Libraries

```
import numpy as np
import h5py
import matplotlib.pyplot as plt
from math import sqrt, pi
from sklearn.cluster import KMeans
```

```
C:\ProgramData\Anaconda3\lib\site-packages\h5py\__init__.py:36:
FutureWarning: Conversion of the second argument of issubdtype from
`float` to `np.floating` is deprecated. In future, it will be treated
as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
```

Read

```
f = h5py.File('../data/assignmentSegmentBrainGmmEmMrf.mat','r')
list(f.keys())
imageData = f.get('imageData')
imageMask = f.get('imageMask')
imageData = np.array(imageData)
imageData = imageData.T
imageMask = np.array(imageMask)
imageMask = imageMask.T
```

GMM With EM

```
def label_priors(input_label,old_labels,beta,map_left,
    map_right,map_top,map_bottom,mask):
    img = input_label
    if len(input_label)==1:
        img = input_label*np.ones((old_labels.shape))
    top = ((img-np.roll(old_labels,[1,1], [0,1]))*map_top)!=0
    bottom = ((img-np.roll(old_labels,[-1,1], [0,1]))*map_bottom)!=0
    left = ((img-np.roll(old_labels,[1,2], [0,1]))*map_left)!=0
    right = ((img-np.roll(old_labels,[-1,2], [0,1]))*map_right)!=0
    return np.exp(-((top+bottom+left+right)*beta))*mask

def memberships( y,means,sigmas,old_labels,mask,prior_val):
    K = means.shape[0]
    likelihood = np.zeros((y.shape[0], y.shape[1], 3))
    prior = np.zeros((y.shape[0], y.shape[1], 3))
    for i in range(K):
        likelihood[:,:,i] = ((1/(sigmas[i,0]*sqrt(2*pi)))*np.exp(-(y-means[i,0])**2/(2*sigmas[i,0]**2)))*mask
        prior[:,:,i] = prior_val(np.array([i]),old_labels)
    norm = np.sum(prior,2)
    for i in range(K):
        prior[:,:,i] /=norm
    membership = likelihood*prior
    norm = np.sum(membership,2)
    for i in range(K):
        membership[:,:,i] /= norm

    temp = np.zeros((old_labels.shape))
    for i in range(K):
        temp = membership[:,:,i]
        temp[mask==0] = 0
        membership[:,:,i] = temp

    return membership

def gaussian_paprameters(y, mem, maps ):
    means = np.zeros((mem.shape[2],1))
    sigmas = np.zeros((mem.shape[2],1))
    for i in range(mem.shape[2]):
        den = np.sum(mem[:, :, i])
        means[i,0] = np.sum(mem[:, :, i]*y)/den
        sigmas[i,0] = np.sqrt(np.sum(mem[:,:,i]*((y-means[i,0])**2)*maps)/den)

    return means, sigmas
```

```
def posterior_val(old_labels, y, means,sigmas, maps, prior_val):
    likelihood = np.zeros((old_labels.shape))
    for i in range(len(means)):
        indeold_labels = np.where(old_labels==i)
        likelihood[indeold_labels] = (1/(sigmas[i,0]*sqrt(2*pi)))*np.exp(-(y[indeold_labels]-means[i,0])**2/(2*(sig
mas[i,0]**2)))
    prior = prior_val(old_labels,old_labels)
    return likelihood*prior*maps

def segmentation( old_labels,y,means,sigmas,iters,mask,prior_val):
    for i in range(iters):
        oldLogPosterior = np.sum(np.log(posterior_val(old_labels,y,means,sigmas,mask,prior_val)[mask!=0]))
        print('%d : Initial log posterior = %f\n'%(i,oldLogPosterior))
        membership = memberships(y,means,sigmas,old_labels,mask,prior_val)
        new_labels = np.argmax(membership,2)
        new_labels = new_labels*mask
        posterior = posterior_val(new_labels,y,means,sigmas,mask,prior_val)
        newLogPosterior = np.sum(np.log(posterior[mask!=0]))
        print('%d : Final log posterior = %f\n'%(i,newLogPosterior))

        if newLogPosterior<oldLogPosterior:
            break
        means,sigmas = gaussian_paprameters(y,membership,mask)
        equal = np.array_equal(old_labels, new_labels)
        if equal:
            break
        old_labels = new_labels

    return old_labels,means,sigmas,iters
```

