CSC528
Assignment 3
Hithesh Shanmugam

Problem 1:

Code:

```
import numpy as np
import cv2
import matplotlib.pyplot as plt

def read_img(filename):
    img = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
    return img

def initialize_parameters(img, k):
    img_flat = img.flatten()

    # Use k-means clustering to initialize the means
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 1.0)
    _, labels, centers = cv2.kmeans(np.float32(img_flat), k, None, criteria, 10, cv2.KMEANS_RANDOM_CENTERS)
    means = centers.squeeze()

    # Initialize covariances and mixing coefficients
    covariances = [np.var(img_flat)] * k
    mixing_coeffs = np.ones(k) / k

    return means, covariances, mixing_coeffs

def expectation_step(img_flat, means, covariances, mixing_coeffs, k):
    responsibilities = np.zeros((img_flat.shape[0], k))
    for i in range(k):
        responsibilities[:, i] = mixing_coeffs[i] * gaussian_prob(img_flat, means[i], covariances[i])
    responsibilities /= np.sum(responsibilities, axis=1, keepdims=True)
    return responsibilities

def maximization_step(img_flat, responsibilities, k):
    N = img_flat.shape[0]
    means = np.zeros(k)
    covariances = np.zeros(k)
    mixing_coeffs = np.zeros(k)
    for i in range(k):
        resp = responsibilities[:, i]
        means[i] = np.sum(resp * img_flat) / np.sum(resp)
        covariances[i] = np.sum(resp * (img_flat - means[i]) ** 2) / np.sum(resp)
        mixing_coeffs[i] = np.mean(resp)
    return means, covariances, mixing_coeffs

def gaussian_prob(x, mean, cov):
    return np.exp(-(x - mean) ** 2 / (2 * cov)) / np.sqrt(2 * np.pi * cov)

def compute_log_likelihood(img_flat, means, covariances, mixing_coeffs, k):
    log_likelihood = 0.0
    for i in range(k):
        log_likelihood += np.log(mixing_coeffs[i] * gaussian_prob(img_flat, means[i], covariances[i]))
    return np.sum(log_likelihood)

def segment_image(img, k, n_iterations=100, epsilon=1e-7):
    img_flat = img.flatten()
    means, covariances, mixing_coeffs = initialize_parameters(img, k)
    log_likelihoods = []

    for iteration in range(n_iterations):
        # Expectation step
        responsibilities = expectation_step(img_flat, means, covariances, mixing_coeffs, k)

        # Maximization step
        means, covariances, mixing_coeffs = maximization_step(img_flat, responsibilities, k)

        # Compute log-likelihood
        log_likelihood = compute_log_likelihood(img_flat, means, covariances, mixing_coeffs, k)
        log_likelihoods.append(log_likelihood)

        # Check for convergence
        if iteration > 0 and np.abs(log_likelihood - log_likelihoods[iteration-1]) < epsilon:
            break

    # Compute segmented image
    segmented_img = np.argmax(responsibilities, axis=1).reshape(img.shape)

    return segmented_img, log_likelihoods, means
```

```python
# Read the image
img = read_img('C:/Users/sures/OneDrive - DePaul University/Desktop/ball.jpg')

# Segment the image
k = 2  # Number of Gaussian components
n_iterations = 100
segmented_img, log_likelihoods, means = segment_image(img, k, n_iterations)

# Label pixels based on Gaussian models
labeled_img = np.zeros_like(segmented_img, dtype=np.uint8)
for i, mean in enumerate(means):
    labeled_img[segmented_img == i] = int(mean)

# Display the labeled image
plt.figure(figsize=(10, 5))
plt.subplot(121)
plt.imshow(img, cmap='gray')
plt.title('Original Image')
plt.axis('off')
plt.subplot(122)
plt.imshow(labeled_img, cmap='gray')
plt.title('Labeled Image')
plt.axis('off')
plt.tight_layout()
plt.show()

# Plot the convergence curve
plt.figure()
plt.plot(log_likelihoods)
plt.xlabel('Iteration')
plt.ylabel('Log-Likelihood')
plt.title('Convergence Curve')
plt.show()
```

Explanation:

The code implements a Gaussian mixture model (GMM) using the Expectation-Maximization (EM) algorithm to segment an image.

1. The image is read using the ***read_img*** function, which converts the image to grayscale.
2. The ***initialize_parameters*** function initializes the means, covariances, and mixing coefficients for the GMM. It uses k-means clustering to estimate the initial means.
3. The ***expectation_step*** function computes the responsibilities of each Gaussian component for each pixel in the image based on the current parameter estimates.
4. The ***maximization_step*** function updates the means, covariances, and mixing coefficients based on the current responsibilities.
5. The ***gaussian_prob*** function computes the probability of a given value belonging to a Gaussian distribution.
6. The ***compute_log_likelihood*** function calculates the log-likelihood of the data given the current parameter estimates.
7. The ***segment_image*** function performs the EM algorithm iteratively. It iteratively performs the expectation step and maximization step, updates the parameter estimates, and checks for convergence based on the change in log-likelihood.
8. The segmented image is obtained by finding the Gaussian component with the highest responsibility for each pixel.
9. The labeled image is created by assigning grayscale values to each pixel based on the mean of the corresponding Gaussian component.
10. The original image, labeled image, and convergence curve are displayed using Matplotlib.

Overall, the code segments the image into different regions based on intensity probabilities using a Gaussian mixture model and assigns grayscale values to pixels based on the Gaussian model they belong to. The convergence of the algorithm is checked based on the change in log-likelihood.

Output: