Practical Session 3

Biomedical Signals and Images

Biomedical Image Processing (Image enhancement, Filtering and Segmentation)

ETRO: Deparment of Electronics and Informatics

Vrije Universiteit Brussel

Jakub Ceranka, Hamza Mekhzoum and Jef Vandemeulebroucke

Questions email or Teams: jakub.ceranka@vub.be (mailto:jakub.ceranka@vub.be), hamza.mekhzoum@vub.be (mailto:hamza.mekhzoum@vub.be)

Student Names and IDs:

Students names and IDs: Siyan Luo(0594750) & Faiza Tasnia (0563547)
Academic Year : 2022-2023

Purpose

The purpose of this exercise session is to obtain insight in the image enhancement, filtering and segmentation operations commonly applied in medical image processing. For more information on these concepts see the course slides and the related material.

The jupyter notebook should be submitted as the report of each practical session by teams of two students. In colab (https://colab.research.google.com/notebooks/welcome.ipynb) you should download the notebook in the format *.ipynb and save it as a pdf version through print->save as pdf. Both the jupyter notebook and the pdf should be uploaded on canvas in a zip file before the deadline. The deadline for the report submission is December 19th, 2022, at 23.59.

Any report sent after the deadline will not be graded.

Required modules

- numpy (https://pypi.org/project/numpy/)
- pylab (https://scipy.github.io/old-wiki/pages/PyLab)
- scipy (https://www.scipy.org/getting-started.html)
- skimage (https://scikit-image.org/docs/dev/api/skimage.html#module-skimage)
- math (https://docs.python.org/3/library/math.html)
- sklearn (https://scikit-learn.org/stable/index.html)
- Matplotlib (https://matplotlib.org/3.1.1/index.html)

1 Image enhancement

1.1 The image histogram

The histogram is a representation of how many pixels have a certain intensity in the corresponding image. Medical images can however have a large intensity range, or even floating point intensities, making the pixel count per intensity low or impractical. In practice, intensities are therefore usually binned, i.e. grouped in a reduced number of bins with similar intensity.

1.2 Image enhancement

We shall discuss two ways of contrast improvement. The first is linear contrast mapping (http://homepages.inf.ed.ac.uk/rbf/HIPR2/stretch.htm) or histogram stretching (https://scikit-image.org/docs/dev/auto_examples/color_exposure/plot_equalize.html), which involves a linear transformation on the image intensities, such that the transformed intensities cover to the full range. Another way to improve the contrast is to perform histogram equalisation (https://scikit-image.org/docs/dev/auto_examples/color_exposure/plot_equalize.html). In this case, the aim is to obtain a uniform histogram, in which all intensities are equally represented. This can be done by applying a nonlinear transformation on the image intensities. It can be shown that the transform corresponds to the cumulative histogram.

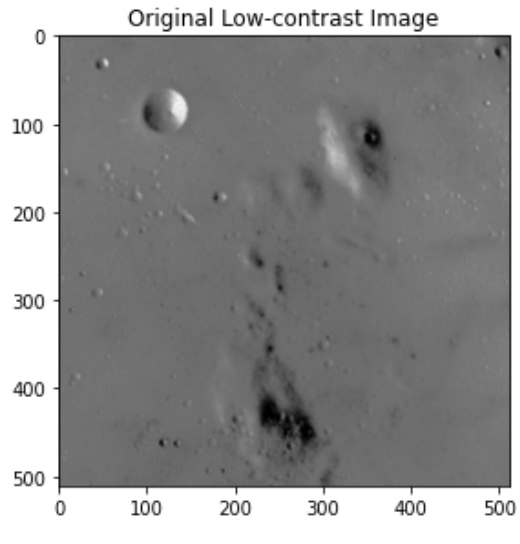
In [ ]:
```
from google.colab import drive
drive.mount('/content/gdrive')

from google.colab import files
files.upload()

```

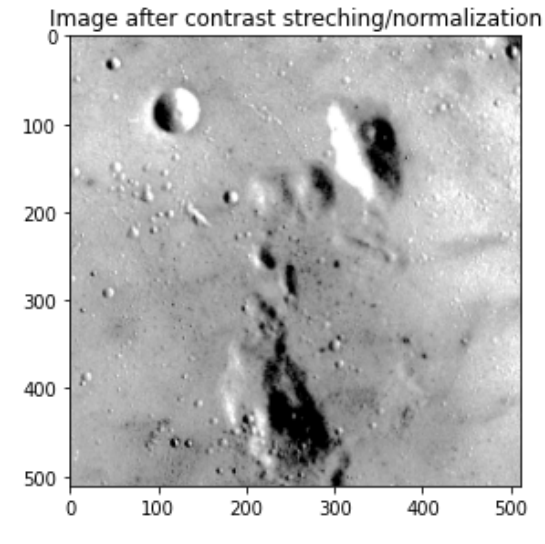
In [ ]:
```
!pip install imread
!pip install -U numpy
```

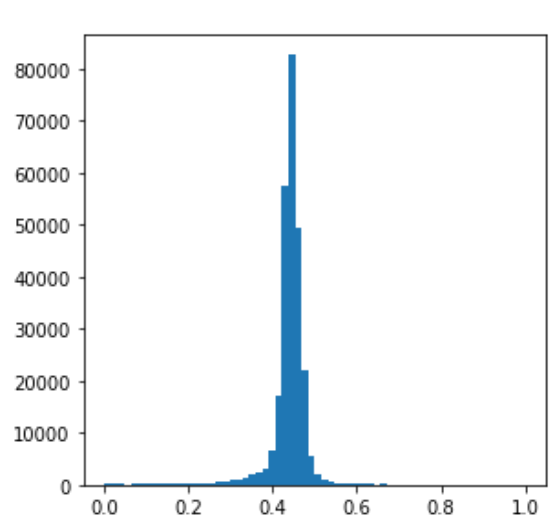
Example 1: Linear contrast mapping

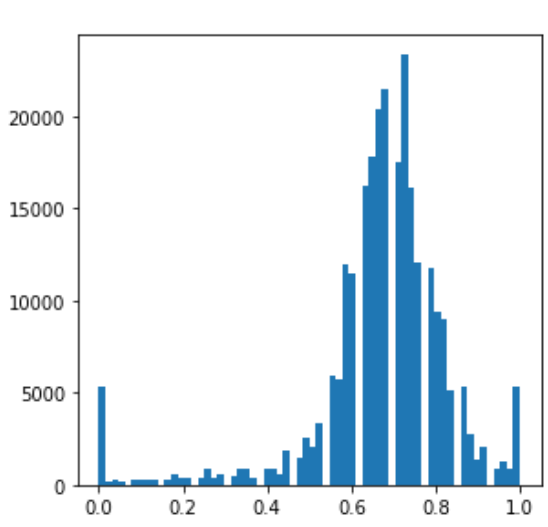
In [ ]:
```
import numpy as np
from skimage import data, img_as_float
from pylab import show,title,figure,imshow,subplot,subplots_adjust, hist

# Load an example image
low_contrast_image = data.moon()

"contrast stretching (i.e., normalization),"
"where the image is rescaled to include all intensities that fall within the 2nd and 98th percentiles."

p2, p98                  = np.percentile(low_contrast_image , (2, 98))

i_min, i_max             = p2,p98

image_cliping            = np.clip(low_contrast_image , i_min,i_max)

image_constrast_streched = (image_cliping - i_min) / float(i_max - i_min)

"Displaying low contrast image, contrast-enhanced image and their corresponding histogrms"

h,w=2,2    # figure height and width
figure(figsize=(10,10))
subplots_adjust(hspace=.2)

subplot(h,w,1)
imshow(low_contrast_image , cmap='gray')
title('Original Low-contrast Image')

subplot(h,w,2)
imshow(image_constrast_streched, cmap='gray')
title('Image after contrast streching/normalization')

subplot(h,w,3)
hist(img_as_float(low_contrast_image).ravel(),bins=64) #plotting histogram of original low-contrast image

subplot(h,w,4)
hist(image_constrast_streched.ravel(),bins=64)#plotting histogram of contrast streched/normalized image
show()
```

```
/usr/local/lib/python3.8/dist-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this
version of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```









Exercise 1.1: Linear contrast mapping

By following above example 1, solve the below exercise.

- Read an image (Brain.tiff)

  hint: user can load the image in different ways, like: skimage.io.imread (https://scikit-image.org/docs/dev/api/skimage.io.html#skimage.io.imread), plt.imread (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.imread.html), imread (https://pypi.org/project/imread/)

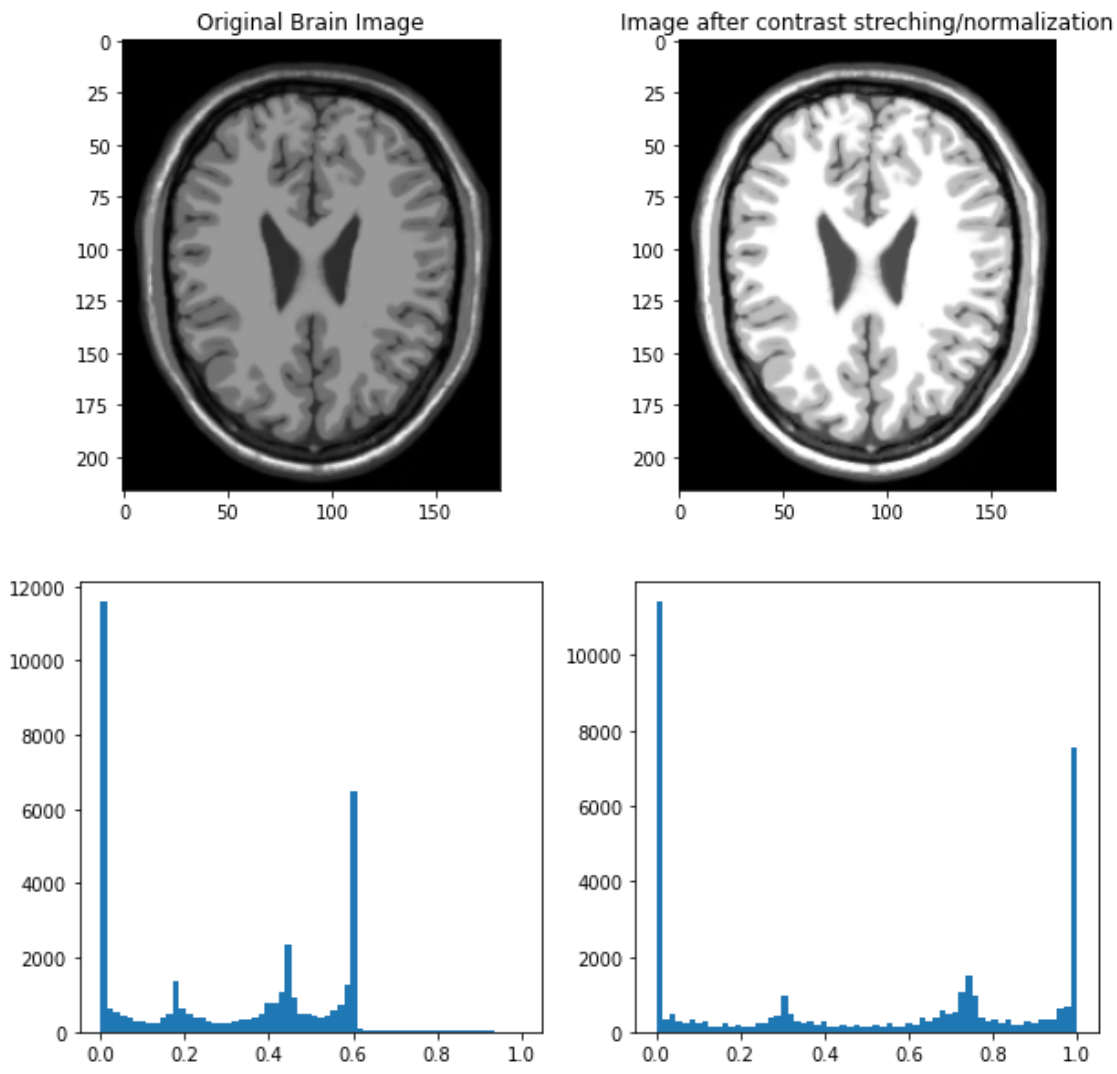
  preferable: from imread import imread. Install imread with pip install imread
- Perform linear contrast mapping (contrast streching/normalization)
- Display input image, output image (after linear contrast mapping), their corresponding histograms with 64 bins in a 2x2 figure.

Report

- Look at the output result and its histogram. Compare it with the histogram of the original input image. Comment about difference between them.

In [ ]:

