Exercise 1

(a) Single gradient descent step and first-order decrease

Current iterate: x_k. Using fixed stepsize λ^{-1}, one (gradient descent) step gives

$$x_{k+1} = x_k - \lambda^{-1} f'(x_k).$$

Use a first-order Taylor expansion of f around x_k to approximate the change in cost:

$$f(x_{k+1}) \approx f(x_k) + f'(x_k)(x_{k+1} - x_k).$$

Substitute $x_{k+1} - x_k = -\lambda^{-1} f'(x_k)$ to obtain the approximate decrease

$$\Delta f \equiv f(x_{k+1}) - f(x_k) \approx f'(x_k)\left(-\lambda^{-1} f'(x_k)\right) = -\lambda^{-1} ¿$$

So the expected (first-order) decrease is,

$$\Delta f \approx -\lambda^{-1} ¿$$

(b) Using the result to adjust stepsize

From part (a), the decrease scales with $\lambda^{-1}(f')^2$. To make progress comparable across situations one can

- choose λ^{-1} inversely proportional to $(f')^2$ (or to a surrogate of it), or
- better: choose λ^{-1} proportional to an estimate of $1/f''$ so that step length adapts to local curvature.

Practically: monitor $f'(x_k)$ and set λ^{-1} smaller when $(f')^2$ (or curvature) is large to keep predictable decreases.

(c) Effect of rescaling the independent variable

Define the rescaled cost $\tilde{f}(x) := f(\alpha x)$ with $\alpha > 0$. Its derivative is

$$\tilde{f}'(x) = \frac{d}{dx} f(\alpha x) = \alpha f'(\alpha x).$$

If we perform one gradient step on $\tilde{f}$ with the same stepsize λ^{-1} starting at x_k, the expected decrease (by part (a)) is

$$\Delta\tilde{f} \approx -\lambda^{-1}¿$$

Thus, holding λ^{-1} fixed, the expected decrease is multiplied by α^2. In other words, rescaling the independent variable changes the magnitude of the predicted cost decrease (so optimization behaviour depends on the unit scale).

(d) Why $\lambda^{-1}=(\tilde{\lambda} f'')^{-1}$ gives scale invariance

For the scaled cost $\tilde{f}(x)=f(\alpha x)$,
$\tilde{f}''(x)=\alpha^2 f''(\alpha x)$.

Set the stepsize as
$\lambda^{-1}=(\tilde{\lambda} f'')^{-1}$
(and for the scaled problem use $\tilde{f}''$).

The predicted decrease becomes

[\begin{aligned} \Delta \tilde f &\approx -(\tilde\lambda, \tilde f''(x_k))^{-1}, [\tilde f'(x_k)]^{2} \ [4pt] &= -\dfrac{1}{\tilde\lambda, \alpha^{2} f''(\alpha x_k)} , \alpha^{2}, [f'(\alpha x_k)]^{2} \ [4pt] &= -\dfrac{1}{\tilde\lambda, f''(\alpha x_k)}, [f'(\alpha x_k)]^{2}. \end{aligned}]

All factors of α cancel, so the expected decrease matches the unscaled case.
Thus choosing $\lambda^{-1} \propto (f'')^{-1}$ yields scale-invariant progress.

(e) Carry-over to Gauss–Newton / Levenberg–Marquardt

For a scalar residual $h(x)$,
$f(x)=h(x)^2$,
$f'(x)=2h(x)h'(x)$,
$f''(x)=2h'(x)^2+2h(x)h''(x)$.

Gauss–Newton keeps only the dominant term $2h'(x)^2$.
Thus the LM stepsize analogue is

$$\lambda^{-1}=\left(\tilde{\lambda}[h'(x)]^2\right)^{-1}.$$

Under a rescaling $x\mapsto \alpha x$,
$h'(x)$ becomes $\alpha h'(\alpha x)$,
so $[h'(x)]^2$ gains a factor α^2.

In the predicted decrease, the term $[f'(x)]^2$ contains $[h'(x)]^2$,
and the stepsize denominator also contains $[h'(x)]^2$.
The α^2 factors cancel, leaving the expected progress unchanged.

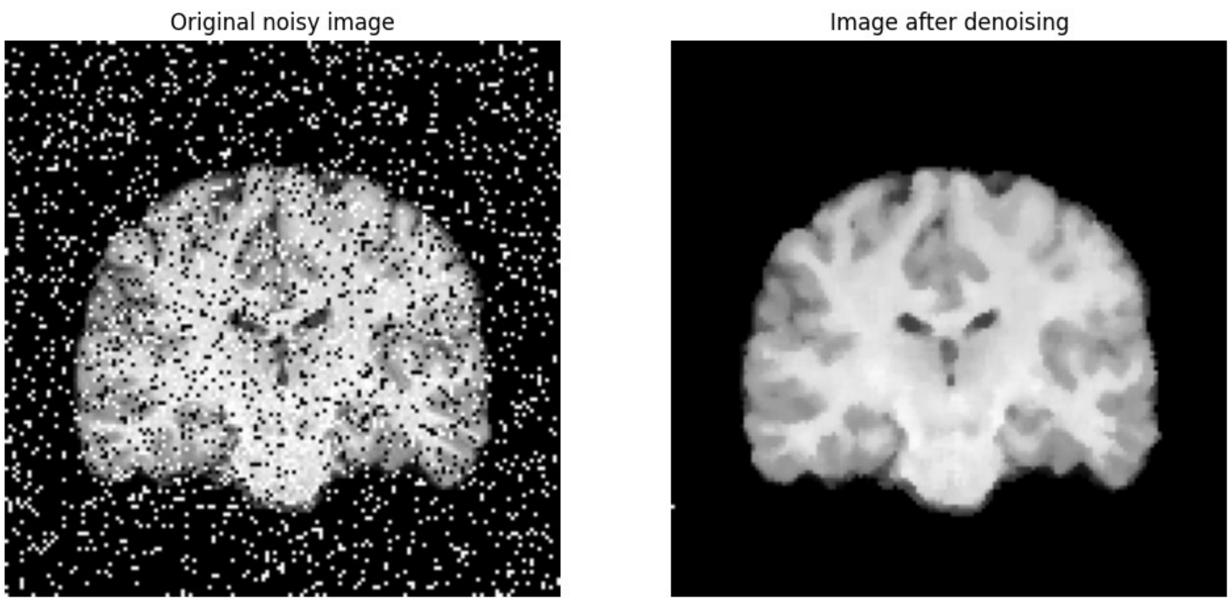
Thus LM retains the same scale invariance when using
$\lambda^{-1} \propto \left[h'(x)\right]^{-2}$.

Exercise 2

a)

We used Scipy's implementation of the median filter
(https://docs.scipy.org/doc/scipy-1.11.4/reference/generated/scipy.ndimage.median_filter.html).
The value for "size" parameter was chosen empirically by testing and comparing to the example image from the assignment.

```python
from skimage import io, img_as_float
from scipy import ndimage
import matplotlib.pyplot as plt

image = img_as_float(io.imread('brain-noisy.png'))
denoised = ndimage.median_filter(image, size=4)

plt.figure(figsize=(12,6))
plt.subplot(1,2,1)
plt.imshow(image,cmap="gray")
plt.title("Original noisy image")
plt.axis("off")

plt.subplot(1,2,2)
plt.imshow(denoised, cmap="gray")
plt.title("Image after denoising")
plt.axis("off")
plt.show()
```



b)

Looking at the log-scaled histogram we can see 4 distinct peaks. The peak near 0 is probably the background values that were included in the mask (our mask looks to cover a larger area than the denoised image brain). Looking at the other peaks we come up with the intensity value ranges as:

CSF : 0.05-0.5
Gray matter(GM): 0.5-0.75
white matter(WM): 0.75-1

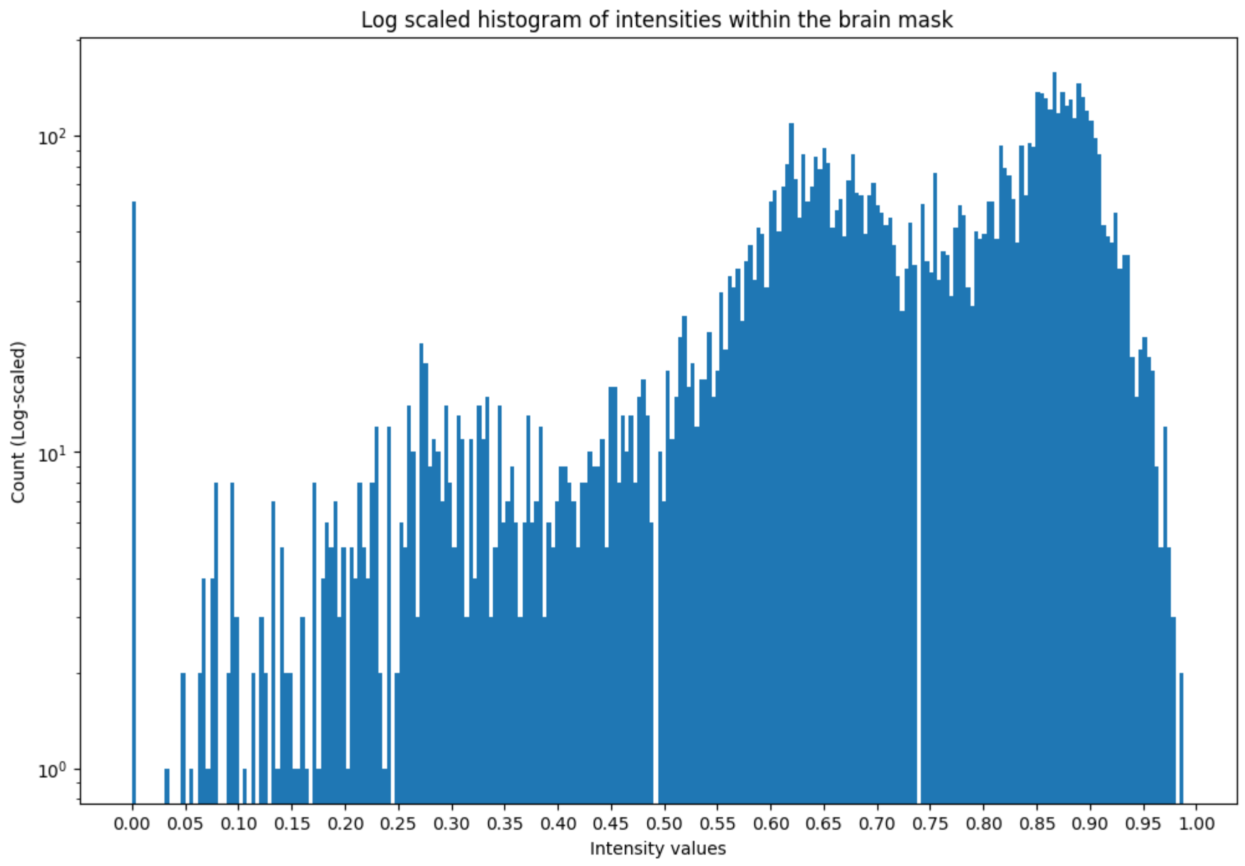
There is defintely overlap between these ranges in the actual values but this is the estimate we could come up with just from looking at the histogram.
To estimate the mixing weights, we used a very rough calculation of area under the curve. For this matter we decided a non-log scaled histogram would be more useful to find the heights. For this we just use area of a rectangle with width=the intensitiy range and height=visually average count. Therefore using the assumptions :

CSF -> width=0.45 height=10 area=4.5
GM -> width=0.25 height=70 area=17.5
WM -> width=0.25 height=120 area=30

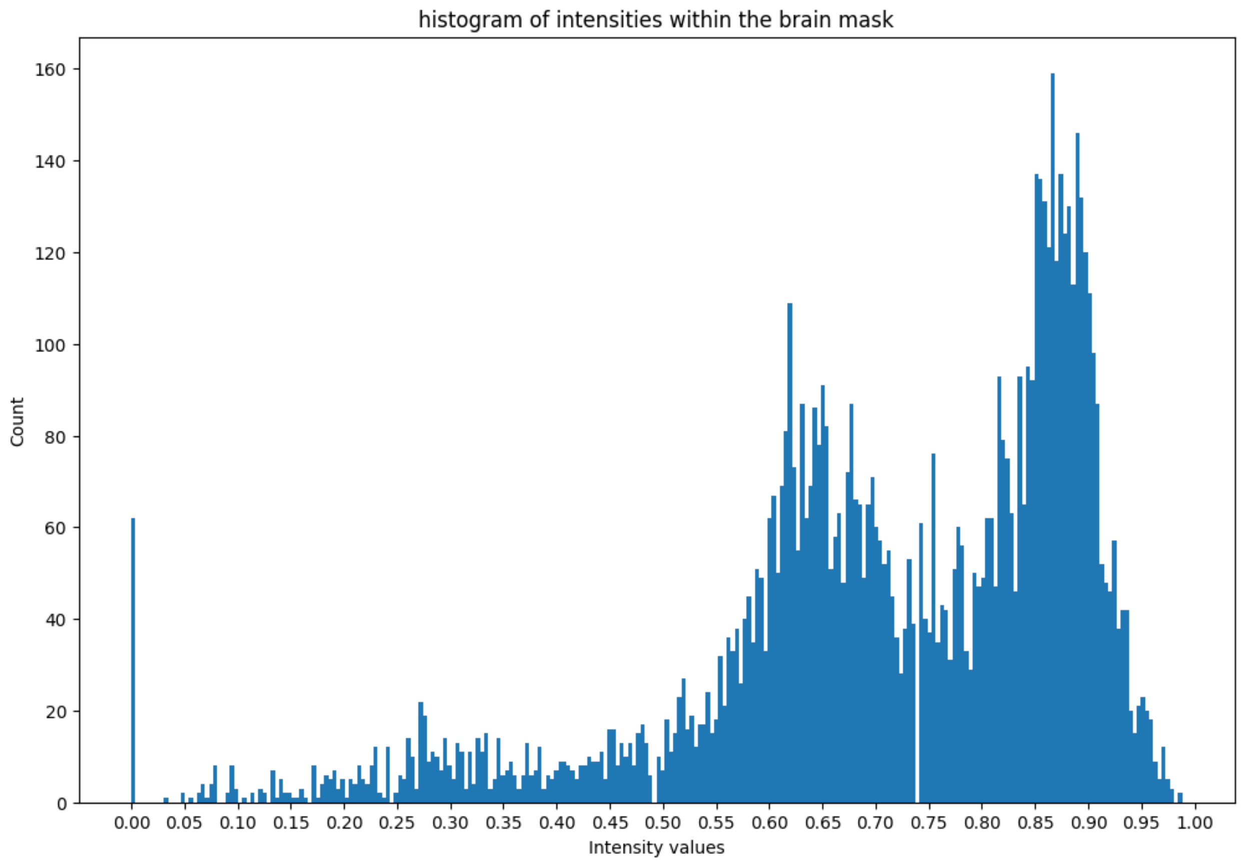
This gives:
CSF = 4.5/52 = 0.086
GM = 17.5/52 = 0.34
WM = 30/52 = 0.57

We approximate mixing weights (to sum=1):
Pi_csf=0.09
Pi_GM=0.34
Pi_WM=0.57

```python
import numpy as np
from matplotlib.ticker import MultipleLocator

mask_image=io.imread("mask.png")
mask= mask_image > 0
brain_vals = denoised[mask]
plt.figure(figsize=(12,8))
plt.hist(brain_vals,bins=256)
plt.yscale('log')
ax = plt.gca()
ax.xaxis.set_major_locator(MultipleLocator(0.05))
plt.title("Log scaled histogram of intensities within the brain mask")
plt.xlabel("Intensity values")
plt.ylabel("Count (Log-scaled)")
plt.show()
```



```
plt.figure(figsize=(12,8))
plt.hist(brain_vals,bins=256)
ax = plt.gca()
ax.xaxis.set_major_locator(MultipleLocator(0.05))
plt.title("histogram of intensities within the brain mask")
plt.xlabel("Intensity values")
plt.ylabel("Count")
plt.show()
```



C)

The mixing weights were already estimated in b.
to estimate mean we just visually choose the intensity values where the peaks are visible:

mean_CSF : 0.27
mean_Gray matter(GM): 0.63
mean_white matter(WM): 0.87

to estimate variance we use the 68-95-99 rule of normal distribution. We assume that the tails of the distributions are overlapping so we use the estimate that the range values of each group should fall within 2 sigma of the mean from both side so 4 sigma overall. Using the ranges from b and the estimated means we get:

var_CSF : (0.45/4)^2 = (0.1125)^2 = 0.0126
var_Gray matter(GM): (0.25/4)^2 = (0.0625)^2 = 0.0039
var_white matter(WM): (0.25/4)^2 = (0.0625)^2 = 0.0039

```
N = brain_vals.shape[0]
K = 3
responsibility = np.zeros((N, K), dtype=float)
means = np.array([0.27, 0.63, 0.87], dtype=float)
vars  = np.array([0.0126, 0.0039, 0.0039], dtype=float)
mix   = np.array([0.09, 0.34, 0.57],  dtype=float)
```

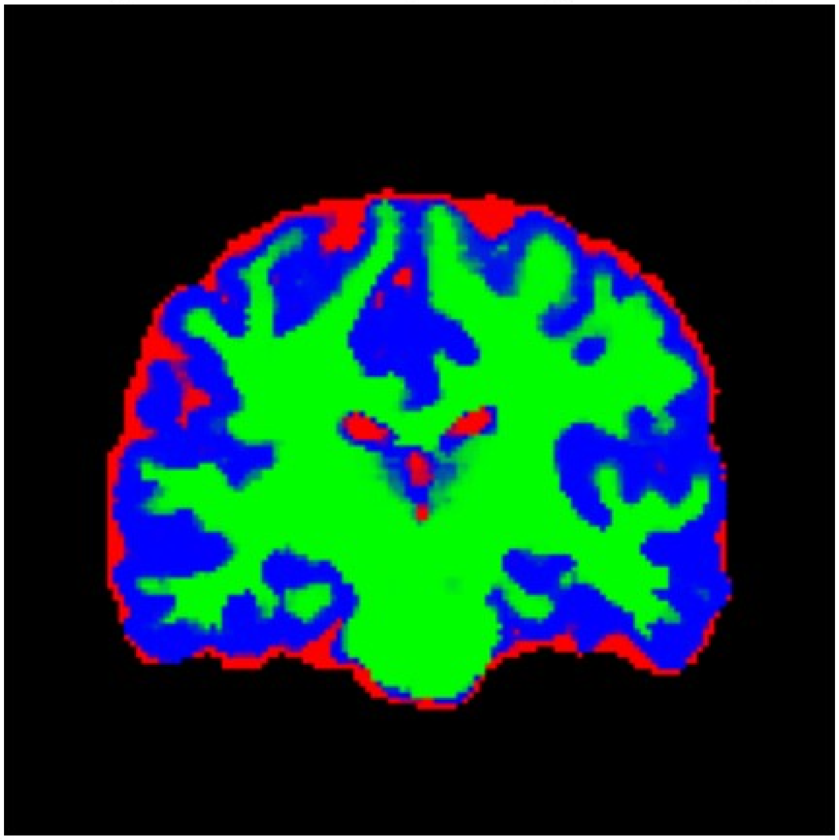
```
def normal_pdf(x, mu, var): # this is N(xi | mu_k, var_k)
    return (1.0 / np.sqrt(2.0 * np.pi * var)) * np.exp(-0.5 * (x - mu)
** 2 / var)

def E_step(img_data, responsibility):
    for i in range(N):
        x_i = img_data[i]
        denominator = 0
        for k in range(K):
            denominator += mix[k] * normal_pdf(x_i, means[k], vars[k])
        for k in range(K):
            numerator = mix[k] * normal_pdf(x_i, means[k], vars[k])
            responsibility[i, k] = numerator / denominator

E_step(brain_vals, responsibility)
print(responsibility)
```

```
[[1.00000000e+00 9.75738846e-21 1.47696033e-40]
 [1.00000000e+00 9.75738846e-21 1.47696033e-40]
 [1.00000000e+00 9.75738846e-21 1.47696033e-40]
 ...
 [9.99947271e-01 5.27290141e-05 3.99094198e-16]
 [9.99947271e-01 5.27290141e-05 3.99094198e-16]
 [9.99992790e-01 7.20961796e-06 1.28261208e-17]]
```

d)

```
H, W = denoised.shape
result= np.zeros((H, W, 3), dtype=float)  #the 3rd dimension is for
rgb so heigh x weight x rgb
result[..., 0][mask] = responsibility[:, 0]
result[..., 2][mask] = responsibility[:, 1]
result[..., 1][mask] = responsibility[:, 2]
plt.figure()
plt.imshow(result)
plt.title("Segmentation result (After only one E-step)")
plt.axis("off")
plt.tight_layout()
plt.show()
```

Segmentation result (After only one E-step)



e)

```
def M_step(img_data,responsibility):
    for k in range(K):
        num_mu=0
        num_var=0
        N_k=0
        for i in range(N):
            num_mu += responsibility[i,k]*img_data[i]

            N_k += responsibility[i,k]
        mean=num_mu/N_k
        means[k]=mean
        mix[k]=N_k/N
        for i in range(N):
            num_var += responsibility[i,k]*pow((img_data[i] - mean),2)
        vars[k]=num_var/N_k


M_step(brain_vals,responsibility)
print("re-computed means:", np.round(means, 3))
print("re-computed variances:", np.round(vars, 3))
print("re-computed mixing weights:", np.round(mix, 3))
```

```
re-computed means: [0.288 0.637 0.854]
re-computed variances: [0.019 0.004 0.003]
re-computed mixing weights: [0.095 0.39  0.515]
```

f)

We really couldn't figure out why our final segmentation is not as smooth as the example picture in the assignment. There are more red (CSF) patches in the blue (GM) area. We even lowered the tolerance and ran it for longer until convergence but no immediate improvements. It might be due to our initiliziation values.

