Expectation-maximization algorithm

Introduction

This report examines the application of the Expectation-Maximization (EM) algorithm for segmenting MRI brain scan data into three tissue classes based on pixel intensity. The task involves fitting a generative Gaussian Mixture Model (GMM) to corrected MRI data, emphasizing key steps such as parameter initialization, iterative updates, and segmentation visualization. By focusing on brain structures and excluding extraneous regions, the analysis aims to identify meaningful tissue types like gray matter, white matter, and cerebrospinal fluid.

This report provides a comprehensive walkthrough, beginning with visual exploration of the image and histogram, followed by the initialization of Gaussian components and their integration into the GMM. The next sections delve into iterative parameter updates, log-likelihood evaluation, and the interplay between model components. In addition to standard segmentation, the report explores how varying a single parameter affects the log-likelihood and demonstrates the role of the lower bound in optimization. By the end, the results highlight the effectiveness of the EM algorithm in medical imaging applications and its capacity to model complex data distributions accurately.

Input data and code hints

Import Python libraries:

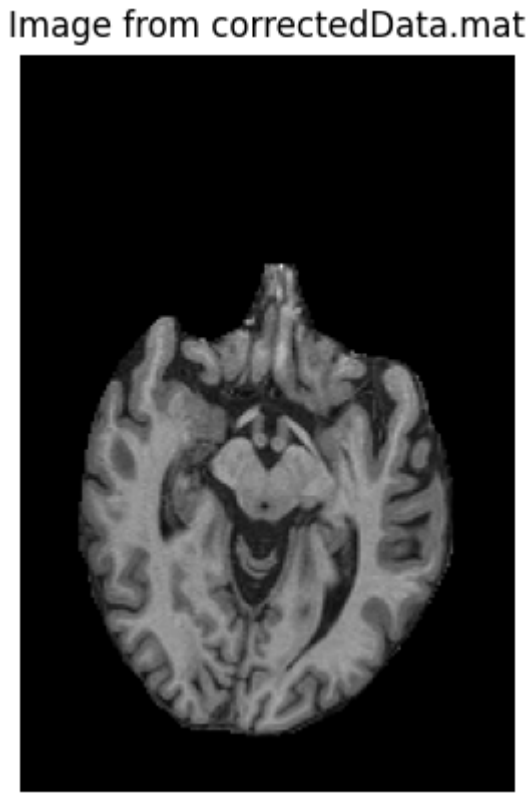
In [11]:
```
import numpy as np
import scipy
from scipy.io import loadmat
from scipy import signal
import warnings
warnings.filterwarnings("ignore")
import matplotlib
import matplotlib.pyplot as plt
from scipy.stats import norm

import matplotlib.mlab as mlab
%matplotlib inline
```

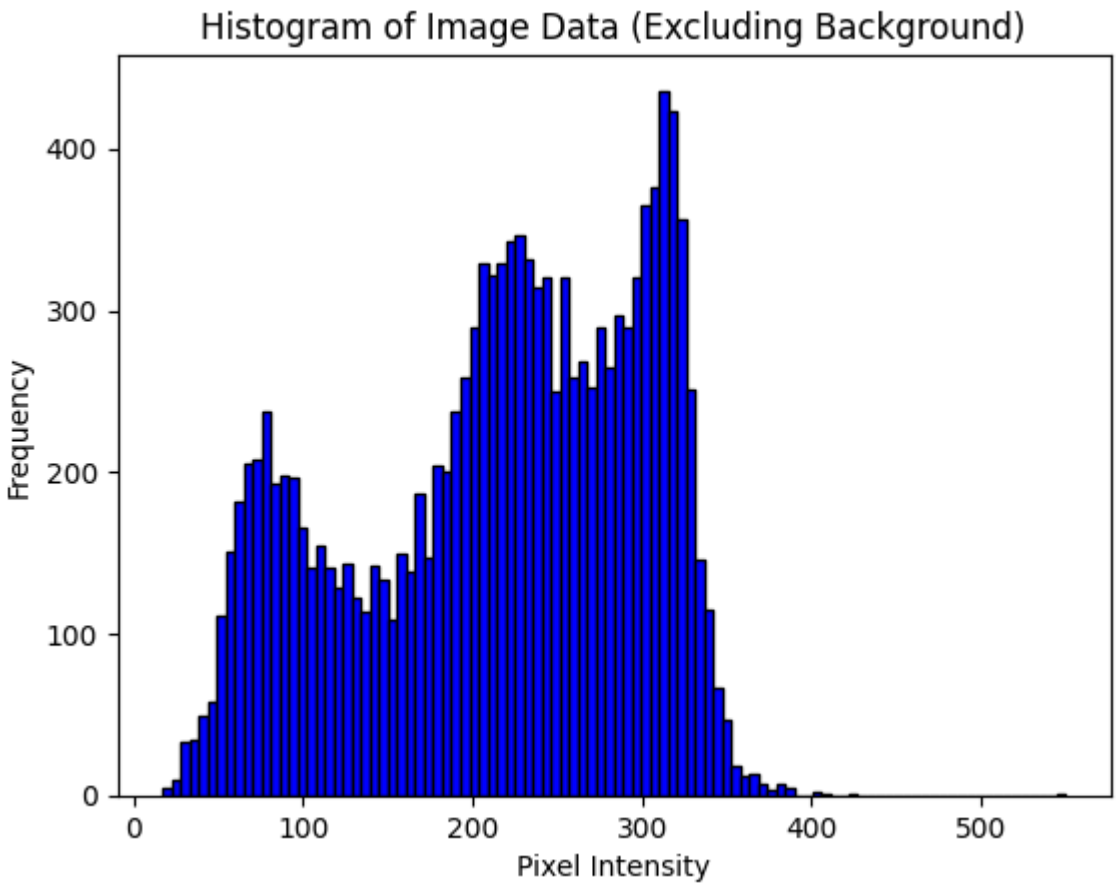
Task 1: Image data

Display the image provided in "correctedData.mat" and plot its histogram. Use 100 bins for your histogram

In [12]:
```
mat = loadmat("correctedData.mat")
data= mat["correctedData"]

plt.imshow(data, cmap='gray')
```

```
plt.title("Image from correctedData.mat")
plt.axis("off")
plt.show()

data_no_background = data[data > 0]  # Mask out zero values

plt.hist(data_no_background.flatten(), bins=100, color='blue', edgecolor='black'
plt.title("Histogram of Image Data (Excluding Background)")
plt.xlabel("Pixel Intensity")
plt.ylabel("Frequency")
plt.show()
```





The code loads MRI image data from correctedData.mat, displaying the anatomical structure of the brain using a grayscale colormap. The data is visualized using plt.imshow with the colormap set to 'gray', which highlights pixel intensities that correspond to various tissue densities. To analyze the pixel intensity distribution, we had to create a histogram, focusing only on the brain structure by excluding background pixels (intensity values of zero). This exclusion is achieved by creating a mask, data_no_background, which filters out zero-valued pixels. The histogram is generated with 100 bins to capture the range of non-zero pixel intensities, offering insight into the image's intensity variation. The result shows peaks in pixel intensity frequencies, indicating dominant intensity ranges that correlate with different tissue types in the brain which represents gray and white matter.

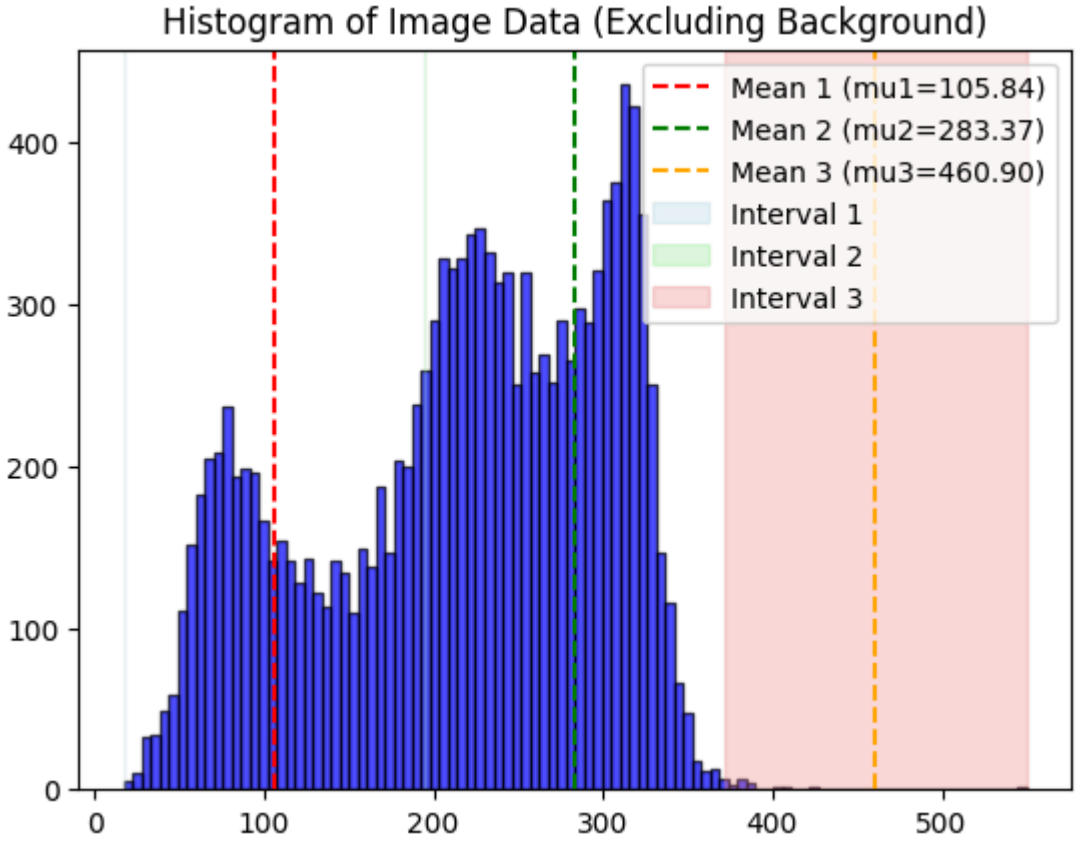
Task 2: Initialize Gaussian-Mixture Model

Compute the minimum and maximum intensity in the image (non-zero pixels), and divide the intensity range up into three equally wide intervals. Initialize the parameters of a 3-component Gaussian mixture model by setting the means of the Gaussians to the centers of the intensity intervals, the variances to the square of the width of the intervals, and the prior weights to $\frac{1}{3}$ each.

In [13]:
```
# Step 3: Compute the minimum and maximum intensity
min_intensity = np.min(data_no_background)
print("The minimum intensity", min_intensity)
max_intensity = np.max(data_no_background)
print("The maximum intensity", max_intensity)
```

```
interval_width = (max_intensity - min_intensity) / 3
mu1 = min_intensity + interval_width / 2
mu2 = min_intensity + 3 * interval_width / 2
mu3 = max_intensity - interval_width / 2
sigma1_2 = sigma2_2 = sigma3_2 = interval_width**2

# Step 4: Set prior weights to 1/3 each
pi1 = pi2 = pi3 = 1 / 3


# Print initialized parameters
print(f"mu1: {mu1}, mu2: {mu2}, mu3: {mu3}")
print(f"sigma1^2: {sigma1_2}, sigma_2^2: {sigma2_2}, sigma_3^2: {sigma3_2}")
print(f"pi1: {pi1}, pi2: {pi2}, pi3: {pi3}")

# Plot histogram
plt.hist(data_no_background.flatten(), bins=100, color='blue', edgecolor='black'
plt.title("Histogram of Image Data (Excluding Background)")

# Plot vertical lines for the means (mu1, mu2, mu3)
plt.axvline(mu1, color='red', linestyle='dashed', label=f'Mean 1 (mu1={mu1:.2f})
plt.axvline(mu2, color='green', linestyle='dashed', label=f'Mean 2 (mu2={mu2:.2f
plt.axvline(mu3, color='orange', linestyle='dashed', label=f'Mean 3 (mu3={mu3:.2

# Mark the intervals with shading
plt.axvspan(min_intensity, mu1 - interval_width / 2, color='lightblue', alpha=0.
plt.axvspan(mu1 + interval_width / 2, mu2 - interval_width / 2, color='lightgree
plt.axvspan(mu2 + interval_width / 2, max_intensity, color='lightcoral', alpha=0

# Add a legend to the plot
plt.legend()

# Show the plot
plt.show()
```

```
The minimum intensity 17.078205010324904
The maximum intensity 549.6669350144115
mu1: 105.84299334433933, mu2: 283.37257001236816, mu3: 460.90214668039704
sigma1^2: 31516.750591929533, sigma_2^2: 31516.750591929533, sigma_3^2: 31516.750
591929533
pi1: 0.3333333333333333, pi2: 0.3333333333333333, pi3: 0.3333333333333333
```



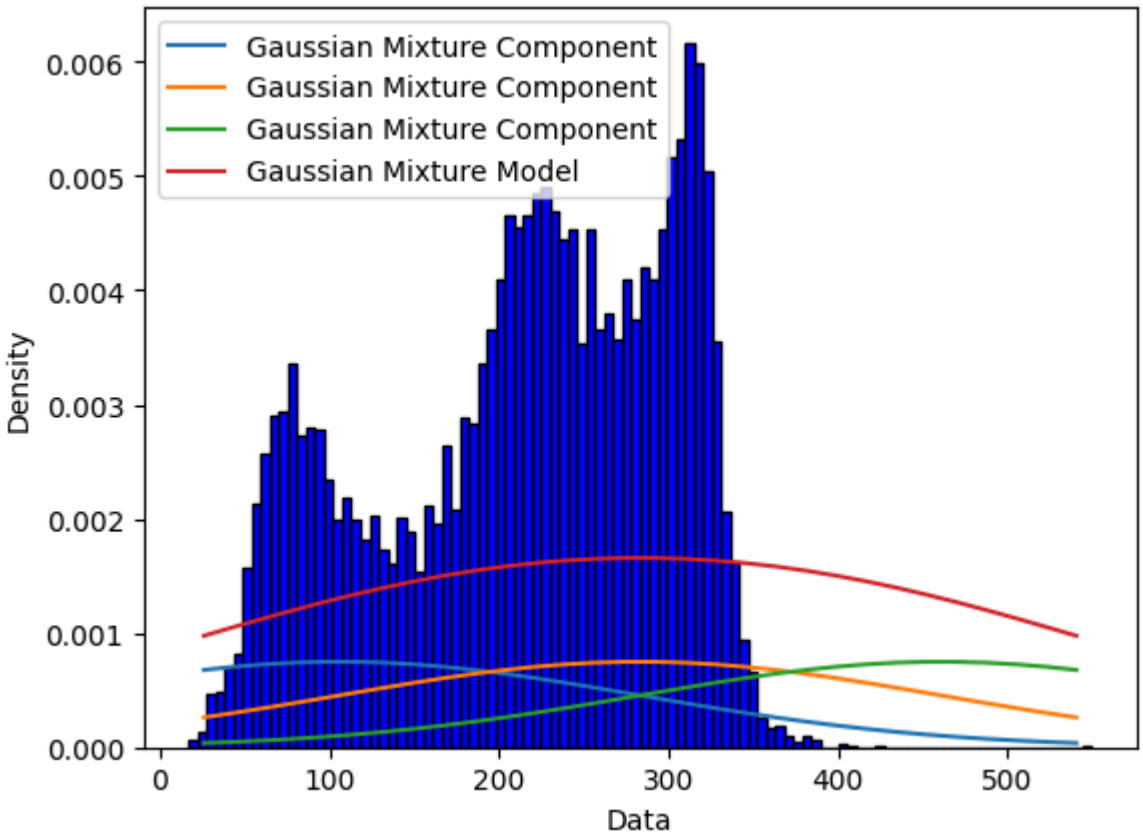
The code initializes a Gaussian Mixture Model (GMM) with three components to model the intensity distribution of the MRI image data. First, it computes the minimum and maximum intensity values from non-background pixels, which are 17.08 and 549.66, respectively. The range between these intensities is divided into three equal-width intervals, each representing a region that is meant to correspond to diferent tissue types. The means (mu1, mu2, mu3) are set at the centers of these intervals (105.84, 283.37, and 460.90), and the variances (sigma1^2, sigma2^2, sigma3^2) are set to the square of each interval's width, yielding variances of approximately 31516.75 for each component. The prior weights for each Gaussian (pi1, pi2, pi3) are set to 1/3 as asked in the task, indicating an equal likelihood for each component initially.