```
# your code is here
# Load the image
from imread import imread

brain_image= imread("Brain.tiff")
"contrast stretching (i.e., normalization),"
"where the image is rescaled to include all intensities that fall within the 2nd and 98th percentiles."

p2, p98                  = np.percentile(brain_image , (2, 98))

i_min, i_max             = p2,p98

image_cliping            = np.clip(brain_image , i_min,i_max)

image_constrast_streched = (image_cliping - i_min) / float(i_max - i_min)

"Displaying low contrast image, contrast-enhanced image and their corresponding histogrms"

h,w=2,2    # figure height and width
figure(figsize=(10,10))
subplots_adjust(hspace=.2)

subplot(h,w,1)
imshow(brain_image , cmap='gray')
title('Original Brain Image')

subplot(h,w,2)
imshow(image_constrast_streched, cmap='gray')
title('Image after contrast streching/normalization')

subplot(h,w,3)
hist(img_as_float(brain_image).ravel(),bins=64) #plotting histogram of original low-contrast image

subplot(h,w,4)
hist(image_constrast_streched.ravel(),bins=64)#plotting histogram of contrast streched/normalized image
show()

```

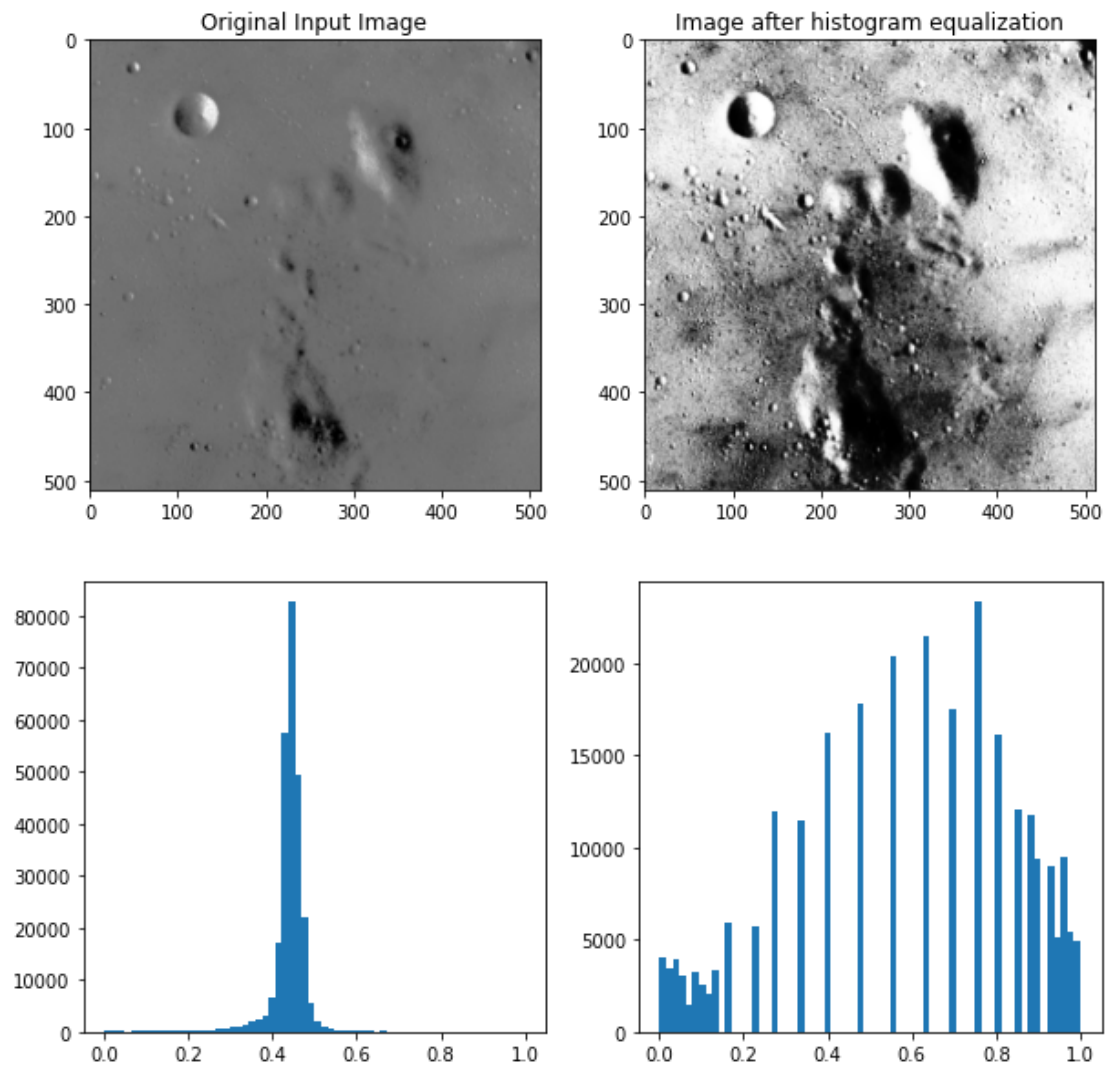


Example 2: Histogram equalisation

Students are suggested to have a look into [np.histogram (https://docs.scipy.org/doc/numpy/reference/generated/numpy.histogram.html)](https://docs.scipy.org/doc/numpy/reference/generated/numpy.histogram.html) for a calculation of histogram.

In [ ]:

```
import numpy as np
from skimage import data, img_as_float
import pylab
from pylab import show,title,figure,imshow,subplot,subplots_adjust

"Load an example image"
image = data.moon()

"calculation of histogram"
hist, bin_edges = np.histogram(image.ravel(), bins=64)
bin_centers = (bin_edges[:-1] + bin_edges[1:]) / 2
img_cdf = hist.cumsum()
img_cdf = img_cdf / float(img_cdf[-1])
out = np.interp(image.flat, bin_centers, img_cdf)
img_eq = out.reshape(image.shape)

"Displaying input image, image-histogram equalized and their corresponding histogrms"

h,w=2,2    # figure height and width
figure(figsize=(10,10))
subplots_adjust(hspace=.2)

subplot(h,w,1)
imshow(image , cmap='gray')
title('Original Input Image')

subplot(h,w,2)
imshow(img_eq, cmap='gray')
title('Image after histogram equalization')

subplot(h,w,3)
pylab.hist(img_as_float(image).ravel(),bins=64) #plotting histogram of original image