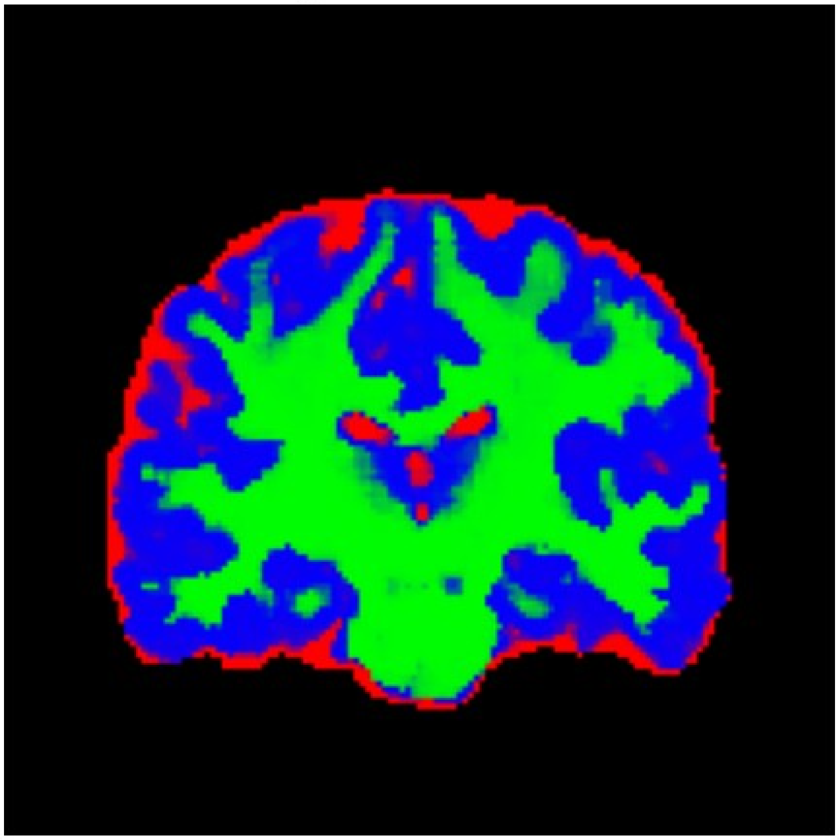
```
N = brain_vals.shape[0]
K = 3
responsibility = np.zeros((N, K), dtype=float)
means = np.array([0.27, 0.63, 0.87], dtype=float)
vars  = np.array([0.0126, 0.0039, 0.0039], dtype=float)
mix   = np.array([0.09, 0.34, 0.57],  dtype=float)
max_iter = 500
tol = 1e-10

for iter in range(max_iter):
    old_means = means.copy()
    old_vars  = vars.copy()
    old_mix   = mix.copy()
    E_step(brain_vals, responsibility)
    M_step(brain_vals, responsibility)

    diff_means = np.max(np.abs(means - old_means))
    diff_vars  = np.max(np.abs(vars  - old_vars))
    diff_mix   = np.max(np.abs(mix   - old_mix))
    diff = max(diff_means, diff_vars, diff_mix)

    print(f"Iter {iter:02d}")

    if diff < tol:
        print("Converged.")
        break
```

```
Iter 00
Iter 01
Iter 02
Iter 03
Iter 04
Iter 05
Iter 06
Iter 07
Iter 08
Iter 09
Iter 10
Iter 11
Iter 12
Iter 13
Iter 14
Iter 15
Iter 16
Iter 17
Iter 18
Iter 19
Iter 20
Iter 21
Iter 22
Iter 23
Iter 24
Iter 25
Iter 26
Iter 27
Iter 28
Iter 29
Iter 30
Iter 31
Iter 32
Iter 33
Iter 34
Iter 35
Iter 36
Iter 37
Iter 38
Iter 39
Iter 40
Iter 41
Iter 42
Iter 43
Iter 44
Iter 45
Iter 46
Iter 47
Iter 48
Iter 49
Iter 50
Iter 51
Iter 52
Iter 53
Iter 54
Iter 55
Iter 56
Iter 57
Iter 58
Iter 59
Iter 60
Iter 61
Iter 62
Iter 63
Iter 64
Iter 65
Iter 66
Iter 67
Iter 68
Iter 69
Iter 70
Iter 71
Iter 72
Iter 73
Iter 74
Iter 75
Iter 76
Iter 77
Iter 78
Iter 79
Iter 80
Iter 81
Iter 82
Iter 83
Iter 84
Iter 85
Iter 86
Iter 87
Iter 88
Iter 89
Iter 90
Iter 91
Iter 92
Iter 93
Iter 94
Iter 95
Iter 96
Iter 97
Iter 98
Iter 99
Iter 100
Iter 101
Iter 102
Iter 103
Iter 104
Iter 105
Iter 106
Iter 107
Iter 108
Iter 109
Iter 110
Iter 111
Iter 112
Iter 113
Iter 114
Iter 115
Iter 116
Iter 117
Iter 118
Iter 119
Iter 120
Iter 121
Iter 122
Iter 123
Iter 124
Iter 125
Iter 126
Iter 127
Iter 128
Iter 129
Iter 130
Iter 131
Iter 132
Iter 133
Iter 134
Iter 135
Iter 136
Iter 137
Iter 138
Iter 139
Iter 140
Iter 141
Iter 142
Iter 143
Iter 144
Iter 145
Iter 146
Iter 147
Iter 148
Iter 149
Iter 150
Iter 151
Iter 152
Iter 153
Iter 154
Iter 155
Iter 156
Iter 157
Iter 158
Iter 159
Iter 160
Iter 161
Iter 162
Iter 163
Iter 164
Iter 165
Iter 166
Iter 167
Iter 168
Iter 169
Iter 170
Iter 171
Iter 172
Iter 173
Iter 174
Iter 175
Iter 176
Iter 177
Iter 178
Iter 179
Iter 180
Iter 181
Iter 182
Iter 183
Iter 184
Iter 185
Iter 186
Iter 187
Iter 188
Iter 189
Iter 190
Iter 191
Iter 192
Iter 193
Iter 194
Iter 195
Iter 196
Iter 197
Iter 198
Iter 199
Iter 200
Iter 201
Iter 202
Iter 203
Iter 204
Iter 205
Iter 206
Iter 207
Iter 208
Iter 209
Iter 210
Iter 211
Iter 212
Iter 213
Iter 214
Iter 215
Iter 216
Iter 217
Iter 218
Iter 219
Iter 220
Iter 221
Iter 222
Iter 223
Iter 224
Iter 225
Iter 226
Iter 227
Iter 228
Iter 229
Iter 230
Iter 231
Iter 232
Iter 233
Iter 234
Iter 235
Iter 236
Iter 237
Iter 238
Iter 239
Iter 240
Iter 241
Iter 242
Iter 243
Iter 244
Iter 245
Iter 246
Iter 247
Iter 248
Iter 249
Iter 250
Iter 251
Iter 252
Iter 253
Iter 254
Iter 255
Iter 256
Iter 257
Iter 258
Iter 259
Iter 260
Iter 261
Iter 262
Iter 263
Iter 264
Iter 265
Iter 266
Iter 267
Iter 268
Iter 269
Iter 270
Iter 271
Iter 272
Iter 273
Iter 274
Iter 275
Iter 276
Iter 277
Iter 278
Iter 279
Iter 280
Iter 281
Iter 282
Iter 283
Iter 284
Iter 285
Iter 286
Iter 287
Iter 288
Iter 289
Iter 290
Iter 291
Iter 292
Iter 293
Iter 294
Iter 295
Iter 296
Iter 297
Iter 298
Iter 299
Iter 300
Iter 301
Iter 302
Iter 303
Iter 304
Iter 305
Iter 306
Iter 307
Iter 308
Iter 309
Iter 310
Iter 311
Iter 312
Iter 313
Iter 314
Iter 315
Iter 316
Iter 317
Iter 318
Iter 319
Iter 320
Iter 321
Iter 322
Iter 323
Iter 324
Iter 325
Iter 326
Iter 327
Iter 328
Iter 329
Iter 330
Iter 331
Iter 332
Iter 333
Iter 334
Iter 335
Iter 336
Iter 337
Iter 338
Iter 339
Iter 340
Iter 341
Iter 342
Iter 343
Iter 344
Iter 345
Iter 346
Iter 347
Iter 348
Iter 349
Iter 350
Iter 351
Iter 352
Iter 353
Iter 354
Iter 355
Iter 356
Iter 357
Iter 358
Iter 359
Iter 360
Iter 361
Iter 362
Iter 363
Iter 364
Iter 365
Iter 366
Converged.
```

```python
print("final means:", np.round(means, 3))
print("final variances:", np.round(vars, 3))
print("final mixing weights:", np.round(mix, 3))
H, W = denoised.shape
result= np.zeros((H, W, 3), dtype=float)
result[..., 0][mask] = responsibility[:, 0]
result[..., 2][mask] = responsibility[:, 1]
result[..., 1][mask] = responsibility[:, 2]
plt.figure()
plt.imshow(result)
plt.title("Segmentation result")
plt.axis("off")
plt.tight_layout()
plt.show()
```

```
final means: [0.351 0.668 0.871]
final variances: [0.031 0.006 0.002]
final mixing weights: [0.12  0.455 0.425]
```