Label initialization

```
s = imageData.shape
K = 3
map_left = np.roll(imageMask, [1,2], [0,1])
map_right = np.roll(imageMask, [-1,2], [0,1])
map_top = np.roll(imageMask, [1,1], [0,1])
map_bottom = np.roll(imageMask, [-1,1], [0,1])

beta1 = 2
beta2 = 0

prior1 = lambda input_label,old_labels: label_priors(
    input_label,old_labels,beta1,map_left,map_right,
    map_top,map_bottom,imageMask)

prior2 = lambda input_label,old_labels: label_priors(
    input_label,old_labels,beta2,map_left,map_right,
    map_top,map_bottom,imageMask)


masked_img = imageData[imageMask>0]
masked_img = np.reshape(masked_img, (masked_img.shape[0], 1))
kmeans = KMeans(n_clusters = K).fit(masked_img)
initial_labels = kmeans.labels_
means_init = kmeans.cluster_centers_
label_map = np.zeros((imageData.shape))
label_map[imageMask>0] = initial_labels
```

Variance

```
sigmas_init = np.zeros((K,1))
for i in range(K):
    clusterVals = masked_img[initial_labels==i]
    sigmas_init[i] = np.linalg.norm(clusterVals - means_init[i])/sqrt(len(clusterVals))
```

Segmentation

```
initial_labels = label_map
print('Modified ICM with beta = %f \n'%(beta1))
labels1,means1,sigmas1,iters1 = segmentation(initial_labels,imageData,means_init,
```

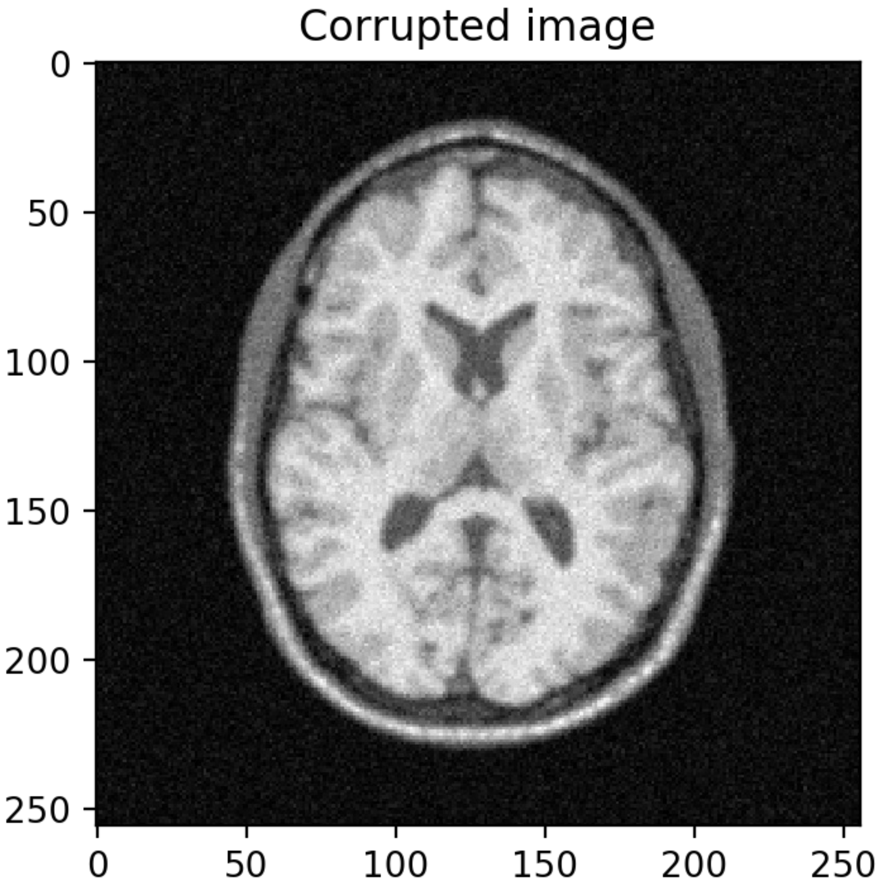
```
    sigmas_init,20,imageMask,prior1)

print('Modified ICM with beta = %f \n'%(beta2))
labels2,means2,sigmas2,iters2 = segmentation(initial_labels,imageData,means_init,
    sigmas_init,20,imageMask,prior2)
```

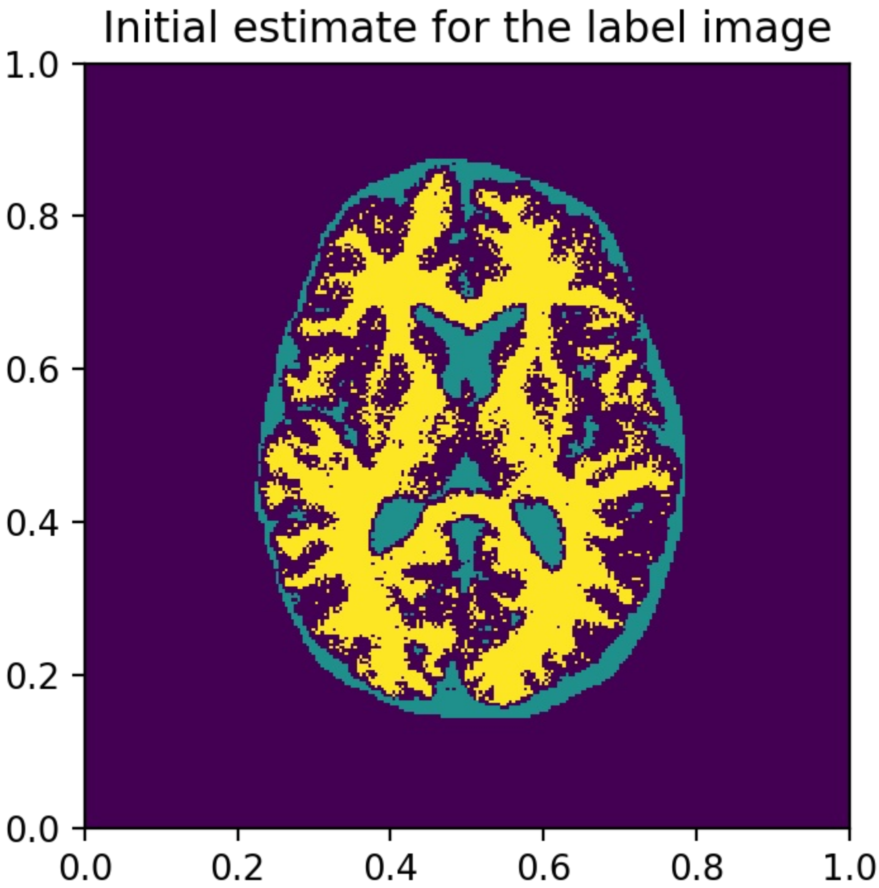
```
Modified ICM with beta = 2.000000

0 : Initial log posterior = 15919.907377

0 : Final log posterior = 18086.013949

1 : Initial log posterior = 18198.711865

1 : Final log posterior = 18697.549658

2 : Initial log posterior = 18714.199795

2 : Final log posterior = 18726.980467

3 : Initial log posterior = 18736.833042

3 : Final log posterior = 18746.036075

4 : Initial log posterior = 18756.510109
C:\ProgramData\Anaconda3\Scripts\pypublish:21: RuntimeWarning: invalid
value encountered in true_divide

4 : Final log posterior = 18630.244260

Modified ICM with beta = 0.000000

0 : Initial log posterior = 35901.907377

0 : Final log posterior = 36296.718625

1 : Initial log posterior = 36396.162509

1 : Final log posterior = 36546.829310

2 : Initial log posterior = 36503.007250

2 : Final log posterior = 36594.082126

3 : Initial log posterior = 36528.833496

3 : Final log posterior = 36585.329633

4 : Initial log posterior = 36497.443716

4 : Final log posterior = 36530.268448

5 : Initial log posterior = 36432.320468

5 : Final log posterior = 36455.949582

6 : Initial log posterior = 36360.936811

6 : Final log posterior = 36372.850988

7 : Initial log posterior = 36288.664975

7 : Final log posterior = 36295.230723

8 : Initial log posterior = 36229.516286

8 : Final log posterior = 36232.930293

9 : Initial log posterior = 36185.176518

9 : Final log posterior = 36186.652464

10 : Initial log posterior = 36153.920895

10 : Final log posterior = 36154.975846

11 : Initial log posterior = 36133.963296
```