The histogram displays the intensity distribution with vertical dashed lines marking the means of each Gaussian, and shaded regions highlighting each interval. The first peak aligns with mu1 and represents lower-intensity tissues, the middle peak around mu2 corresponds to medium-intensity tissues, and the third peak near mu3 corresponds to higher-intensity tissues. This GMM initialization provides a starting point for more precise parameter estimation, which is crucial for segmenting the image into meaningful regions based on tissue types as we will do later.

Task 3: Display inital Gaussian distributions

Overlay the resulting Gaussian mixture model on the histogram by plotting each Gaussian weighted by its π_k, as well as the total weighted sum of all three Gaussian distributions (as in fig. 3.1(b) in the course notes).

In [4]:
```
hist, bin_edges = np.histogram(data_no_background, bins=100)
plt.hist(data_no_background.flatten(), bins=100, color='blue', edgecolor='black'
x= (bin_edges[:-1] + bin_edges[1:])/2

means=[mu1, mu2, mu3]
variances=[sigma1_2, sigma2_2, sigma3_2]
weights=[pi1, pi2, pi3]

def compute_gaussian(x,mean,var):
    return scipy.stats.norm.pdf(x,mean,np.sqrt(var))


for mean,var,weight in zip(means,variances,weights):
   plt.plot(x,weight*compute_gaussian(x,mean,var),label='Gaussian Mixture Compon

total_pdf=np.zeros_like(x)

for mean,var,weight in zip(means,variances,weights):
    total_pdf+=weight*compute_gaussian(x,mean,var)


plt.plot(x,total_pdf,label='Gaussian Mixture Model')
plt.legend()
plt.xlabel('Data')
plt.ylabel('Density')
plt.legend()
plt.show()
```



The histogram of the data was plotted to represent the observed data distribution, with each bin displaying its density. The compute_gaussian function calculates the probability density function (PDF) of a Gaussian distribution given a mean and variance, which allowed us to plot each Gaussian distribution separately. These Gaussian components

were plotted with their respective weights to reflect their contributions to the mixture model.

The code also computes the total PDF by summing the weighted PDFs of all three Gaussians across the data range, reepresenting the entire GMM. This total PDF was then plotted as the Gaussian Mixture Model on top of the histogram to show how well it approximates the observed data distribution. The plot shows each Gaussian component's individual contribution and the combined GMM. The GMM shows a smooth, continuous distribution that represents the combined probability density function obtained by summing the weighted contributions of each Gaussian component.

Task 4: Compute weights

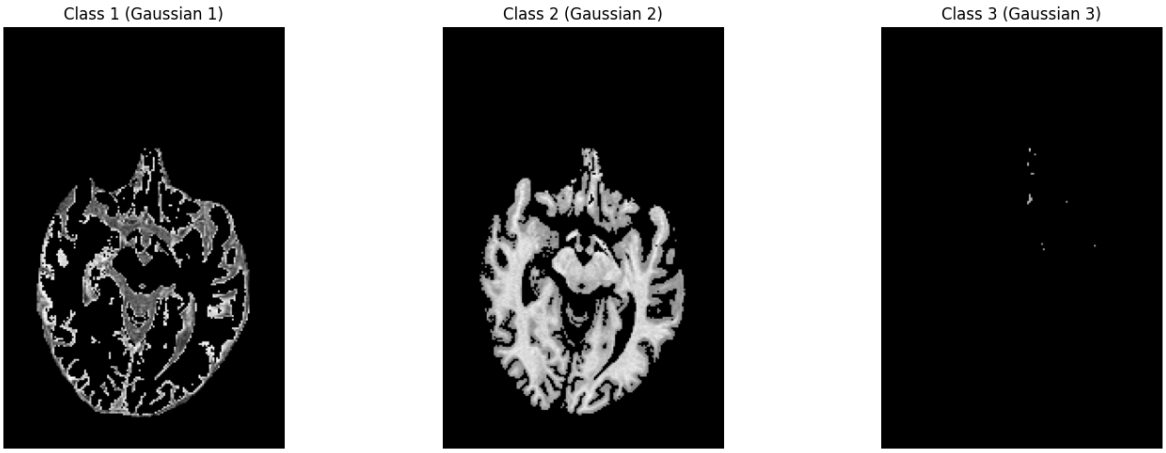
Compute the values of $w_{n,k}$ defined by:

$$w_{n,k} = \frac{\mathcal{N}(d_n|\tilde{\mu}_k, \tilde{\sigma}_k^2)\tilde{\pi}_k}{\sum_{k'=1}^{K} \mathcal{N}(d_n|\tilde{\mu}'_k, \tilde{\sigma}_{k'}^2)\tilde{\pi}'_k}.$$

Visualize the values of each $w_{n,k}$ in a figure. You should have three plots which each display only the pixels that belong to the respective class, according to the weights. The segmentation of a pixel is determined by assigning the class of the highest probability.