Segmentation result



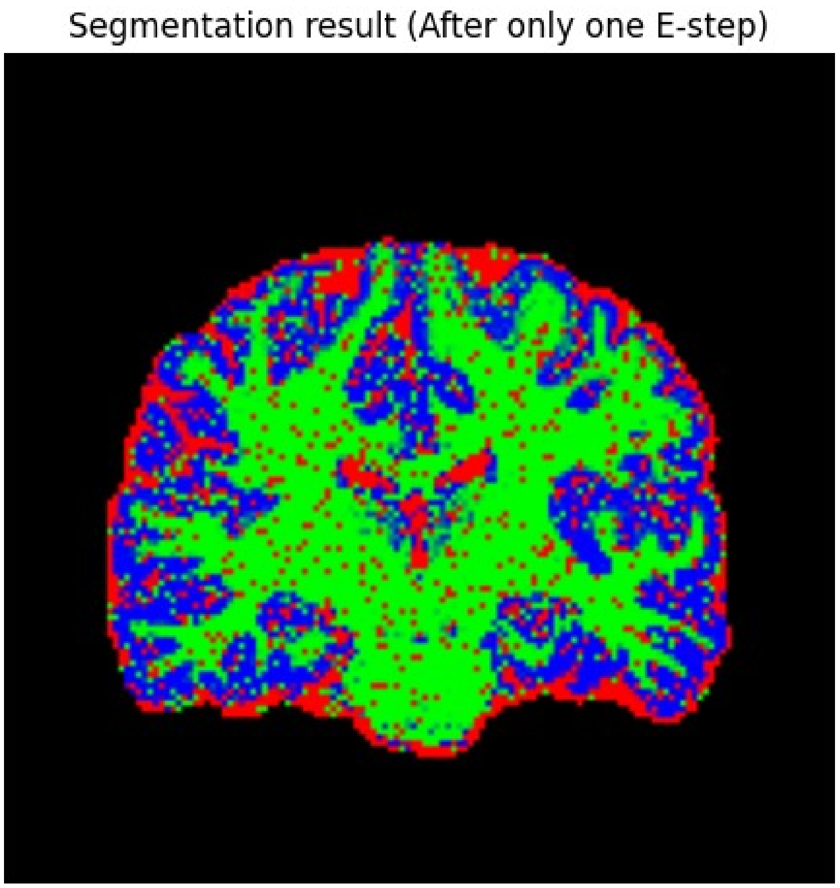
Exercise 3

a)

Leaving out the median filtering and running one 1 E-step we get the hard labeled image of the noisy original image.

```
from skimage import io, img_as_float
from scipy import ndimage
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.ticker import MultipleLocator

image = img_as_float(io.imread('brain-noisy.png'))

mask_image=io.imread("mask.png")
mask= mask_image > 0
brain_vals = image[mask]

N = brain_vals.shape[0]
```

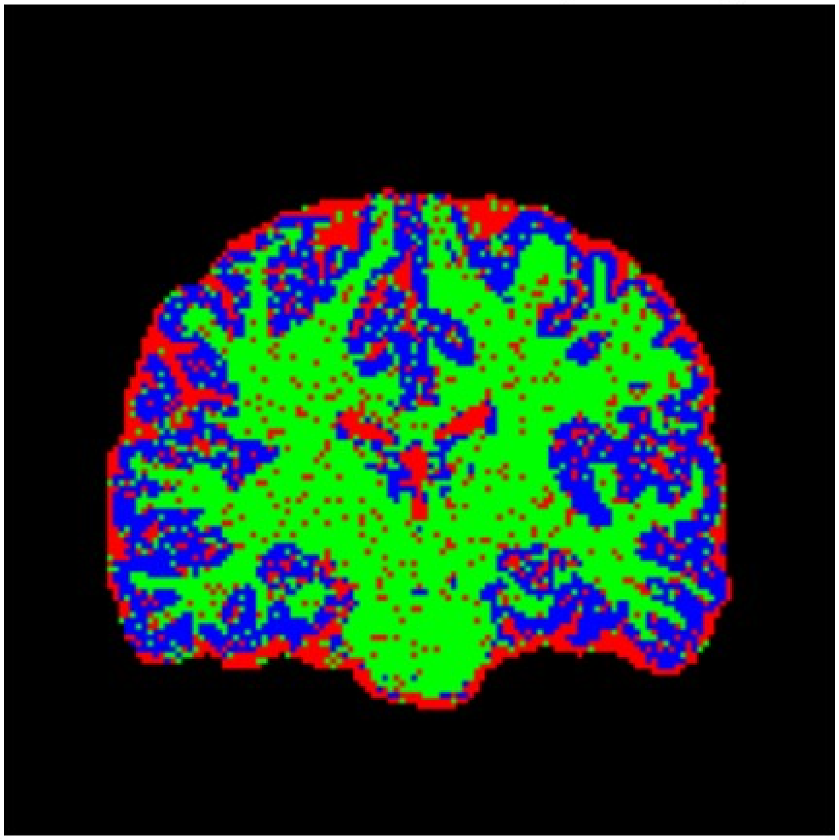
```
K = 3
responsibility = np.zeros((N, K), dtype=float)
means = np.array([0.27, 0.63, 0.87], dtype=float)
vars  = np.array([0.0126, 0.0039, 0.0039], dtype=float)
mix   = np.array([0.09, 0.34, 0.57],  dtype=float)

E_step(brain_vals, responsibility)

H, W = image.shape
result= np.zeros((H, W, 3), dtype=float)  #the 3rd dimension is for
rgb so heigh x weight x rgb
result[..., 0][mask] = responsibility[:, 0]
result[..., 2][mask] = responsibility[:, 1]
result[..., 1][mask] = responsibility[:, 2]
plt.figure()
plt.imshow(result)
plt.title("Segmentation result (After only one E-step)")
plt.axis("off")
plt.tight_layout()
plt.show()
```



b)

```python
import numpy as np
import matplotlib.pyplot as plt

def get_neighbors_indices(row, col, H, W):
    neighbors = []
    deltas = [(-1, 0), (0, 1), (1, 0), (0, -1)]
    for dr, dc in deltas:
        nr, nc = row + dr, col + dc
        if 0 <= nr < H and 0 <= nc < W:
            neighbors.append((nr, nc))
    return neighbors

def compute_unary_potential(pixel_index, label_k, means, vars):
    x_i = brain_vals[pixel_index]
    pdf_val = normal_pdf(x_i, means[label_k], vars[label_k])
    return -np.log(pdf_val + 1e-15)

def compute_pairwise_potential(label_i, label_j, beta):
    return beta if label_i != label_j else 0.0

def ICM_iteration(labels_2D, means, vars, beta, mask, H, W, N, K,
brain_vals):

    map_2D_to_1D = -np.ones((H, W), dtype=int)
    map_2D_to_1D[mask] = np.arange(N)

    brain_pixels_coords = np.argwhere(mask)
    new_labels_2D = labels_2D.copy()
    changes = 0

    for r, c in brain_pixels_coords:
        original_label = labels_2D[r, c]
        min_energy = float('inf')
        best_label = original_label
        pixel_1D_index = map_2D_to_1D[r, c]

        for k in range(K):
            # Calculate Unary Potential
            x_i = brain_vals[pixel_1D_index]
            pdf_val = (1.0 / np.sqrt(2.0 * np.pi * vars[k])) *
np.exp(-0.5 * (x_i - means[k]) ** 2 / vars[k])
            unary_potential = -np.log(pdf_val + 1e-15)

            # Calculate Pairwise Potential
            pairwise_sum = 0.0
            neighbors_coords = get_neighbors_indices(r, c, H, W)

            for nr, nc in neighbors_coords:
                if mask[nr, nc]:
                    neighbor_label = labels_2D[nr, nc]
                    pairwise_sum += beta if neighbor_label != k else
0.0

            # Total Energy = Unary + Pairwise
            total_energy = unary_potential + pairwise_sum

            if total_energy < min_energy:
                min_energy = total_energy
                best_label = k

        # Update and count changes
        new_labels_2D[r, c] = best_label
        if best_label != original_label:
            changes += 1

    return new_labels_2D, changes

gmm_labels_1D = np.argmax(responsibility, axis=1)
labels_2D = np.zeros((H, W), dtype=int)
labels_2D[mask] = gmm_labels_1D

beta = 0.5

labels_after_1_icm, changes = ICM_iteration(labels_2D.copy(), means,
vars, beta, mask, H, W, N, K, brain_vals)

rgb_after_1_icm = np.zeros((H, W, 3), dtype=float)

for r, c in np.argwhere(mask):
    label_k = labels_after_1_icm[r, c]

    if label_k == 0:
        rgb_after_1_icm[r, c, 0] = 1
    elif label_k == 1:
        rgb_after_1_icm[r, c, 2] = 1
    elif label_k == 2:
        rgb_after_1_icm[r, c, 1] = 1

plt.figure()
plt.imshow(rgb_after_1_icm)
plt.title(f"Results after 1 ICM iteration (beta={beta})")
plt.axis("off")
plt.tight_layout()
plt.show()

print(f"Pixel labels changed after 1 ICM iteration: {changes}")
```