subplot(h,w,4)
pylab.hist(img_eq.ravel(),bins=64)#plotting histogram of histogram-equalized image
show()


```



Exercise 1.2: Histogram equalization

By following the example 2, solve the below exercise.

- Read an image (Brain.tiff) using imread (https://pypi.org/project/imread/)

  eg: from imread import imread

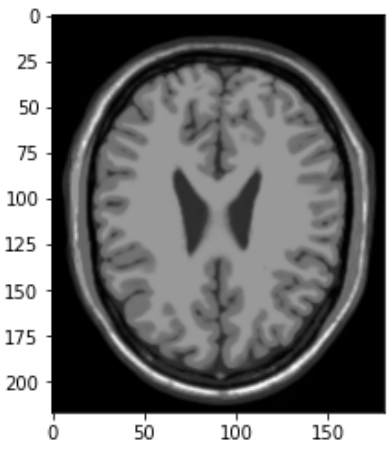
- Perform histogram equalization np.histogram (https://docs.scipy.org/doc/numpy/reference/generated/numpy.histogram.html)
- Display input image, output image (after linear contrast mapping) of exercise 1.1, output image (histogram-equalized), their corresponding histograms with 64 bins in a 3x2 figure.

Report

- Look at the output results and their histograms. Compare them with the histogram of the original input image. The histogram of the histogram-equalized output image is not perfectly uniform. What is the reason for this?

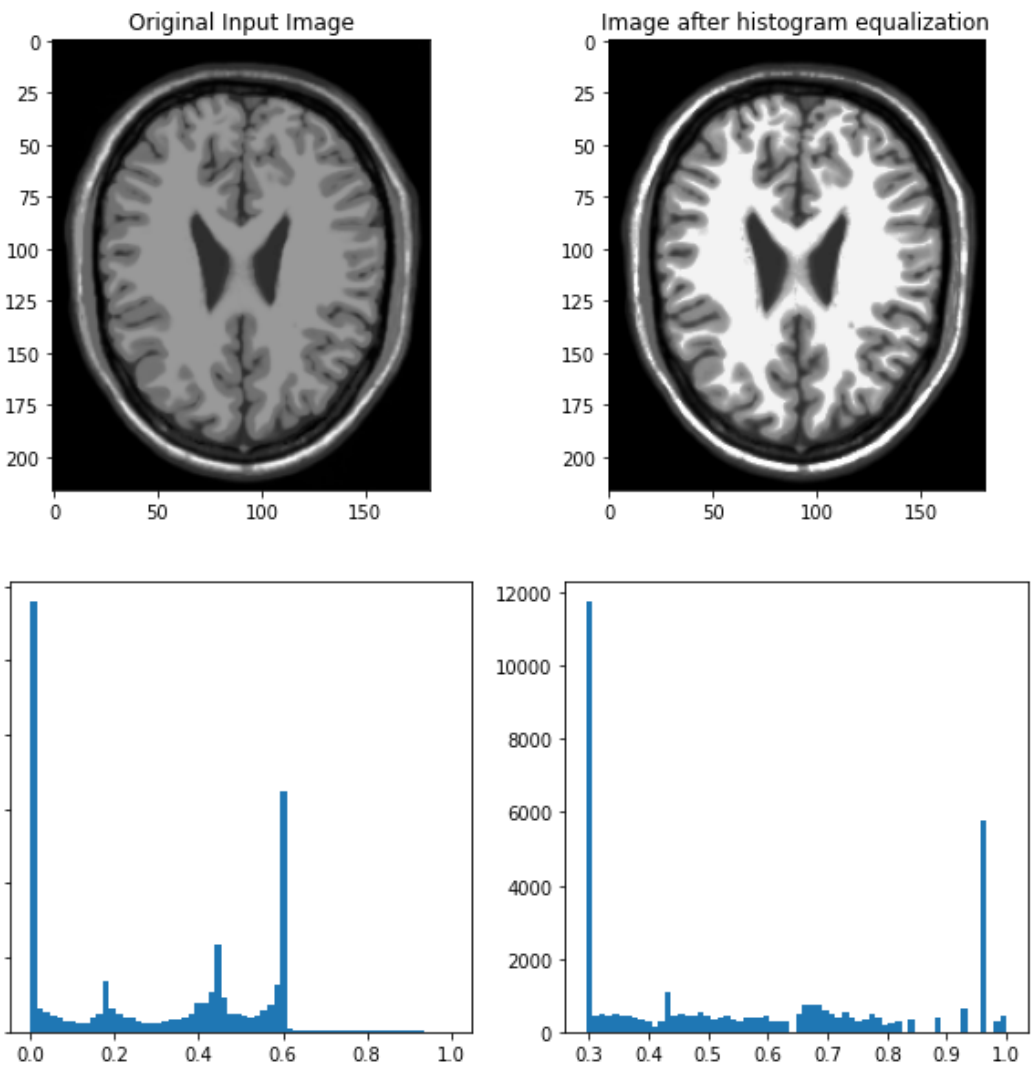
In [ ]:
```
# your code is here
from imread import imread
brain_image = imread("Brain.tiff")
imshow(brain_image , cmap='gray')
```

Out[5]: <matplotlib.image.AxesImage at 0x7fec517569d0>



In [ ]:
```
"calculation of histogram"
hist, bin_edges = np.histogram(brain_image.ravel(), bins=64)
bin_centers = (bin_edges[:-1] + bin_edges[1:]) / 2
img_cdf = hist.cumsum()
img_cdf = img_cdf / float(img_cdf[-1])
out = np.interp(brain_image.flat, bin_centers, img_cdf)
img_eq = out.reshape(brain_image.shape)
```

In [ ]:
```
"Displaying input image, image-histogram equalized and their corresponding histogrms"

h,w=2,2    # figure height and width
figure(figsize=(10,10))
subplots_adjust(hspace=.2)

subplot(h,w,1)
imshow(brain_image, cmap='gray')
title('Original Input Image')

subplot(h,w,2)
imshow(img_eq, cmap='gray')
title('Image after histogram equalization')

subplot(h,w,3)
pylab.hist(img_as_float(brain_image).ravel(),bins=64) #plotting histogram of original image

subplot(h,w,4)
pylab.hist(img_eq.ravel(),bins=64)#plotting histogram of histogram-equalized image
show()


```



2 Image Denoising

Filters are used in medical imaging to enhance or suppress certain features of images. They may be used to improve the image quality before reviewing them, or as a preprocessing step to improve the result of further image processing steps such as segmentation. For many filters, the extent of the neighbourhood considered for each pixel is determined by a spatial filter mask (kernel). The weights of the mask can be combined with the underlying pixels in a linear way, in which case this comes down to a convolution of the mask and image. Other filters however exist, based on non-linear operations.

2.1 Noise suppression

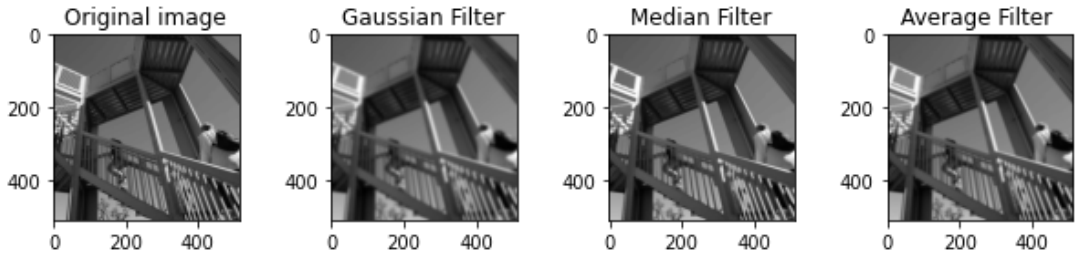
An important processing task is the suppression of noise, either for enhanced visualization or for improving the result of further processing. Noise can often be assumed to be a high frequency signal. Many noise reduction approaches are therefore based on attenuating the high frequency components while preserving the low frequency components. A popular linear filter for this purpose is the (2D) Gaussian filter. Low pass filtering for noise suppression has the side effect of blurring the edges of an image, which is often undesirable. Smoothing filters that preserve the edges of an image have therefore been proposed, such as the non-linear median filter.

2.2 Edge enhancement

Image filtering can also be used for the enhancement or detection of edges. The goal of such filters is often to enhance the edge contrast of an image in an attempt to improve its apparent sharpness. If the final goal is to retain an edge image, i.e. a binary image in which only the edges are preserved, the operation is termed edge detection. Such images can later on serve as inputs for further image processing steps such as segmentation.

Example 3: Image denoising by different filters (i.e, Gaussian filter (https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.gaussian_filter.html), median filter (https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.median_filter.html) and average filter (https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.convolve.html))

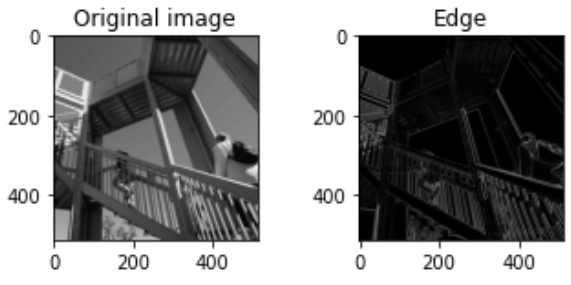
In [ ]:
```
from scipy import misc
from scipy import ndimage
from pylab import show,title,figure,subplot,subplots_adjust, imshow

image           = misc.ascent()                              # load input image
Gaussian_filter = ndimage.gaussian_filter(image, sigma = 3) # gaussian filter with standard deviation = 3
median_filter   = ndimage.median_filter(image, size = 7)    # median filter with kernel size of 7
kernel          = np.ones(shape=(6,6))/18                     # define kernel for average filter, kernel size is 6
average_filter  = ndimage.convolve (image, kernel)          # average filter

"displaying original image, results from gaussian, median and average filters"

h,w=1,4    # figure height and width
figure(figsize=(10,10));subplots_adjust(hspace=0.2,wspace=0.5)
subplot(h,w,1);imshow(image, cmap='gray');title('Original image')
subplot(h,w,2);imshow(Gaussian_filter, cmap='gray');title('Gaussian Filter')
subplot(h,w,3);imshow(median_filter, cmap='gray');title('Median Filter')
subplot(h,w,4);imshow(average_filter , cmap='gray');title('Average Filter ')
show()
```



Example 4: Edge Enhancement

Enhancing edge of an image using prewitt function (https://scikit-image.org/docs/dev/api/skimage.filters.html#skimage.filters.prewitt)

In [ ]:
```
from scipy import misc
from skimage.filters import prewitt
from pylab import show,title,figure,subplot,subplots_adjust, imshow

image  = misc.ascent()      # load input image
Edge   = prewitt (image)   # find edge

