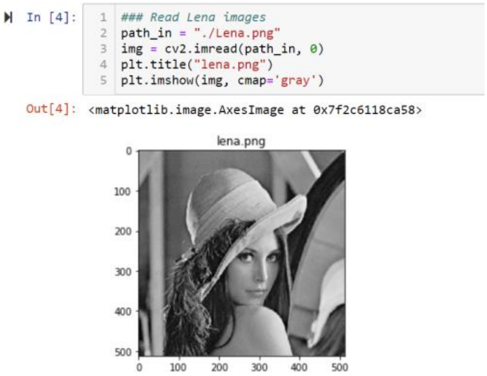
11849060 谭树杰

Lab 5

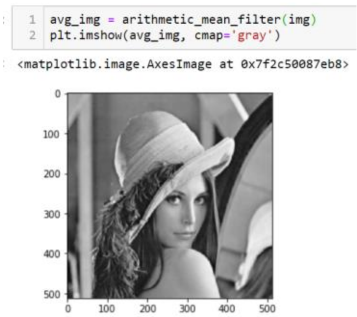
Lab 5-1 Image input and output

1. Read Lena images.

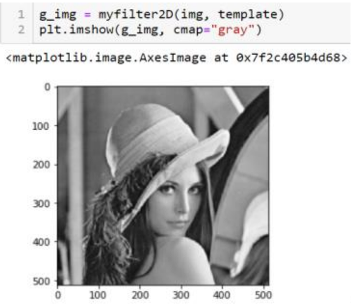


```
In [4]:
### Read Lena images
path_in = "./Lena.png"
img = cv2.imread(path_in, 0)
plt.title("lena.png")
plt.imshow(img, cmap='gray')

Out[4]: <matplotlib.image.AxesImage at 0x7f2c6118ca58>
```

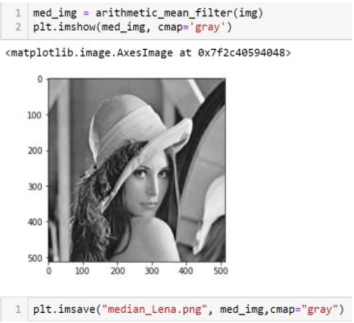
2. After reading in image, use averaging filer, Gaussian filter, 3x3 median filter to process the image matrices (note: you need write your own algorithms for these filters. Finally, the original Lena image is a color image, if you want, you can process the color image)
   The required processed images are shown below, corresponding code can be found in Lab 5 – Application to Image Segmentation.ipynb. Saved images are named avg_Lena.png, g_Lena.png, median_Lena.png correspondly.



```
avg_img = arithmetic_mean_filter(img)
plt.imshow(avg_img, cmap='gray')

<matplotlib.image.AxesImage at 0x7f2c50087eb8>
```



```
g_img = myfilter2D(img, template)
plt.imshow(g_img, cmap="gray")

<matplotlib.image.AxesImage at 0x7f2c405b4d68>
```

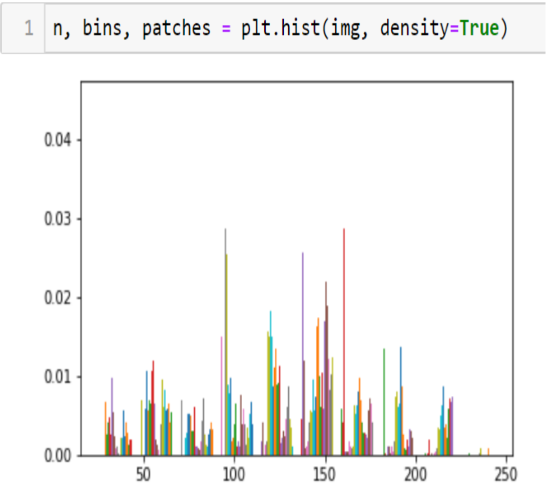


```
med_img = arithmetic_mean_filter(img)
plt.imshow(med_img, cmap='gray')

<matplotlib.image.AxesImage at 0x7f2c40594048>
```

```
plt.imsave("median_Lena.png", med_img,cmap="gray")
```

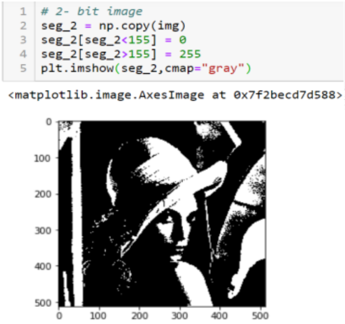
Lab 5-2 Probability density functions

1. Consider image pixel value function as a random variable, generate its probability density function – actually it is the normalized histogram, draw this probability function



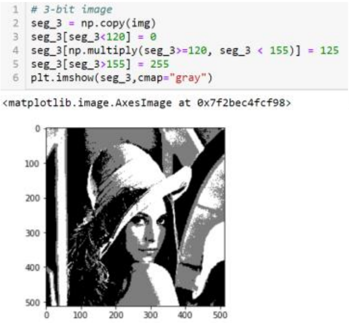
```
n, bins, patches = plt.hist(img, density=True)
```