In [5]:
```
def computing_weights(data, means, variances, weights):
    # Initialize responsibilities array with shape (number of data points, 3 com
    responsibilities = np.zeros((len(data), 3))

    # Iterate over each Gaussian component
    for k in range(3):
        pdf_k = compute_gaussian(data, means[k], variances[k])  # shape should m
        responsibilities[:, k] = weights[k] * pdf_k  # Broadcasting weights[k] o

    # Normalize responsibilities to get w_n,k (posterior probabilities)
    denominator = np.sum(responsibilities, axis=1)[:, np.newaxis]  # shape (len(
    w_nk = responsibilities / denominator  # Divide each row by its correspondin

    return w_nk


w_nk = computing_weights(data_no_background,means,variances,weights)


max_class = np.argmax(w_nk, axis=1)

fig, axes = plt.subplots(1, 3, figsize=(15, 5))

for k in range(3):

    mask = (max_class == k)

    class_image = np.zeros_like(data)
    class_image[data > 0] = mask * data_no_background


    axes[k].imshow(class_image, cmap='gray')
```

```
    axes[k].set_title(f"Class {k+1} (Gaussian {k+1})")
    axes[k].axis('off')

plt.tight_layout()
plt.show()
```



Using the function computing_weights, the code computes these weights by first evaluating the probability density function (PDF) of each Gaussian component, given its mean and variance, and then weighting it according to the component's prior probability (weight) in the mixture. The formula given, normalizes these weighted PDFs across all components, producing a probability distribution across the classes for each pixel. Next, np.argmax is used to assign each pixel to the class with the highest probability, resulting in a segmentation map that groups pixels by Gaussian component. The three images generated display masks, where each one corresponds to a specific Gaussian component, highlighting only the pixels most likely to belong to that class. In these images, "Class 1," "Class 2," and "Class 3" represent the areas of the image assigned to the first, second, and third Gaussian components, respectively. These masks reveal the regions of the image that each Gaussian component captures, with different brightness and intensity levels indicating the variations in pixel values across the image, which correspond to different structural features identified by the GMM.

Task 5: Estimate Maximum Likelihood parameters

Estimate the maximum likelihood parameters by iterating between updating the model parameter estimate according to

$$\tilde{\mu}_k \leftarrow \frac{\sum_{n=1}^{N} w_{n,k} d_n}{\sum_{n=1}^{N} w_{n,k}}$$

$$\tilde{\sigma}_k^2 \leftarrow \frac{\sum_{n=1}^{N} w_{n,k} (d_n - \tilde{\mu}_k)^2}{\sum_{n=1}^{N} w_{n,k}}$$

$$\tilde{\pi}_k \leftarrow \frac{\sum_{n=1}^{N} w_{n,k}}{N}$$

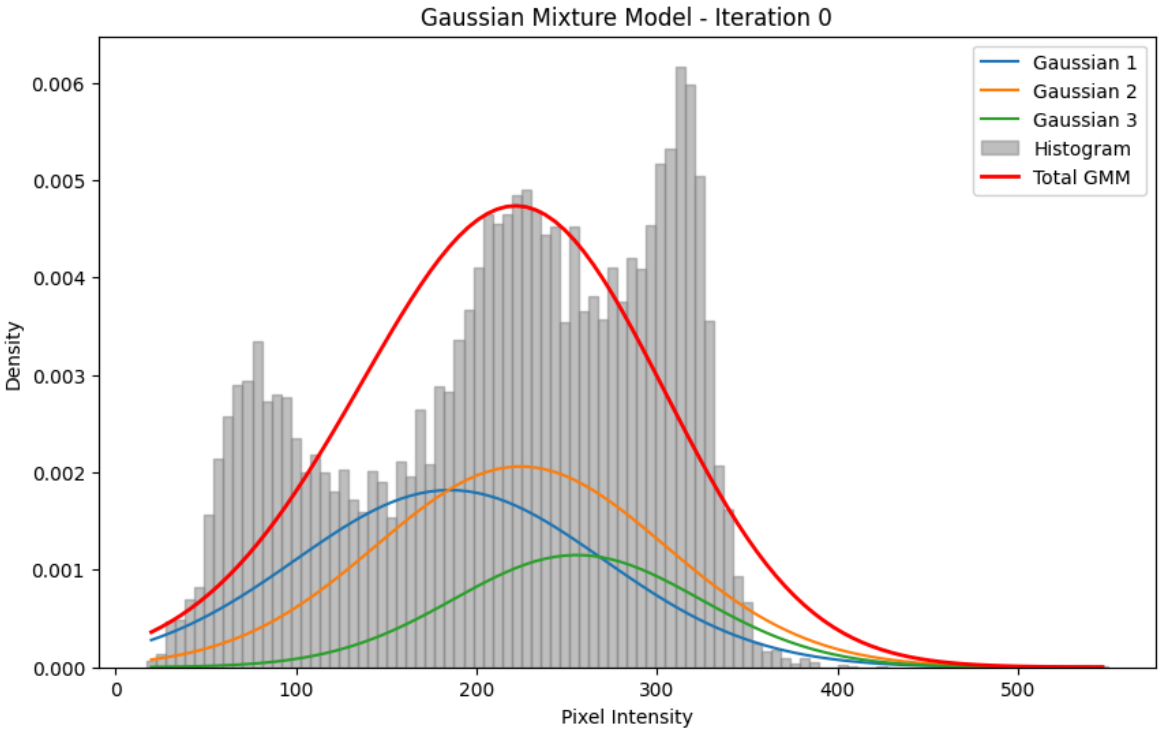
and by recomputing $w_{n,k}$ according to eq. 3.33 (see Task 4).

Make sure to perform enough iterations (e.g., 100) for the algorithm to converge. As the iterations progress, plot the evolution of the log likelihood function, and update each time the display of $w_{n,k}$ as well as the Gaussian mixture model plot overlaid on the histogram. Include the evolution of the log likelihood function and the plot of the final $w_{n,k}$ and the final mixture model in your report.