Results after 1 ICM iteration (beta=0.5)

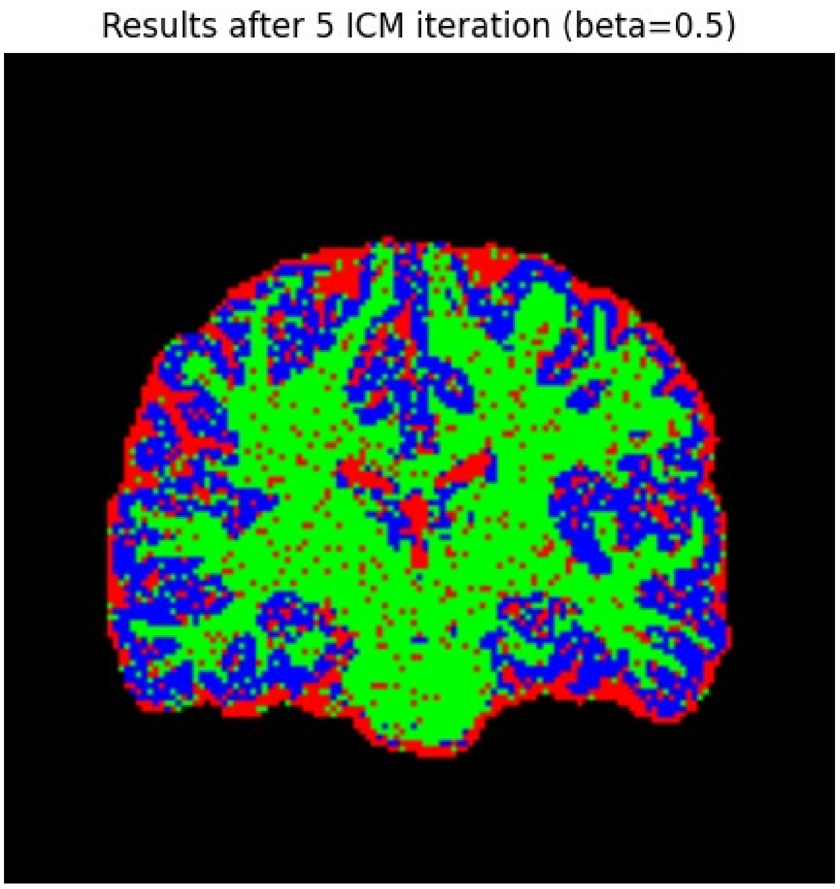


```
Pixel labels changed after 1 ICM iteration: 149
```

c)

```
labels_after_5_icm = labels_2D.copy()
beta = 0.5
total_changes = 0

for i in range(5):
    # Perform ICM on the result of the previous iteration
    labels_after_5_icm, current_changes =
ICM_iteration(labels_after_5_icm, means, vars, beta, mask, H, W, N, K,
brain_vals)
    total_changes += current_changes
    print(f"ICM Iteration {i+1}: {current_changes} pixels changed.")

rgb_after_5_icm = np.zeros((H, W, 3), dtype=float)
for r, c in np.argwhere(mask):
    label_k = labels_after_5_icm[r, c]

    if label_k == 0:
        rgb_after_5_icm[r, c, 0] = 1
    elif label_k == 1:
        rgb_after_5_icm[r, c, 2] = 1
    elif label_k == 2:
        rgb_after_5_icm[r, c, 1] = 1

plt.figure()
plt.imshow(rgb_after_5_icm)
plt.title(f"Results after 5 ICM iteration (beta={beta})")
plt.axis("off")
plt.tight_layout()
plt.show()

print(f"\nTotal pixel labels changed across the 5 ICM iterations:
{total_changes}")
```

```
ICM Iteration 1: 149 pixels changed.
ICM Iteration 2: 16 pixels changed.
ICM Iteration 3: 5 pixels changed.
ICM Iteration 4: 4 pixels changed.
ICM Iteration 5: 3 pixels changed.
```



```
Total pixel labels changed across the 5 ICM iterations: 177
```

d)