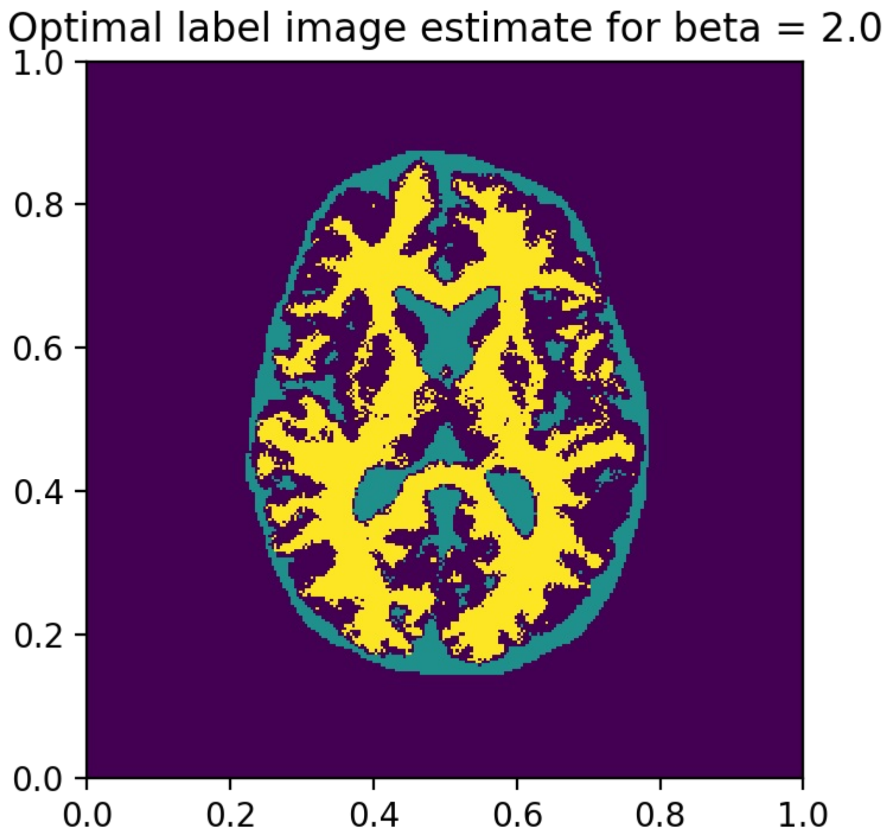
```
11 : Final log posterior = 36134.595021

12 : Initial log posterior = 36121.520462

12 : Final log posterior = 36121.891785

13 : Initial log posterior = 36113.994770

13 : Final log posterior = 36114.276436

14 : Initial log posterior = 36109.663961

14 : Final log posterior = 36109.796252

15 : Initial log posterior = 36107.073517

15 : Final log posterior = 36107.148979

16 : Initial log posterior = 36105.570802

16 : Final log posterior = 36105.618037

17 : Initial log posterior = 36104.693355

17 : Final log posterior = 36104.707351

18 : Initial log posterior = 36104.151019

18 : Final log posterior = 36104.177058

19 : Initial log posterior = 36103.856528

19 : Final log posterior = 36103.858490
```