h,w=1,2                   # figure height and width
figure(figsize=(5,5));subplots_adjust(hspace=0.5, wspace=0.5)
subplot(h,w,1);imshow(image, cmap='gray');title('Original image')
subplot(h,w,2);imshow(Edge, cmap='gray');title('Edge')
show()
```



Exercise 2

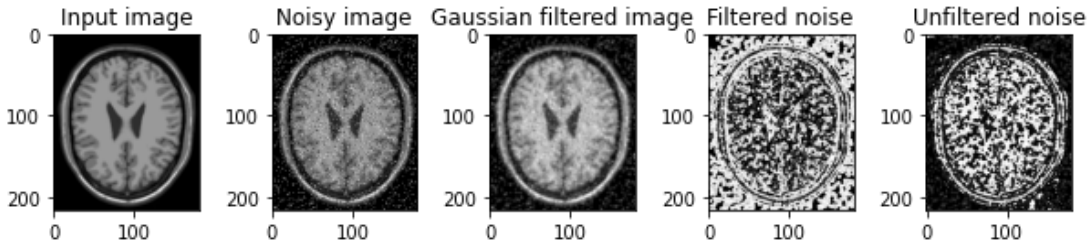
To illustrate image filtering we will try to restore an image, obtained by distorting Brain.tiff with Salt and Pepper noise (Brain_noise_SnP.tiff). In the first part of the exercise, we will focus on Gaussian smoothing. Apply Gaussian filtering (https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.gaussian_filter.html) to the noisy image with a standard deviation of 1.

- Read input image (Brain.tiff)
- Read noisy image (Brain_noise_SnP.tiff)
- Apply Gaussian filtering (https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.gaussian_filter.html) to the noisy image with a standard deviation of 1.
- Calculate the difference image of the input noisy image with the obtained filtered image.
- Calculate the difference image of the obtained filtered image with the provided ground truth (Brain.tiff).
- Create a simple edge map of the obtained filtered image using the edge function and prewitt mask (https://scikit-image.org/docs/dev/api/skimage.filters.html#skimage.filters.prewitt).
- Display input image, noisy image, gaussian filtered image, filtered noise and unfiltered noise.

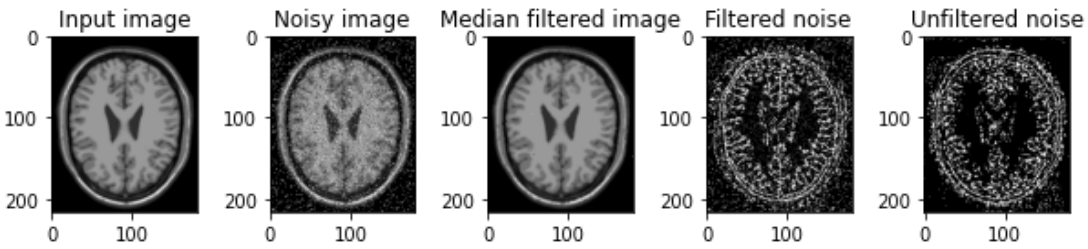
Report

- Repeat the process, for a median filter (https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.median_filter.html) for a kernel of size 3 and an average filter using a kernel of size 3. For the average filter you will have to create your own filter kernel.
- Calculate the root mean squared differences (RMSD) (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html) of the pixels of the obtained filtered images with those of the ground truth. RMSD is a frequently used measure of the differences between values. Hint: RMSD = sqrt (https://www.tutorialspoint.com/python/number_sqrt.htm) (mean_squared_error (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html) (input image, filtered image))
- Display a three-by-four plot with the filtered images obtained using the different filters, the filtered noise, the noise that remained after the filtering (unfiltered) and the edge maps of the filtered image.
- Provide all three values for the RMSD between filtered image and the ground truth.
- Comment briefly on the results.
- What is the interpretation of the difference image with the ground truth and the difference image with the original input image?
- Which filter works best in terms of RMSD and why?
- Which filter preserves the edges the best?

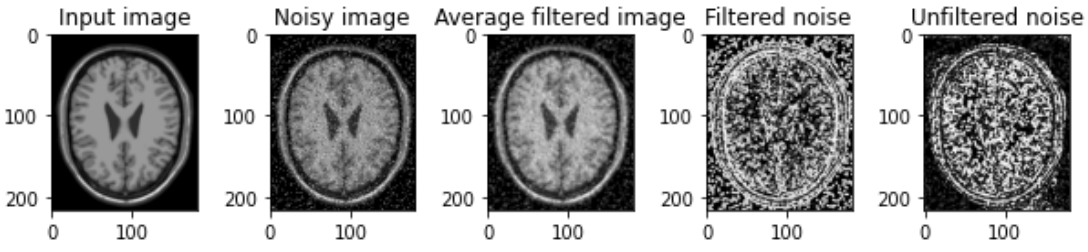
In [ ]:
```
# your code is here
from scipy import misc
from scipy import ndimage
from pylab import show,title,figure,subplot,subplots_adjust, imshow
# brain_image = imread("Brain.tiff")
# Brain_noise_SnP=imread("Brain_noise_SnP.tiff")
# h,w=1,2                    # figure height and width
# figure(figsize=(5,5));subplots_adjust(hspace=0.5, wspace=0.5)
# subplot(h,w,1);imshow(brain_image, cmap='gray');title('Original image')
# subplot(h,w,2);imshow(Brain_noise_SnP, cmap='gray');title('Noisy Image')
# show()
# Load the image
brain_image = imread("Brain.tiff")
brain_noise_image = imread("Brain_noise_SnP.tiff")
# Apply Gaussian filtering to the noisy image with a standard deviation of 1
Gaussian_filter = ndimage.gaussian_filter(brain_noise_image, sigma = 1) # gaussian filter with standard deviation = 1
# Calculate the difference image of the input noisy image with the obtained filtered image.
dif_noiseNgaussian = brain_noise_image - Gaussian_filter
dif_gaussianNtruth = Gaussian_filter - brain_image
# Create a simple edge map of the obtained filtered image
Gaussian_edge_map = prewitt(Gaussian_filter)
h,w=1,5    # figure height and width
figure(figsize=(10,10));subplots_adjust(hspace=0.2,wspace=0.5)
subplot(h,w,1);imshow(brain_image, cmap='gray');title('Input image')
subplot(h,w,2);imshow(brain_noise_image, cmap='gray');title('Noisy image')
subplot(h,w,3);imshow(Gaussian_filter, cmap='gray');title('Gaussian filtered image')
subplot(h,w,4);imshow(dif_noiseNgaussian, cmap='gray');title('Filtered noise')
subplot(h,w,5);imshow(dif_gaussianNtruth, cmap='gray');title('Unfiltered noise')
# subplot(h,w,5);imshow(Gaussian_edge_map, cmap='gray');title('Edge map')
show()
```



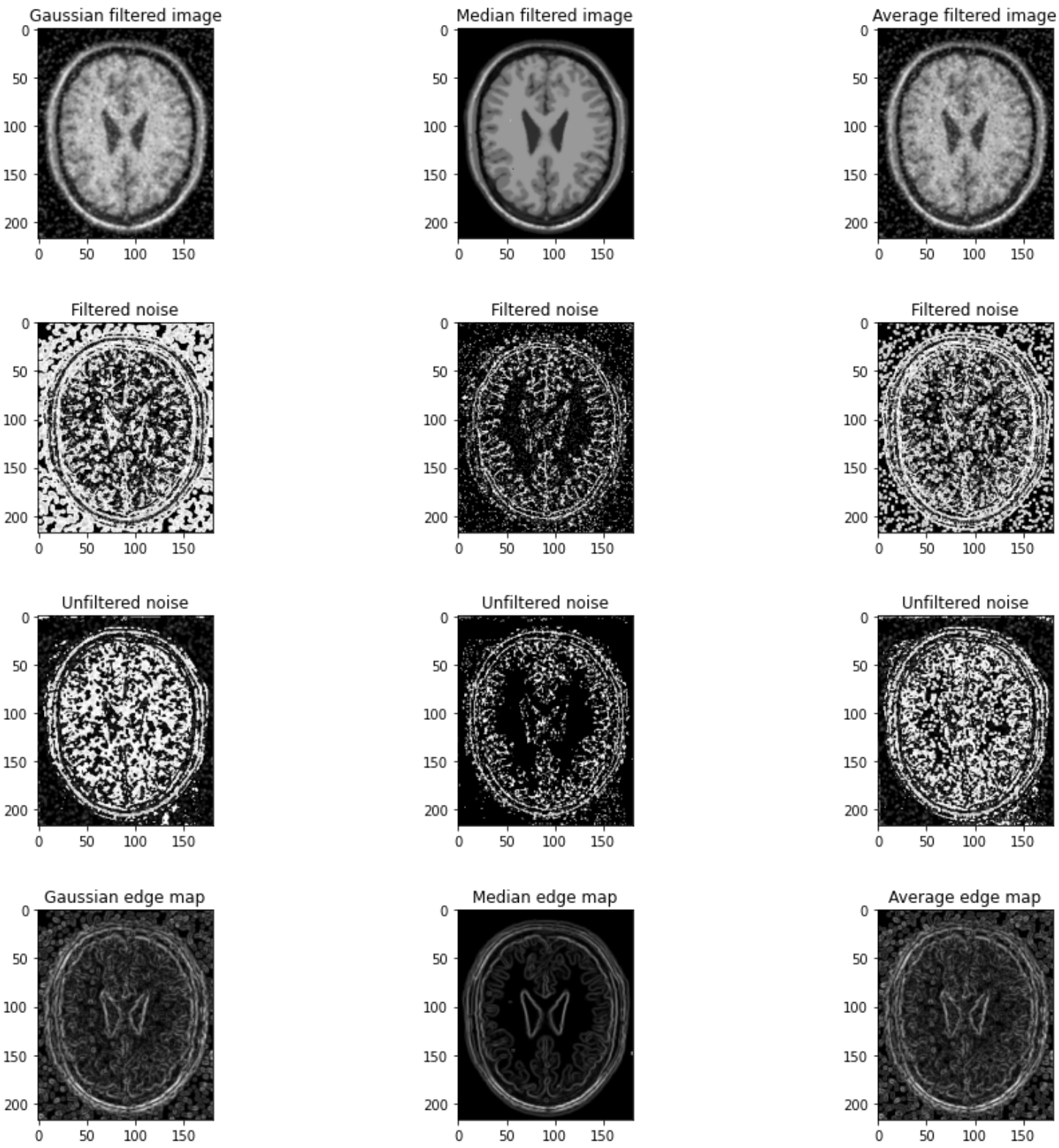
In [ ]:
```
# Median filter
median_filter = ndimage.median_filter(brain_noise_image, size = 3)    # median filter with kernel size of 3
# Calculate the difference image of the input noisy image with the obtained filtered image.
dif_noiseNmedian = brain_noise_image - median_filter
dif_medianNtruth = median_filter - brain_image
# Create a simple edge map of the obtained filtered image
median_edge_map = prewitt(median_filter)
h,w=1,5    # figure height and width
figure(figsize=(10,10));subplots_adjust(hspace=0.2,wspace=0.5)
subplot(h,w,1);imshow(brain_image, cmap='gray');title('Input image')
subplot(h,w,2);imshow(brain_noise_image, cmap='gray');title('Noisy image')
subplot(h,w,3);imshow(median_filter, cmap='gray');title('Median filtered image')
subplot(h,w,4);imshow(dif_noiseNmedian, cmap='gray');title('Filtered noise')
subplot(h,w,5);imshow(dif_medianNtruth, cmap='gray');title('Unfiltered noise')
# subplot(h,w,5);imshow(Gaussian_edge_map, cmap='gray');title('Edge map')
show()

```



In [ ]:
```
# average filter
kernel          = np.ones(shape=(3,3))/9                    # define kernel for average filter, kernel size is 3
average_filter  = ndimage.convolve (brain_noise_image, kernel)          # average filter
# Calculate the difference image of the input noisy image with the obtained filtered image.
dif_noiseNaverage = brain_noise_image - average_filter
dif_averageNtruth = average_filter - brain_image
# Create a simple edge map of the obtained filtered image
average_edge_map = prewitt(average_filter)
h,w=1,5    # figure height and width
figure(figsize=(10,10));subplots_adjust(hspace=0.2,wspace=0.5)
subplot(h,w,1);imshow(brain_image, cmap='gray');title('Input image')
subplot(h,w,2);imshow(brain_noise_image, cmap='gray');title('Noisy image')
subplot(h,w,3);imshow(average_filter, cmap='gray');title('Average filtered image')
subplot(h,w,4);imshow(dif_noiseNaverage, cmap='gray');title('Filtered noise')
subplot(h,w,5);imshow(dif_averageNtruth, cmap='gray');title('Unfiltered noise')
# subplot(h,w,5);imshow(Gaussian_edge_map, cmap='gray');title('Edge map')
show()
```



In [ ]:
```
# Calculate the root mean squared differences
from sklearn.metrics import mean_squared_error
RMSD_Gaussian = mean_squared_error(brain_image, Gaussian_filter)
RMSD_median = mean_squared_error(brain_image, median_filter)
RMSD_average = mean_squared_error(brain_image, average_filter)
h,w = 4,3    # figure height and width
figure(figsize=(15,15));subplots_adjust(hspace=0.4,wspace=0.5)
subplot(h,w,1);imshow(Gaussian_filter, cmap='gray');title('Gaussian filtered image')
subplot(h,w,2);imshow(median_filter, cmap='gray');title('Median filtered image')
subplot(h,w,3);imshow(average_filter, cmap='gray');title('Average filtered image')
subplot(h,w,4);imshow(dif_noiseNgaussian, cmap='gray');title('Filtered noise')
subplot(h,w,5);imshow(dif_noiseNmedian, cmap='gray');title('Filtered noise')
subplot(h,w,6);imshow(dif_noiseNaverage, cmap='gray');title('Filtered noise')
subplot(h,w,7);imshow(dif_gaussianNtruth, cmap='gray');title('Unfiltered noise')
subplot(h,w,8);imshow(dif_medianNtruth, cmap='gray');title('Unfiltered noise')
subplot(h,w,9);imshow(dif_averageNtruth, cmap='gray');title('Unfiltered noise')
subplot(h,w,10);imshow(Gaussian_edge_map, cmap='gray');title('Gaussian edge map')
subplot(h,w,11);imshow(median_edge_map, cmap='gray');title('Median edge map')
subplot(h,w,12);imshow(average_edge_map, cmap='gray');title('Average edge map')
show()
```



In [ ]:
```
# Provide all three values for the RMSD between filtered image and the ground truth.
print(RMSD_Gaussian)
print(RMSD_median)
print(RMSD_average)
```