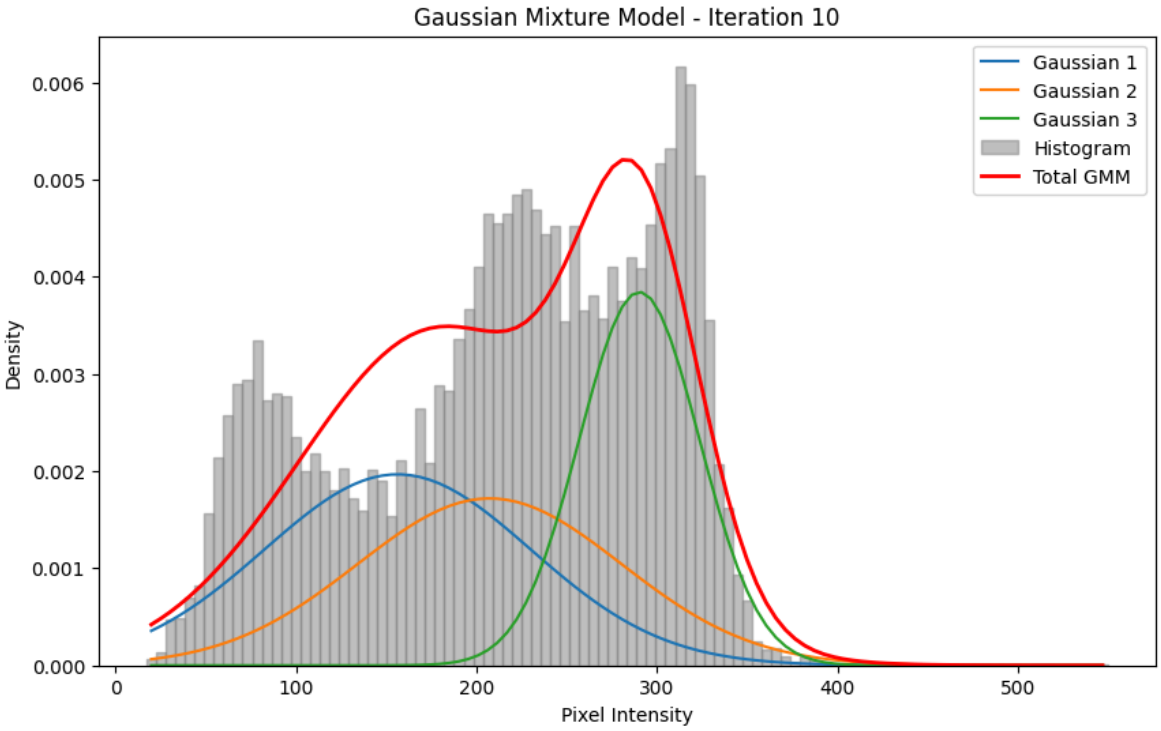
In [7]:
```
def compute_log_likelihood(data, means, variances, weights):
    responsibilities = computing_weights(data, means, variances, weights)
    log_likelihood = np.sum(np.log(np.sum(responsibilities* weights,axis=1 )))
    return log_likelihood, responsibilities

# EM Algorithm parameters and initialization
iterations = 110
log_likelihoods = []
param_history = {'means': [], 'variances': [], 'weights': []}

# Initialization of parameters (means, variances, weights)
means = [mu1, mu2, mu3]
variances = [sigma1_2, sigma2_2, sigma3_2]
weights = [pi1, pi2, pi3]

# EM Iterations
for iteration in range(iterations):
    # E-step: Compute responsibilities w_n,k
    log_likelihood, responsibilities = compute_log_likelihood(data_no_background
    log_likelihoods.append(log_likelihood)

    # M-step: Update parameters based on responsibilities
    N = len(data_no_background)
    for k in range(3):
        # Update the means
        means[k] = np.sum(responsibilities[:, k] * data_no_background) / np.sum(
        # Update the variances
        variances[k] = np.sum(responsibilities[:, k] * (data_no_background - mea
        # Update the weights
        weights[k] = np.sum(responsibilities[:, k]) / N

    # Store the parameters after each iteration
    param_history['means'].append(means.copy())
    param_history['variances'].append(variances.copy())
    param_history['weights'].append(weights.copy())

    # Display the log-likelihood evolution and the GMM plot every 10 iterations
    if iteration % 10 == 0:
        print(f"Iteration {iteration}, Log-Likelihood: {log_likelihood}")

        # Plot the GMM overlay on the histogram
        plt.figure(figsize=(10, 6))
        hist, bin_edges = np.histogram(data_no_background, bins=100, density=Tru
        x = (bin_edges[:-1] + bin_edges[1:]) / 2  # Bin centers
        total_gmm = np.zeros_like(x)

        # Plot each Gaussian component
        for k in range(3):
            gaussian = weights[k] * compute_gaussian(x, means[k], variances[k])
            total_gmm += gaussian
            plt.plot(x, gaussian, label=f"Gaussian {k+1}")
```

```
        plt.hist(data_no_background, bins=100, density=True,alpha=0.5, color='gr
        plt.plot(x, total_gmm, color='red', linewidth=2, label="Total GMM")
        plt.title(f"Gaussian Mixture Model - Iteration {iteration}")
        plt.xlabel("Pixel Intensity")
        plt.ylabel("Density")
        plt.legend()
        plt.show()
```