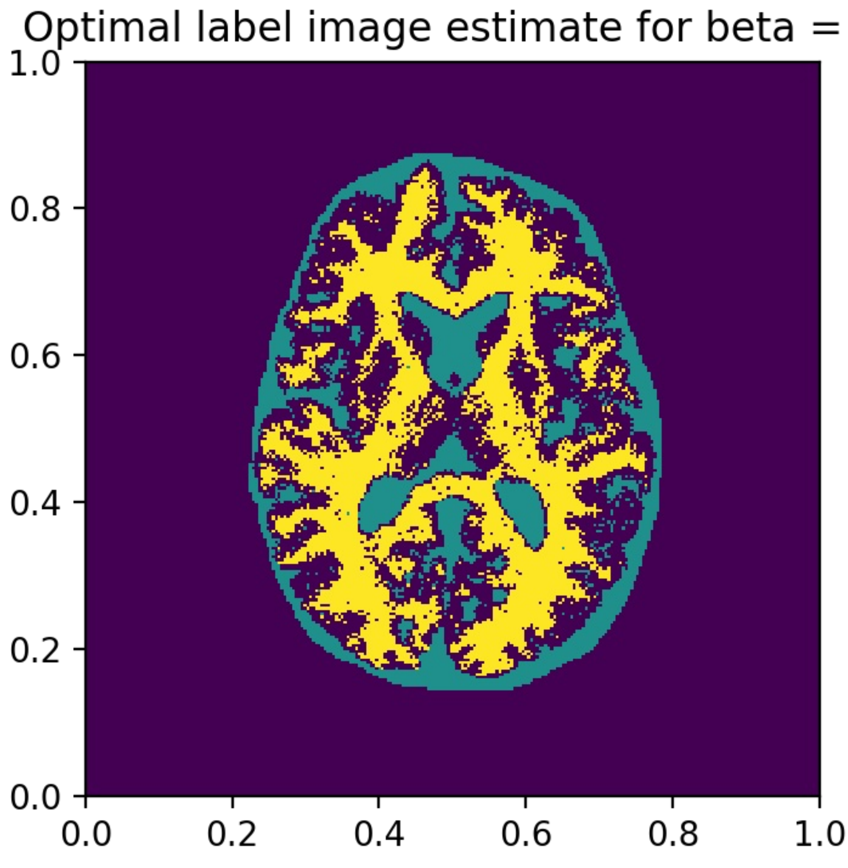
Images

```python
plt.figure()
plt.imshow(imageData, cmap=plt.cm.gray)
plt.title('Corrupted image')

plt.figure()
plt.imshow(initial_labels, extent=[0, 1, 0, 1])
plt.title('Initial estimate for the label image')


plt.figure()
plt.imshow(labels1, extent=[0, 1, 0, 1])
plt.title('Optimal label image estimate for beta = 2.0')

plt.figure()
plt.imshow(labels2, extent=[0, 1, 0, 1])
plt.title('Optimal label image estimate for beta = 0')

#Beta = 2.0
for i in range(K):
    seg = np.zeros((imageData.shape))
    seg[labels1==i] = imageData[labels1==i]
    plt.figure()
    plt.imshow(seg, cmap=plt.cm.gray)
    plt.title('Optimal class membership image estimate %d for beta = 2.0'%(i+1))

#Beta =0
for i in range(K):
    seg = np.zeros((imageData.shape))
    seg[labels2==i] = imageData[labels2==i]
    plt.figure()
    plt.imshow(seg, cmap=plt.cm.gray)
    plt.title('Optimal class membership image estimate %d for beta = 0'%(i+1))
```

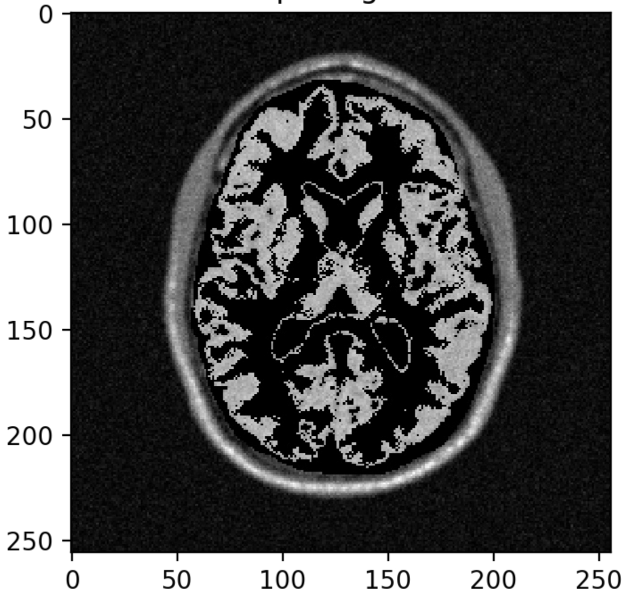

Corrupted image
0
50
100
150
200
250
0
50
100
150
200
250


Initial estimate for the label image
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

