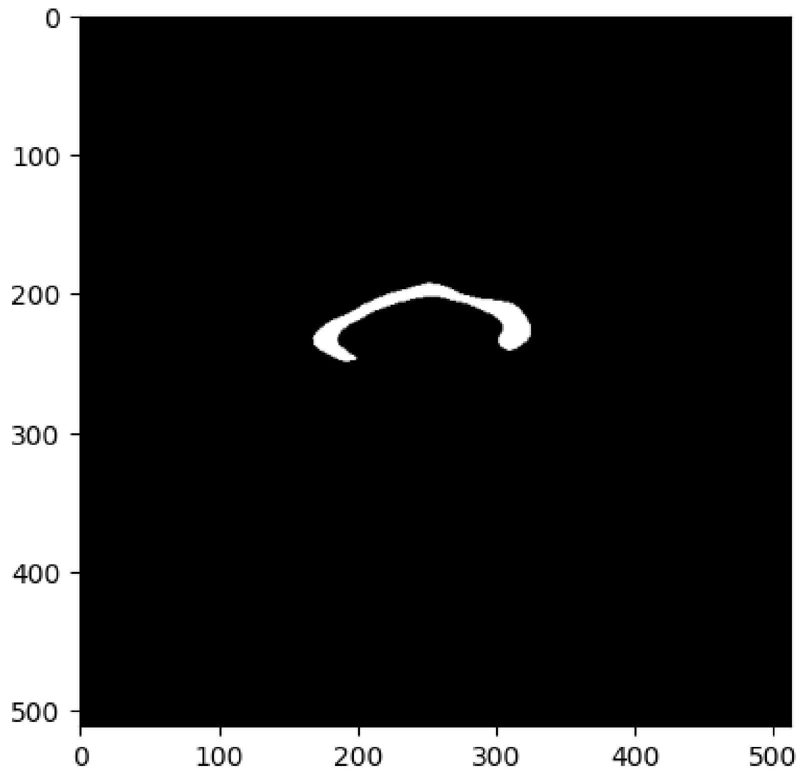

0
50
100
150
200
250
0
50
100
150
200
250

[16]:
```
# Choose a brain MRI and plot it with the ground truth segmentation
indexIm      = 3 # between 0 and 3
filename_seg = listImagesbrainMRI[indexIm]
Seg_brain_ex = imread(filename_seg)
filename     = filename_seg[:-8] + '.png'
Img_brain_ex = imread(filename)

Img_test = ndimage.median_filter(Img_brain_ex, size=size_median)

segments = felzenszwalb(Img_test, sigma=sigma,scale=scale)
plt.imshow(segments, cmap="nipy_spectral")
plt.title("Segmentation Felzenszwalb")
plt.show()

# Trouver les labels uniques
#unique_labels = np.unique(segments)

# Dictionnaire pour stocker l'intensité moyenne de chaque région
#diff_size = {label: np.abs(Img_test[segments == label].shape[0] -␣
 ↪size_segmentation) for label in unique_labels}
#sorted_labels = sorted(diff_size, key=diff_size.get)

#for i in range(10):
#    print(f"Label {sorted_labels[i]}: {diff_size[sorted_labels[i]]}")
#    plt.imshow(segments == sorted_labels[i], cmap="gray")
#    plt.figure()

# Trier les segments par intensité moyenne (ordre croissant)


plt.imshow(segments == 49, cmap="gray")
plt.figure()
plt.imshow(Seg_brain_ex, cmap="gray")
print(f"Error rate (using Sum Squared Difference) between Ground-Truth and␣
 ↪performed segmentation : {np.sum((segments==49-Seg_brain_ex)**2)/np.
 ↪sum(Seg_brain_ex**2)}")
```



```
Error rate (using Sum Squared Difference) between Ground-Truth and performed
segmentation : 0.21944177093359
```


0
100
200
300
400
500
0
100
200
300
400
500


0
100
200
300
400
500
0
100
200
300
400
500

La segmentation est sensiblement proche de la ground-truth fournie, mais donne en général des contours moins réguliers. Elle nécessite peux de prétraitement (seulement un filtre médian puis gaussien appliqué pour améliorer la régularité des countours).

Pour la segmentation, on utilise une échelle de 100 adaptée à la taille des tissus cérébraux, un filtre médian de taille 4 et un filtre gaussien de variance $\sigma^2 = 1$.