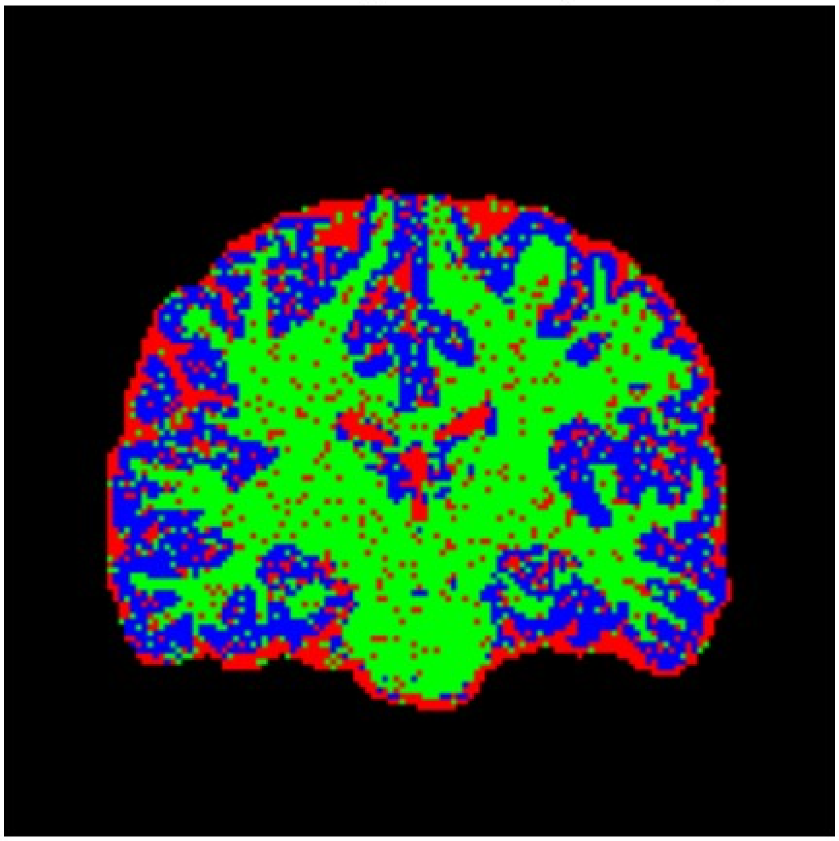
```
def EM_MRF_iteration(img_data, labels_2D, means, vars, mix, beta,
mask, H, W, N, K):
    icm_max_iter = 500
    icm_labels_2D = labels_2D.copy()

    for i in range(icm_max_iter):
        icm_labels_2D, changes = ICM_iteration(icm_labels_2D, means,
vars, beta, mask, H, W, N, K, img_data)
        if changes == 0:
            break

    labels_1D = icm_labels_2D[mask]
    hard_responsibility = np.zeros((N, K), dtype=float)
    for k in range(K):
        hard_responsibility[labels_1D == k, k] = 1.0

    new_means = means.copy()
    new_vars = vars.copy()
    new_mix = mix.copy()

    for k in range(K):
        N_k = np.sum(hard_responsibility[:, k])

        if N_k > 0:
            num_mu = np.sum(hard_responsibility[:, k] * img_data)
            mean = num_mu / N_k
            new_means[k] = mean

            new_mix[k] = N_k / N

            diff_sq = (img_data - mean)**2
            num_var = np.sum(hard_responsibility[:, k] * diff_sq)
            new_vars[k] = num_var / N_k

    return icm_labels_2D, new_means, new_vars, new_mix

gmm_labels_2D = np.zeros((H, W), dtype=int)
gmm_labels_2D[mask] = gmm_labels_1D
em_mrf_labels_2D = gmm_labels_2D.copy()
em_mrf_means = means.copy()
em_mrf_vars = vars.copy()
em_mrf_mix = mix.copy()
beta = 0.5

em_mrf_max_iter = 50
```

```
em_mrf_tol = 1e-6


for iter in range(em_mrf_max_iter):
    old_means = em_mrf_means.copy()

    print(f"EM-MRF Outer Iter {iter:02d}")

    # Perform one iteration
    em_mrf_labels_2D, em_mrf_means, em_mrf_vars, em_mrf_mix =
EM_MRF_iteration(
        brain_vals, em_mrf_labels_2D, em_mrf_means, em_mrf_vars,
em_mrf_mix, beta, mask, H, W, N, K
    )

    diff_means = np.max(np.abs(em_mrf_means - old_means))

    if diff_means < em_mrf_tol:
        print("EM-MRF Converged.")
        break
    elif iter == em_mrf_max_iter - 1:
        print("EM-MRF max iterations reached.")

final_em_mrf_rgb = np.zeros((H, W, 3), dtype=float)

for r, c in np.argwhere(mask):
    label_k = em_mrf_labels_2D[r, c]

    if label_k == 0:
        final_em_mrf_rgb[r, c, 0] = 1
    elif label_k == 1:
        final_em_mrf_rgb[r, c, 2] = 1
    elif label_k == 2:
        final_em_mrf_rgb[r, c, 1] = 1

plt.figure()
plt.imshow(final_em_mrf_rgb)
plt.title(f"Final EM-MRF Segmentation (beta={beta})")
plt.axis("off")
plt.tight_layout()
plt.show()
```

```
EM-MRF Outer Iter 00
EM-MRF Outer Iter 01
EM-MRF Outer Iter 02
EM-MRF Outer Iter 03
EM-MRF Outer Iter 04
EM-MRF Outer Iter 05
EM-MRF Outer Iter 06
```

```
EM-MRF Outer Iter 07
EM-MRF Converged.
```

Final EM-MRF Segmentation (beta=0.5)



e)

After some tests we noticed that with β in between 3 to 8, the results start to be less noisy, but when it equals 9, the result resembles more to the example image given in the exercise sheet.

```
gmm_labels_2D = np.zeros((H, W), dtype=int)
gmm_labels_2D[mask] = gmm_labels_1D
em_mrf_labels_2D = gmm_labels_2D.copy()
em_mrf_means = means.copy()
em_mrf_vars = vars.copy()
em_mrf_mix = mix.copy()
beta = 9

em_mrf_max_iter = 50
em_mrf_tol = 1e-6


for iter in range(em_mrf_max_iter):
    old_means = em_mrf_means.copy()

    print(f"EM-MRF Outer Iter {iter:02d}")

    # Perform one iteration
    em_mrf_labels_2D, em_mrf_means, em_mrf_vars, em_mrf_mix =
EM_MRF_iteration(
        brain_vals, em_mrf_labels_2D, em_mrf_means, em_mrf_vars,
em_mrf_mix, beta, mask, H, W, N, K
    )

    diff_means = np.max(np.abs(em_mrf_means - old_means))

    if diff_means < em_mrf_tol:
        print("EM-MRF Converged.")
        break
    elif iter == em_mrf_max_iter - 1:
        print("EM-MRF max iterations reached.")

final_em_mrf_rgb = np.zeros((H, W, 3), dtype=float)

for r, c in np.argwhere(mask):
    label_k = em_mrf_labels_2D[r, c]

    if label_k == 0:
        final_em_mrf_rgb[r, c, 0] = 1
    elif label_k == 1:
        final_em_mrf_rgb[r, c, 2] = 1
    elif label_k == 2:
        final_em_mrf_rgb[r, c, 1] = 1

plt.figure()
plt.imshow(final_em_mrf_rgb)
plt.title(f"Final EM-MRF Segmentation (beta={beta})")
plt.axis("off")
plt.tight_layout()
plt.show()
```

```
EM-MRF Outer Iter 00
EM-MRF Outer Iter 01
EM-MRF Outer Iter 02
EM-MRF Outer Iter 03
EM-MRF Converged.
```

Final EM-MRF Segmentation (beta=9)