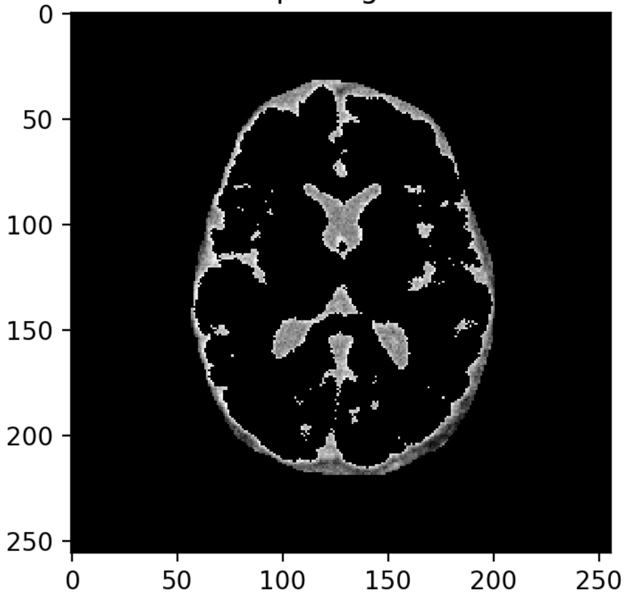

Optimal label image estimate for beta = 2.0
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


Optimal label image estimate for beta = 0
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

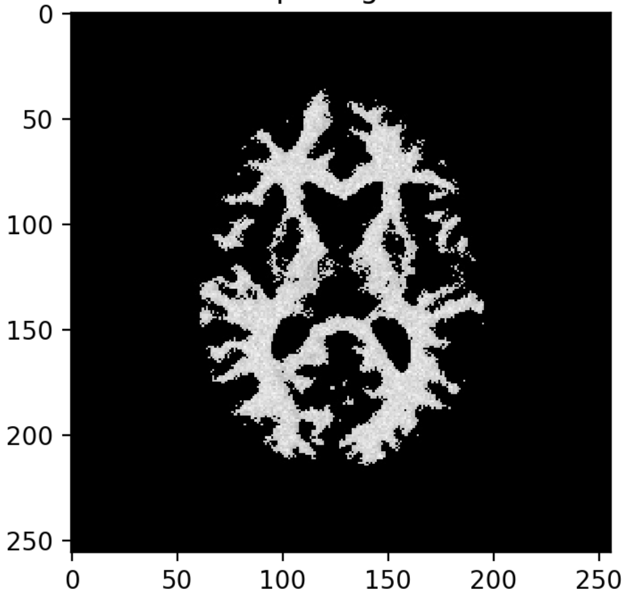
Optimal class membership image estimate 1 for beta = 2.0



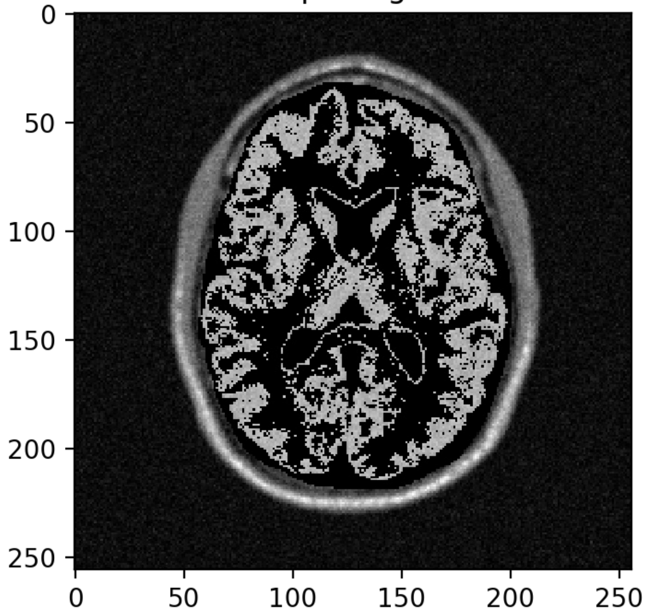
Optimal class membership image estimate 2 for beta = 2.0