```
65.62324006415967
11.593222496626526
53.091580314178785
```

Comment briefly on the results.

A: So as we can find from the results, the RMSD values of these three filters are: Gaussian > average > median, which shows the corresponding filtering abilities are: median > average > Gaussian.

What is the interpretation of the difference image with the ground truth and the difference image with the original input image?

A: The previous one shows the remaining noises that haven't been successfully removed after applying the filter. The latter one shows the noises that have been removed after the implementation of the filter.

Which filter works best in terms of RMSD and why?

A: Median filter works the best cause it gives the smallest RMSD value.

Which filter preserves the edges the best?

A: Median filter.

3 Image segmentation

Segmentation is the task of defining the boundaries of an object or region in an image. It is often used for measuring size or volume of organs or other tissues of interest. A multitude of different methods exist and the optimal choice of segmentation method is highly dependent on the region to be segmented, and the type and quality of the input image.

3.1 Thresholding

Image thresholding is the simplest and fastest segmentation method. The process comes down to defining one or more boundaries of intensity in the im- age histogram. Pixels with intensity within the boundaries will get mapped to 1 (inside), while others are considered background. The process can be extended to multiple labels using multiple (upper and lower) boundaries. Thresholding can be done by manually selecting the boundaries, or automatically, by optimizing the boundary values with respect to a certain criterion. For instance, Otsu thresholding will automatically select boundaries that maximize the between class variance of two or more regions.

3.2 Region Growing

Region growing is an iterative segmentation approach in which an initial region (usually a single seed point) is grown by including its neighbouring pixels if they fulfil certain requirements. In its simplest form, region growing is closely related to thresholding, mainly using the image intensity to drive the algorithm. The algorithm has the benefit of taking into account spatial connectivity, thereby enabling to limit the segmentation to connected regions.

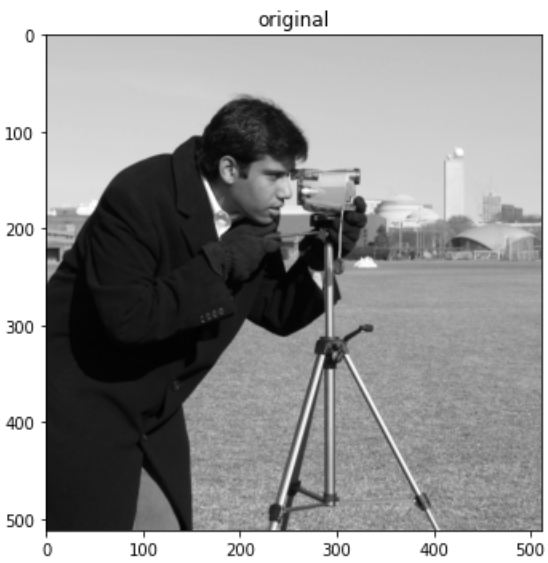
3.3 Dice Coefficient

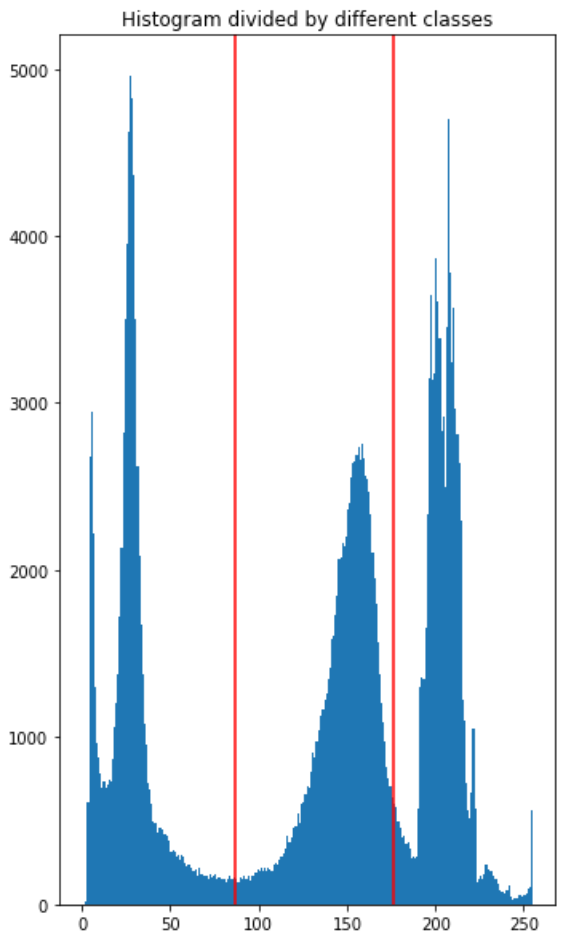
A common way to evaluate segmentations is to compare the obtained object S, with the reference or ground truth R provided by physicians after manual segmentations. A popular measure for quantitative evaluation is computing the Dice coefficient D, which compares the volumes (|.|) of the overlap of both objects to average volume,

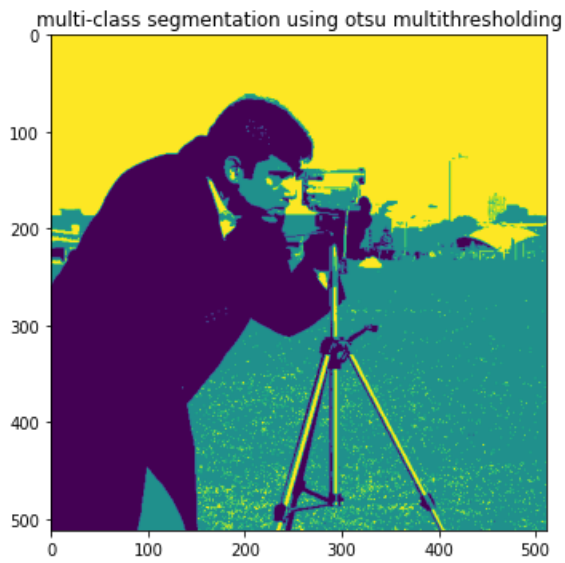
$$D(S, R) = \frac{2|S \cap R|}{|S| + |R|}$$

Example 5: [Multi-Otsu thresholding (https://scikit-image.org/docs/dev/auto_examples/segmentation/plot_multiotsu.html#sphx-glr-auto-examples-segmentation-plot-multiotsu-py)](https://scikit-image.org/docs/dev/auto_examples/segmentation/plot_multiotsu.html#sphx-glr-auto-examples-segmentation-plot-multiotsu-py)

In [ ]:
```
import numpy as np
from skimage import data
from skimage.filters import threshold_multiotsu
from pylab import show,title,figure,subplot,subplots_adjust, imshow, hist,axvline

# The input image.
image = data.camera()

# Applying multi-Otsu threshold for the default value, generating
# three classes.
thresholds = threshold_multiotsu(image, 3, nbins=255)

# Using the threshold values, we generate the three regions.
regions = np.digitize(image, bins=thresholds)

h,w=1,3                    # figure height and width
figure(figsize=(20,10));subplots_adjust(hspace=0.3, wspace=0.3)

subplot(h,w,1)
imshow(image, cmap='gray');title('original')

subplot(h,w,2)
hist(image.ravel(), bins=255, histtype = 'bar');title('Histogram divided by different classes')
for thresh in thresholds:
    axvline(thresh, color='r')

subplot(h,w,3)
imshow(regions);title('multi-class segmentation using otsu multithresholding')
show()

```







Exercise 3.1 [Multi-Otsu thresholding (https://scikit-image.org/docs/dev/auto_examples/segmentation/plot_multiotsu.html#sphx-glr-auto-examples-segmentation-plot-multiotsu-py)](https://scikit-image.org/docs/dev/auto_examples/segmentation/plot_multiotsu.html#sphx-glr-auto-examples-segmentation-plot-multiotsu-py) **for segmenting ventricles and white matter.**

By following the example 5, solve exercise 3.1:

- Read input image (Brain.tiff) using [imread (https://pypi.org/project/imread/)](https://pypi.org/project/imread/)
- Read groundtruth image for ventricals segmentation (GroundTruthVentricles.tiff)
- Read groundtruth image for white matter segmentation (grndTruthWM1.tiff)
- Apply [multi-Otsu threshold (https://scikit-image.org/docs/dev/auto_examples/segmentation/plot_multiotsu.html#sphx-glr-auto-examples-segmentation-plot-multiotsu-py)](https://scikit-image.org/docs/dev/auto_examples/segmentation/plot_multiotsu.html#sphx-glr-auto-examples-segmentation-plot-multiotsu-py) on the input image with number of classes is 4.
- Plot the histogram and the four thresholds obtained from [multi-Otsu (https://scikit-image.org/docs/dev/auto_examples/segmentation/plot_multiotsu.html#sphx-glr-auto-examples-segmentation-plot-multiotsu-py)](https://scikit-image.org/docs/dev/auto_examples/segmentation/plot_multiotsu.html#sphx-glr-auto-examples-segmentation-plot-multiotsu-py), result of multi-class segmentation using [multi-Otsu (https://scikit-image.org/docs/dev/auto_examples/segmentation/plot_multiotsu.html#sphx-glr-auto-examples-segmentation-plot-multiotsu-py)](https://scikit-image.org/docs/dev/auto_examples/segmentation/plot_multiotsu.html#sphx-glr-auto-examples-segmentation-plot-multiotsu-py) and different four regions. Hint for plotting four different regions:

```
b0 = regions==0;imshow(b0,'gray');title('Background');show();
b1 = regions==1;imshow(b1,'gray');title('Ventricles');show();
b2 = regions==2;imshow(b2,'gray');title('Gray matter');show();
b3 = regions==3;imshow(b3,'gray');title('White matter');show();
```

- Calculate Dice coeffient between segmented ventricals (b1) and groundtruth image for ventricals segmentation (GroundTruthVentricles[:,:,0])
- Calculate Dice coeffient between segmented gray matter (b3) and groundtruth image for white matter segmentation (grndTruthWM1[:,:,0])
- Dice coefficient function:

  def dice_coeff(im1,im2):

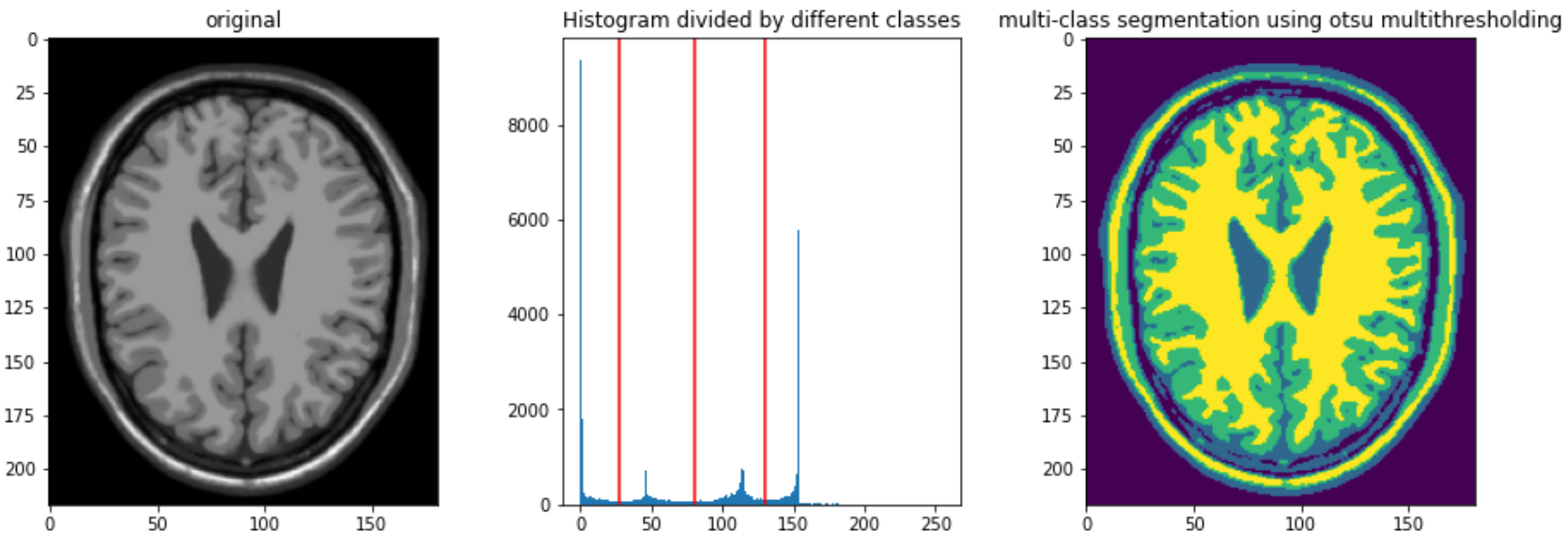
```
im1 = im1.astype(np.bool)
im2 = im2.astype(np.bool)
intersect = np.logical_and(im1,im2)
return 2*intersect.sum()/(im1.sum() + im2.sum())
```

In [ ]:
```
# your code is here
from scipy import misc
from scipy import ndimage
from pylab import show,title,figure,subplot,subplots_adjust, imshow


Brain_image = imread("Brain.tiff")
GroundTruthVentricles_image=imread("GroundTruthVentricles.tiff")
grndTruthWM1_image=imread("grndTruthWM1.tiff")

h,w=1,3                    # figure height and width
figure(figsize=(15,5));subplots_adjust(hspace=0.5, wspace=0.5)
subplot(h,w,1);imshow(Brain_image, cmap='gray');title('Brain image')
subplot(h,w,2);imshow(GroundTruthVentricles_image, cmap='gray');title('GroundTruthVentricles Image')
subplot(h,w,3);imshow(grndTruthWM1_image, cmap='gray');title('grndTruthWM1 Image')
show()


```

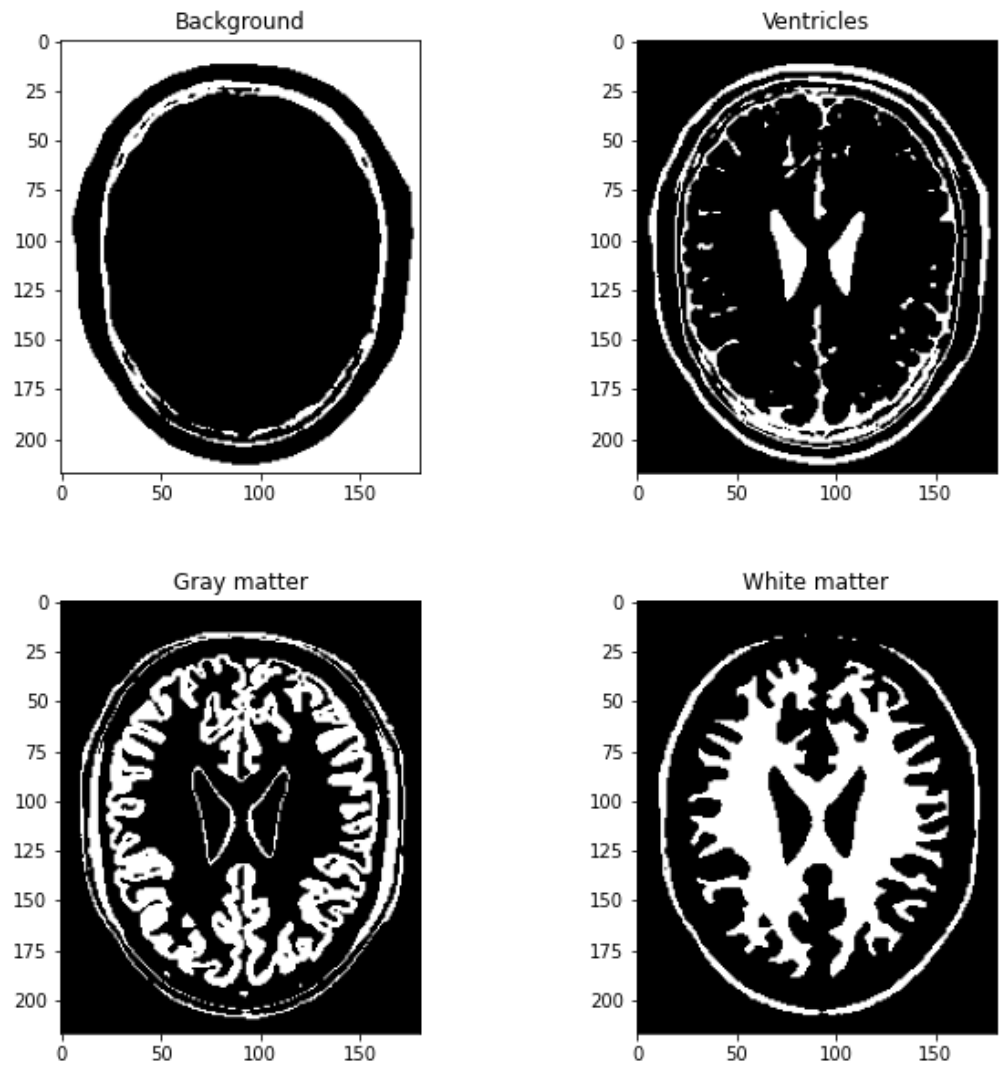
In [ ]:
```
# import numpy as np
# from skimage import data
# from skimage.filters import threshold_multiotsu
# from pylab import show,title,figure,subplot,subplots_adjust, imshow, hist,axvline

# # The input image.
# image = data.camera()

# Applying multi-Otsu threshold for the default value, generating
# four classes.
thresholds = threshold_multiotsu(Brain_image, 4, nbins=255)
print(thresholds)
```