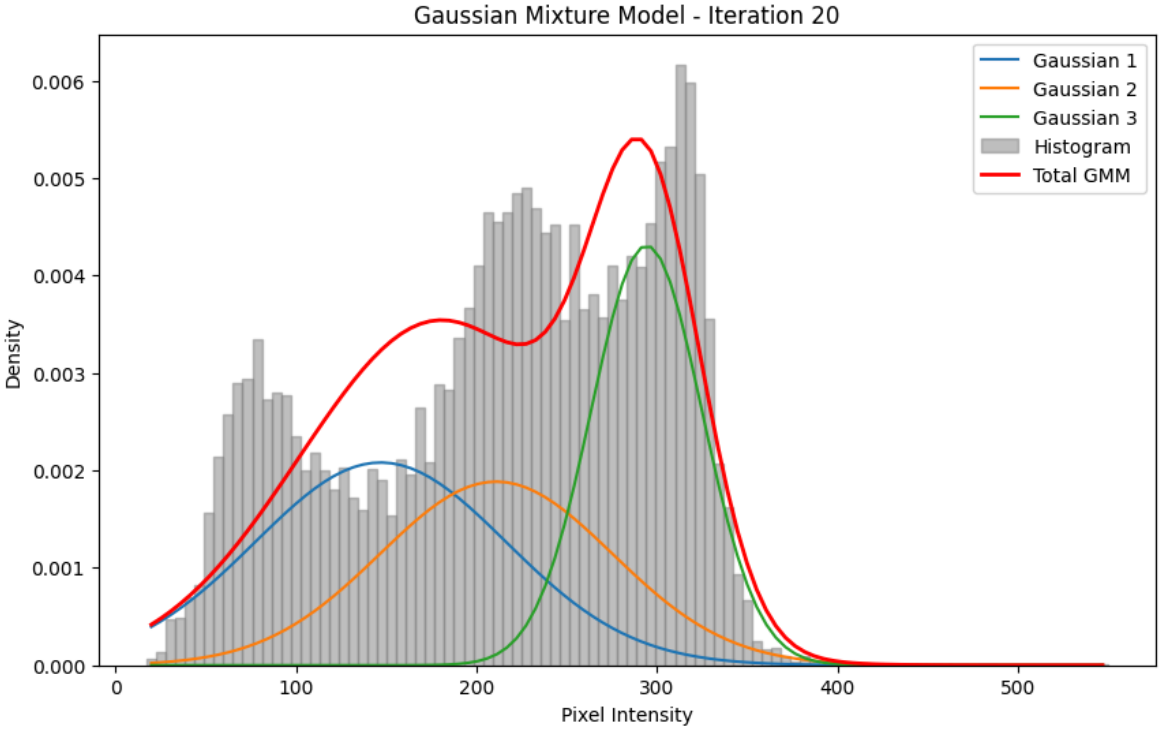
```
Iteration 0, Log-Likelihood: -14580.782295203151
```



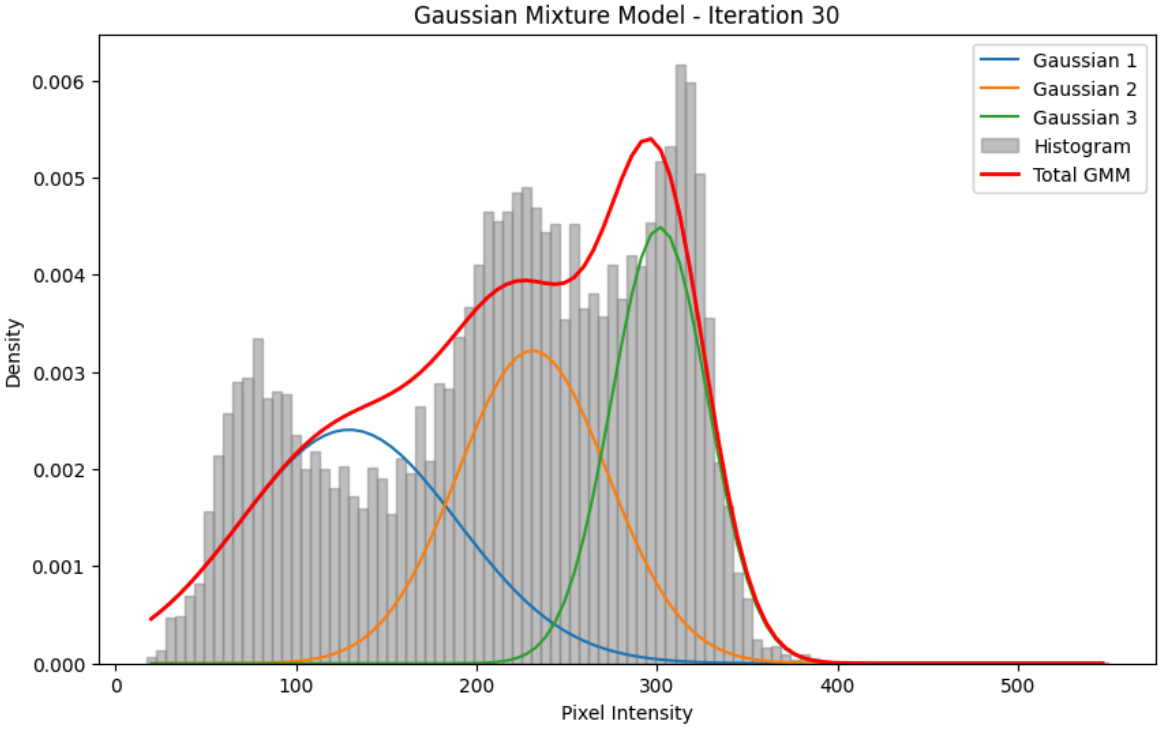

```
Iteration 10, Log-Likelihood: -14534.341872543875
```