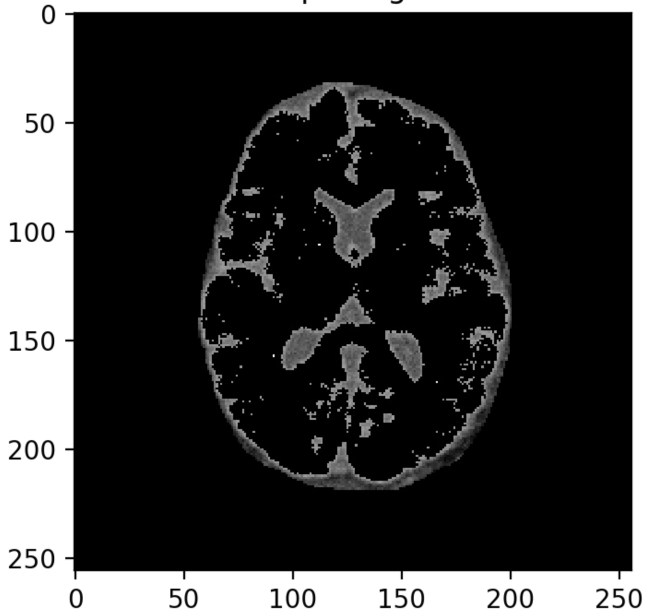
Optimal class membership image estimate 3 for beta = 2.0



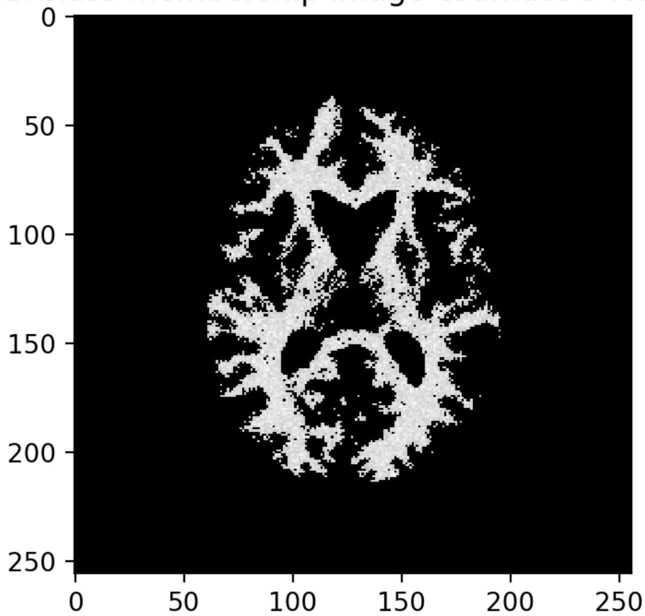
Optimal class membership image estimate 1 for beta = 0



Optimal class membership image estimate 2 for beta = 0



Optimal class membership image estimate 3 for beta = 0



Optimal Estimates

```
print('Initial class means are (%f, %f, %f)'%(means_init[0,0],means_init[1,0],means_init[2,0]))
print('For beta = 2.0, optimal class means are (%f, %f, %f)'%(means1[0,0],means1[1,0],means1[2,0]))
print('For beta = 0, optimal class means are (%f, %f, %f)'%(means2[0,0],means2[1,0],means2[2,0]))
```

```
Initial class means are (0.504863, 0.269874, 0.628400)
For beta = 2.0, optimal class means are (0.522365, 0.312841, 0.629673)
For beta = 0, optimal class means are (0.534559, 0.379461, 0.635745)
```

Published from q2.py using Pweave 0.30.3 on 23-03-2019.