2. Use this function to classify the image pixels into one binary image, 2- bit image, 3-bit image, 4-bit image. That is, classifying image pixels into different classes



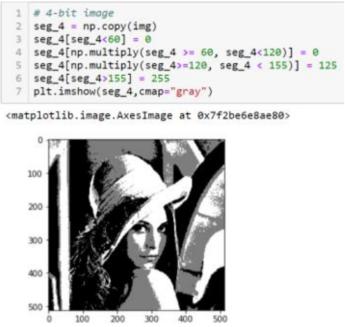
```
# 2- bit image
seg_2 = np.copy(img)
seg_2[seg_2<155] = 0
seg_2[seg_2>155] = 255
plt.imshow(seg_2,cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f2becd7d588>
```



```
# 3-bit image
seg_3 = np.copy(img)
seg_3[seg_3<120] = 0
seg_3[np.multiply(seg_3>=120, seg_3 < 155)] = 125
seg_3[seg_3>155] = 255
plt.imshow(seg_3,cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f2bec4fcf98>
```



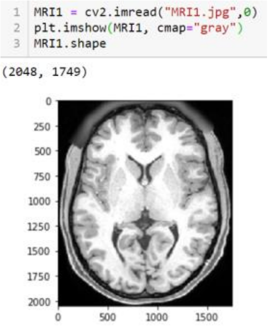
```
# 4-bit image
seg_4 = np.copy(img)
seg_4[seg_4<60] = 0
seg_4[np.multiply(seg_4 >= 60, seg_4<120)] = 0
seg_4[np.multiply(seg_4>=120, seg_4 < 155)] = 125
seg_4[seg_4>155] = 255
plt.imshow(seg_4,cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f2be6e8ae80>
```

.

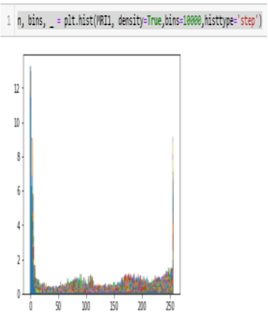
Lab 5-3 Application of Gaussian Models

1. Segmentation of MRI images using mixed Gaussian models for provided data sets – you need to determine the number of Gaussian models, mean, and variance for each Gaussian model, also use color for different categories.

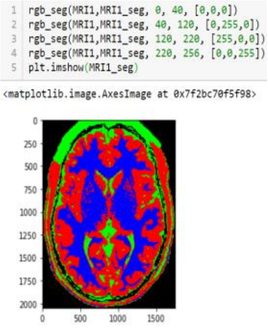


```
MRI1 = cv2.imread("MRI1.jpg",0)
plt.imshow(MRI1, cmap="gray")
MRI1.shape
```

```
(2048, 1749)
```

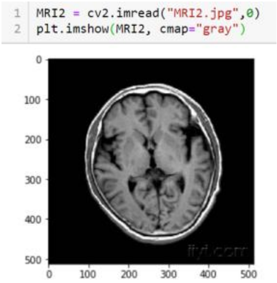


```
n, bins, _ = plt.hist(MRI1, density=True,bins=10000,histtype='step')
```

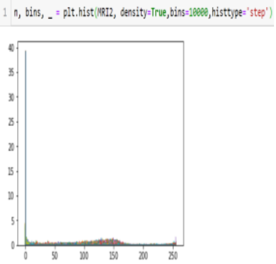


```
rgb_seg(MRI1,MRI1_seg, 0, 40, [0,0,0])
rgb_seg(MRI1,MRI1_seg, 40, 120, [0,255,0])
rgb_seg(MRI1,MRI1_seg, 120, 220, [255,0,0])
rgb_seg(MRI1,MRI1_seg, 220, 256, [0,0,255])
plt.imshow(MRI1_seg)
```

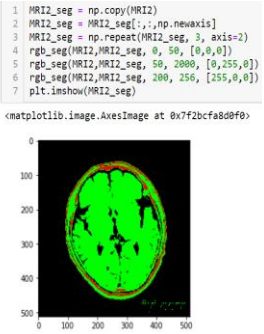
```
<matplotlib.image.AxesImage at 0x7f2bc70f5f98>
```



```
MRI2 = cv2.imread("MRI2.jpg",0)
plt.imshow(MRI2, cmap="gray")
```



```
n, bins, _ = plt.hist(MRI2, density=True,bins=10000,histtype='step')
```



```
MRI2_seg = np.copy(MRI2)
MRI2_seg = MRI2_seg[:,:,np.newaxis]
MRI2_seg = np.repeat(MRI2_seg, 3, axis=2)
rgb_seg(MRI2,MRI2_seg, 0, 50, [0,0,0])
rgb_seg(MRI2,MRI2_seg, 50, 2000, [0,255,0])
rgb_seg(MRI2,MRI2_seg, 200, 256, [255,0,0])
plt.imshow(MRI2_seg)
```

```
<matplotlib.image.AxesImage at 0x7f2bcfa8d0f0>
```