```
[ 27  80 130]
```

In [ ]:
```
# Using the threshold values, we generate the three regions.
regions = np.digitize(Brain_image, bins=thresholds)

h,w=1,3                    # figure height and width
figure(figsize=(15,5));subplots_adjust(hspace=0.3, wspace=0.3)

subplot(h,w,1)
imshow(Brain_image, cmap='gray');title('original')

subplot(h,w,2)
hist(Brain_image.ravel(), bins=255, histtype = 'bar');title('Histogram divided by different classes')
for thresh in thresholds:
    axvline(thresh, color='r')

subplot(h,w,3)
imshow(regions);title('multi-class segmentation using otsu multithresholding')
show()
```



In [ ]:
```
h,w=2,2
figure(figsize=(10,10));subplots_adjust(hspace=0.3, wspace=0.3)
subplot(h,w,1)
b0 = regions==0;imshow(b0,'gray');title('Background');
subplot(h,w,2)
b1 = regions==1;imshow(b1,'gray');title('Ventricles');
subplot(h,w,3)
b2 = regions==2;imshow(b2,'gray');title('Gray matter');
subplot(h,w,4)
b3 = regions==3;imshow(b3,'gray');title('White matter');
```



In [ ]:
```
#Dice coefficient function:
def dice_coeff(im1,im2):

    im1 = im1.astype(np.bool)
    im2 = im2.astype(np.bool)
    intersect = np.logical_and(im1,im2)
    return 2*intersect.sum()/(im1.sum() + im2.sum())
```

In [ ]:
```
import warnings
warnings.filterwarnings("ignore")

dice_coeff1=dice_coeff(b1,GroundTruthVentricles_image[:,:,0])
# (['Ventricles','Multi-otsu',dice_coeff1])
print("The Dice coeffient between segmented ventricals and groundtruth image for ventricals segmentation is:")
print(dice_coeff1)
```

```
The Dice coeffient between segmented ventricals and groundtruth image for ventricals segmentation is:
0.2424980959634425
```

In [ ]:
```
dice_coeff2=dice_coeff(b2,grndTruthWM1_image[:,:,0])
# (['White Matter','Multi-otsu',dice_coeff2])
print("The Dice coeffient between segmented gray matter and groundtruth image for white matter segmentation is:")
print(dice_coeff2)
```

```
The Dice coeffient between segmented gray matter and groundtruth image for white matter segmentation is:
0.16550389570708357
```

Exercise 3.2 Region Growing Segmentation

You should note that in the previous exercise it was not possible to separate the ventricles or white matter from some other structures completely. In this exercise will attempt to do this by implementing a region growing algorithm in its simplest form. The algorithm should output a binary image with pixel values 1 for the structure under study and 0 for all other pixels. As an input it should use one or two threshold values that were obtained from the previous exercise and a seed point.

To select your seeds for the region growing algorithm, inspect the image and find a point coordinate (X,Y) which later will be used as a seed point for your segmentation algorithm. A chosen point location has to be within the structure you are planning to segment.

Using the implemented algorithm, try to segment each of the ventricles and the white matter using suitable seed points and calculate the Dice coefficients with respect to the ground truth images introduced above.

Ventricles segmentation

- Step 1. Read input image (Brain.tiff)
- Step 2. Read groundtruth image for ventricals segmentation (GroundTruthVentricles.tiff)

- Step 3. Apply multi-Otsu threshold (https://scikit-image.org/docs/dev/auto_examples/segmentation/plot_multiotsu.html#sphx-glr-auto-examples-segmentation-plot-multiotsu-py) on the input image with number of classes is 4.
- For the first ventricle segmentation
- Step 4. Choose seed point. There are two ventricles, therefore, user has to repeat the process of choosing seed point twice.
- Step 5. Apply region growing function using selected seed point, calculated threshold, and connectivity = 4.
- For the second ventricle segmentation
- Step 6. Repeat steps 4 and 5
- Step 7. Get final segmentation by adding first and second ventricle segmentations.
- Step 8. Calculate Dice coefficient between final ventricles segmentation and groundtruth (GroundTruthVentricles[:,:,0]).

White matter segmentation

- Step 1. Read input image (Brain.tiff)
- Step 2. Read groundtruth image for white matter segmentation (grndTruthWM1.tiff)
- Step 3. Apply multi-Otsu threshold (https://scikit-image.org/docs/dev/auto_examples/segmentation/plot_multiotsu.html#sphx-glr-auto-examples-segmentation-plot-multiotsu-py) on the input image with number of classes is 4.
- Step 4. Choose seed point.
- Step 5. Apply region growing function using selected seed point, calculated threshold, and connectivity = 4.
- Step 6. Calculate Dice coefficient between final ventricles segmentation and groundtruth (grndTruthWM1[:,:,0]).

Improving results

Improve you segmentation region growing results - both for the ventricles and white matter - using morphological operations. Sci-kit image - morphology (https://scikit-image.org/docs/dev/api/skimage.morphology.html)

Report

- For Exercises 3.1 and 3.2 plot a 2-by-3 figure of the segmentations of the ventricles and white matter (after thresholding, region growing and region growing followed by morphological operations) and their corresponding ground truth.
- Provide a table of the obtained Dice coefficients for each method and each structure.
- Briefly comment on the obtained results for the segmentations and corresponding measures.
- What morphological methods did you use? Why?

Hints on writing your own region growing function:

To build this function:

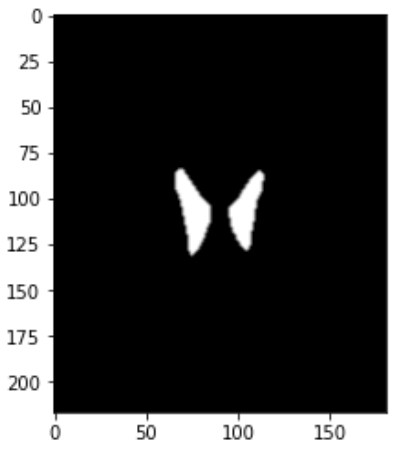
- Start from the seed point
- List the 4 (or 8) neighboring pixels
- Check if their intensity falls within the threshold boundaries.
- Grow your region by adding the pixels that meet the condition.
- List all new neighboring pixels of the obtained new region.
- Repeat until there are no more pixels added.

It may be handy to store the indexes (locations) and values of pixels which are already marked inside and those, which are currently marked as neighbors. For example: 0 - outside, 1 - inside, 2 - neighbor.

In [ ]:
```
#Ventricles segmentation
#step- 1,2,3

from scipy import misc
from scipy import ndimage
from pylab import show,title,figure,subplot,subplots_adjust, imshow

#Step 1
Brain_image=imread("Brain.tiff")

#Step 2
GroundTruthVentricles_image=imread("GroundTruthVentricles.tiff")

#Step 3
from skimage.filters import threshold_multiotsu
# Applying multi-Otsu threshold for the default value, generating
# four classes.
thresholds = threshold_multiotsu(Brain_image, 4, nbins=255)

# Using the threshold values, we generate the three regions.
regions = np.digitize(Brain_image, bins=thresholds)

print(thresholds)

# figure(figsize=(5,5))
# b1 = regions==1;imshow(b1,'gray');title('Ventricles');
# print(Brain_image.shape)
# print(GroundTruthVentricles_image.shape)
# What we should get after segmentation
imshow(GroundTruthVentricles_image[:,:,0] , cmap='gray')

```

```
[ 27  80 130]
```

Out[24]: <matplotlib.image.AxesImage at 0x7fec494f3cd0>



Step 5. Apply region growing function using selected seed point, calculated threshold, and connectivity = 4.

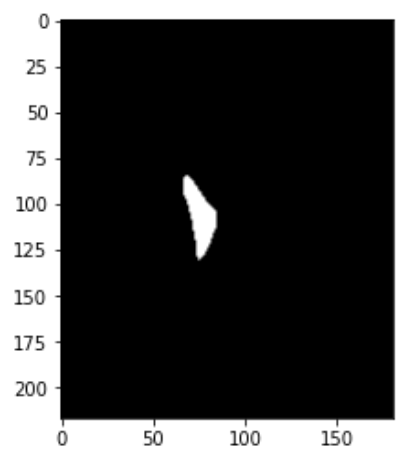
In [ ]:

```python
def region_growing(img, seed, threshold, conn):
    dims = img.shape


    # Connectivity
    if conn == 2:
        orient = [(1, 0), (0, 1)] # 2 connectivity
    elif conn == 4:
        orient = [(1, 0), (0, 1), (-1, 0), (0, -1)] # 4 connectivity
    elif conn == 8:
        orient = [(1, 0), (1, 1), (0, 1), (-1, 1), (-1, 0), (-1, -1), (0, -1), (1, -1)] # 8 connectivity

    reg = np.zeros(dims)


    #parameters
    mean_reg = float(img[seed[1], seed[0]])
    size = 1
    pix_area = dims[0]*dims[1]

    contour = [] # will be [ [[x1, y1], val1],..., [[xn, yn], valn] ]
    contour_val = []
    dist = 0

    cur_pix = [seed[0], seed[1]]

    #Spreading
    while(dist<threshold and size<pix_area):
    #adding pixels
        for j in range(len(orient)):
            #select new candidate
            temp_pix = [cur_pix[0] +orient[j][0], cur_pix[1] +orient[j][1]]

            #check if it belongs to the image
            is_in_img = dims[0]>temp_pix[0]>=0 and dims[1]>temp_pix[1]>=0 #returns boolean
            #candidate is taken if not already selected before
            if (is_in_img and (reg[temp_pix[1], temp_pix[0]]==0)):
                contour.append(temp_pix)
                contour_val.append(img[temp_pix[1], temp_pix[0]] )
                reg[temp_pix[1], temp_pix[0]] = 2
        #add the nearest pixel of the contour in it
        dist_list = [abs(i - mean_reg) for i in contour_val ]
        dist = min(dist_list)    #get min distance
        index = dist_list.index(min(dist_list)) #mean distance index
        size += 1 # updating region size
        reg[cur_pix[1], cur_pix[0]] = 255

        #updating mean MUST BE FLOAT
        mean_reg = (mean_reg*size + float(contour_val[index]))/(size+1)
        #updating seed
        cur_pix = contour[index]

        #removing pixel from neigborhood
        del contour[index]
        del contour_val[index]

    return reg
```

In [ ]:

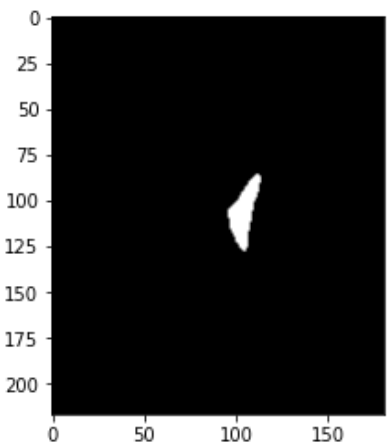
```python
seed_point1 = [75,100]
seg1 = region_growing(Brain_image, seed_point1, thresholds[0], conn=4)
imshow(seg1,'gray')
```

Out[37]: <matplotlib.image.AxesImage at 0x7fec454aa760>



In [ ]:
```
seed_point2 = [105,100]
seg2 = region_growing(Brain_image, seed_point2, thresholds[0], conn=4)
imshow(seg2,'gray')
```

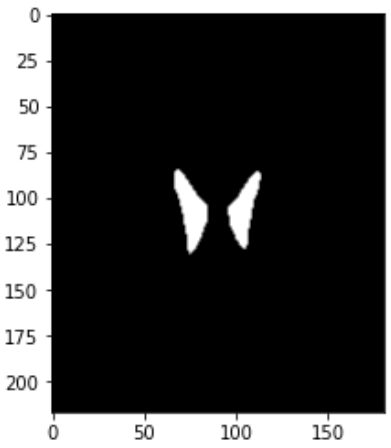
Out[38]: <matplotlib.image.AxesImage at 0x7fec48b43be0>



Step 6. Calculate Dice coefficient between final ventricles segmentation and groundtruth (grndTruthWM1[:,:,0]).

In [ ]:
```
final_seg=seg1+seg2
imshow(final_seg,'gray')
```

Out[39]: <matplotlib.image.AxesImage at 0x7fec45f9dc70>



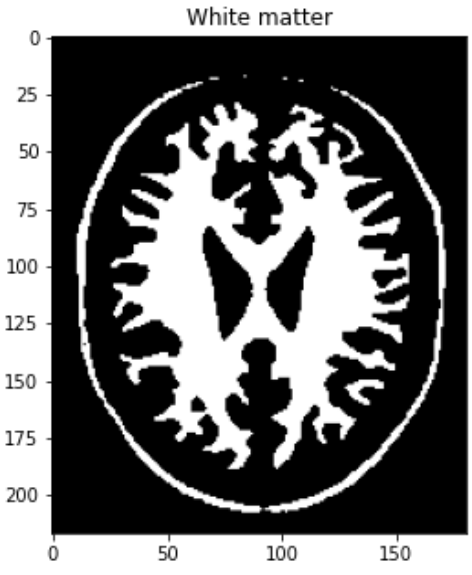
In [ ]:
```
dice_coeff3=dice_coeff(final_seg,GroundTruthVentricles_image[:,:,0])
# (['White Matter','Multi-otsu',dice_coeff2])
#print("The Dice coeffient between segmented gray matter and groundtruth image for white matter segmentation is:")
print(dice_coeff3)

#dice_coeff3=dice_coeff(final_seg,GroundTruthVentricles_image[:,:])
#data.append(['Ventricles','Region Growing',dice_coeff3])
```

0.9959100204498977

White matter Segmentation

In [ ]:
```
from scipy import misc
from scipy import ndimage
from pylab import show,title,figure,subplot,subplots_adjust, imshow

#Step 1
Brain_image=imread("Brain.tiff")

#Step 2
grndTruthWM1_image=imread("grndTruthWM1.tiff")
#Step 3
from skimage.filters import threshold_multiotsu
# Applying multi-Otsu threshold for the default value, generating
# four classes.
thresholds = threshold_multiotsu(Brain_image, 4, nbins=255)

# Using the threshold values, we generate the three regions.
regions = np.digitize(Brain_image, bins=thresholds)
```

In [ ]:
```
figure(figsize=(5,5))
b3 = regions==3;imshow(b3,'gray');title('White matter');

```

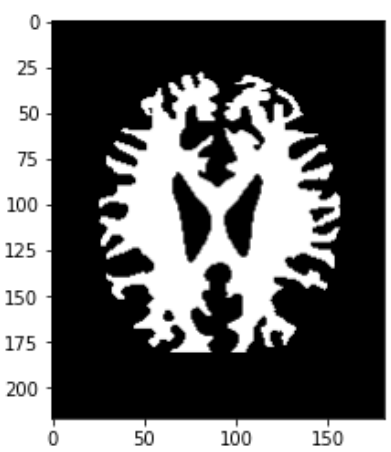


In [ ]:
```
# print(Brain_image)
# print(thresholds[2])
# print(thresholds[1])
print(thresholds)
```

```
[ 27  80 130]
```

In [ ]:
```
seed_point3 = (50, 110)
# seg3 = region_growing(Brain_image, seed3, 130, conn=4)
seg3 = region_growing(Brain_image, seed_point3, thresholds[0], conn=4)
imshow(seg3,'gray')
```

Out[42]: <matplotlib.image.AxesImage at 0x7fec48a9d340>



In [ ]:
```
# seed3=(90,125)
# seg3 = region_growing(Brain_image, seed3, 130, conn=4)
# imshow(seg3,'gray')
```

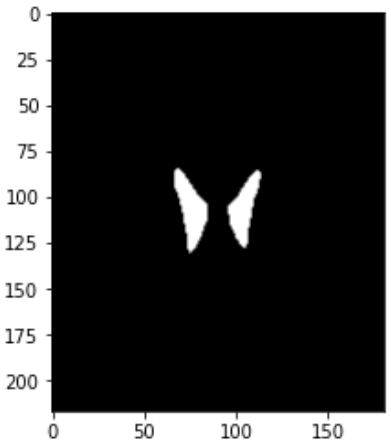
In [ ]:
```
dice_coeff4=dice_coeff(seg3,grndTruthWM1_image[:,:,0])
print(dice_coeff4)
# data.append(['White Matter','Region Growing',dice_coeff4])
```

```
0.9398916886720673
```

Improving results

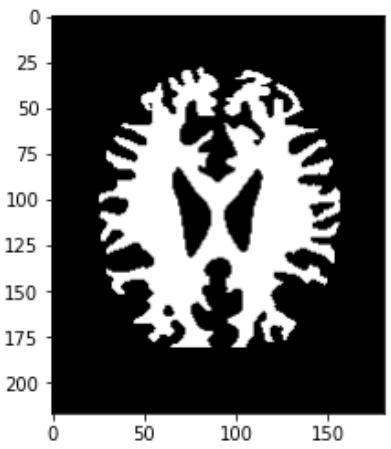
In [ ]:
```
from skimage import morphology
mor_ventrcles = morphology.area_opening(final_seg)
imshow(mor_ventrcles,'gray')
```

Out[46]: <matplotlib.image.AxesImage at 0x7fec3de0b400>



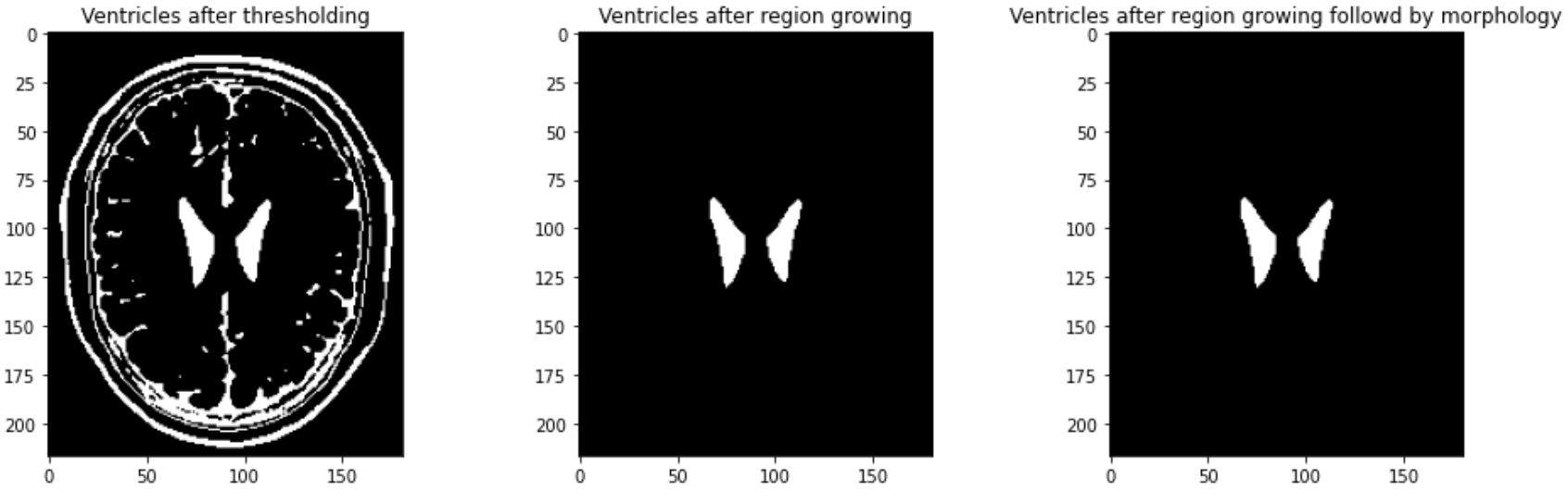
In [ ]:
```
mor_WM = morphology.area_opening(seg3)
imshow(mor_WM,'gray')
```

Out[47]: <matplotlib.image.AxesImage at 0x7fec3dd236d0>



Report

In [ ]:
```
h,w = 2,3    # figure height and width
figure(figsize=(15,15));subplots_adjust(hspace=0.4,wspace=0.5)
subplot(h,w,1);imshow(b1,'gray');title('Ventricles after thresholding')
subplot(h,w,2);imshow(final_seg,'gray');title('Ventricles after region growing')
subplot(h,w,3);imshow(mor_ventrcles,'gray');title('Ventricles after region growing followd by morphology')
subplot(h,w,4);imshow(b3,'gray');title('White matter after thresholding')
subplot(h,w,5);imshow(seg3,'gray');title('White matter after region growing')
subplot(h,w,6);imshow(mor_WM,'gray');title('White matter after region growing followd by morphology')
show()
```



In [ ]:
```
import matplotlib.pyplot as plt
data = [[dice_coeff1, dice_coeff2],[dice_coeff3,dice_coeff4]]
columns = ('Ventricles','White Matter')
rows = ['Multi-Otsu thresholding','Region growthing']

fig, ax = plt.subplots()
ax.set_axis_off()
the_table = ax.table(cellText=data,rowLabels=rows,
                     colLabels=columns,cellLoc='center',
                     loc='upper left')
ax.set_title('Dice coefficients for each method and each structure')
plt.show()
```

Dice coefficients for each method and each structure

	Ventricles	White Matter
Multi-Otsu thresholding	0.2424980959634425	0.16550389570708357
Region growthing	0.9959100204498977	0.9398916886720673

Briefly comment on the obtained results for the segmentations and corresponding measures.

Compared to white matter, ventricles have higher dice coefficients. Because white matter has more complex structure. Compared to Multi-Otsu thresholding method, region growthing function has better results. Because for Multi-Otsu thresholding, some parts which share similar intensities with the segmentation part are also involved.

What morphological methods did you use? Why?

We used closing method, because we want to keep the small structures, make the whole structure more solid.