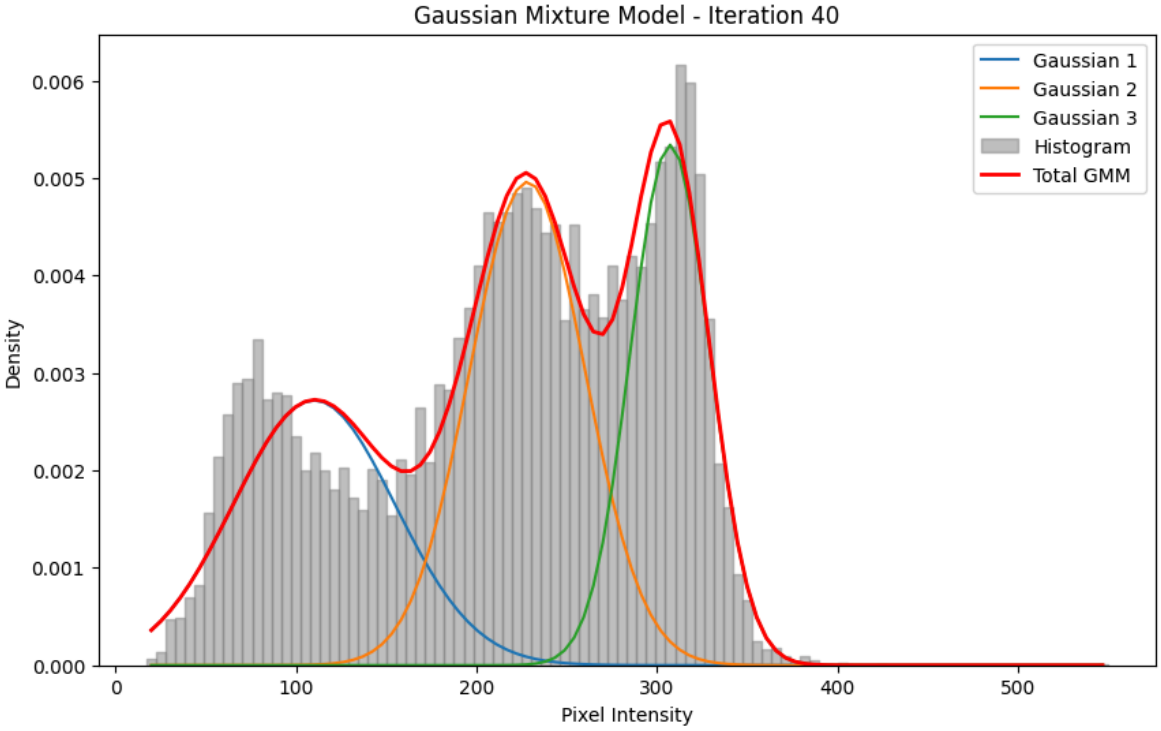

```
Iteration 20, Log-Likelihood: -14516.551209560834
```



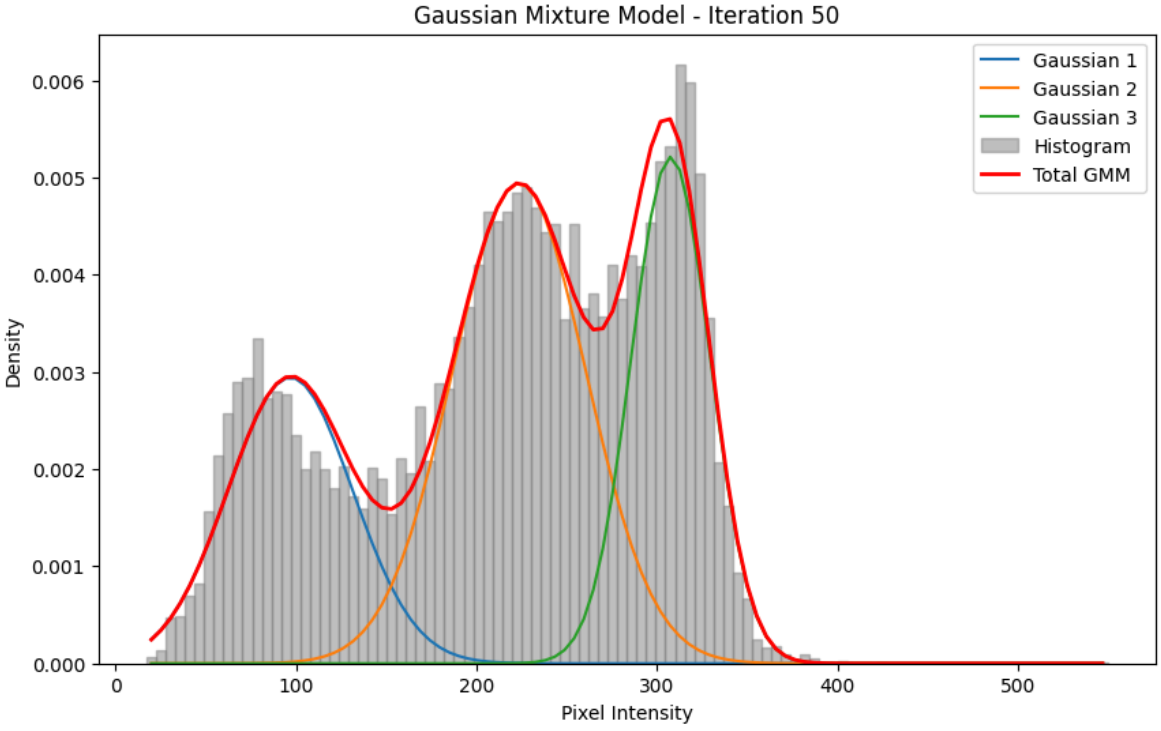

```
Iteration 30, Log-Likelihood: -14532.444920726994
```



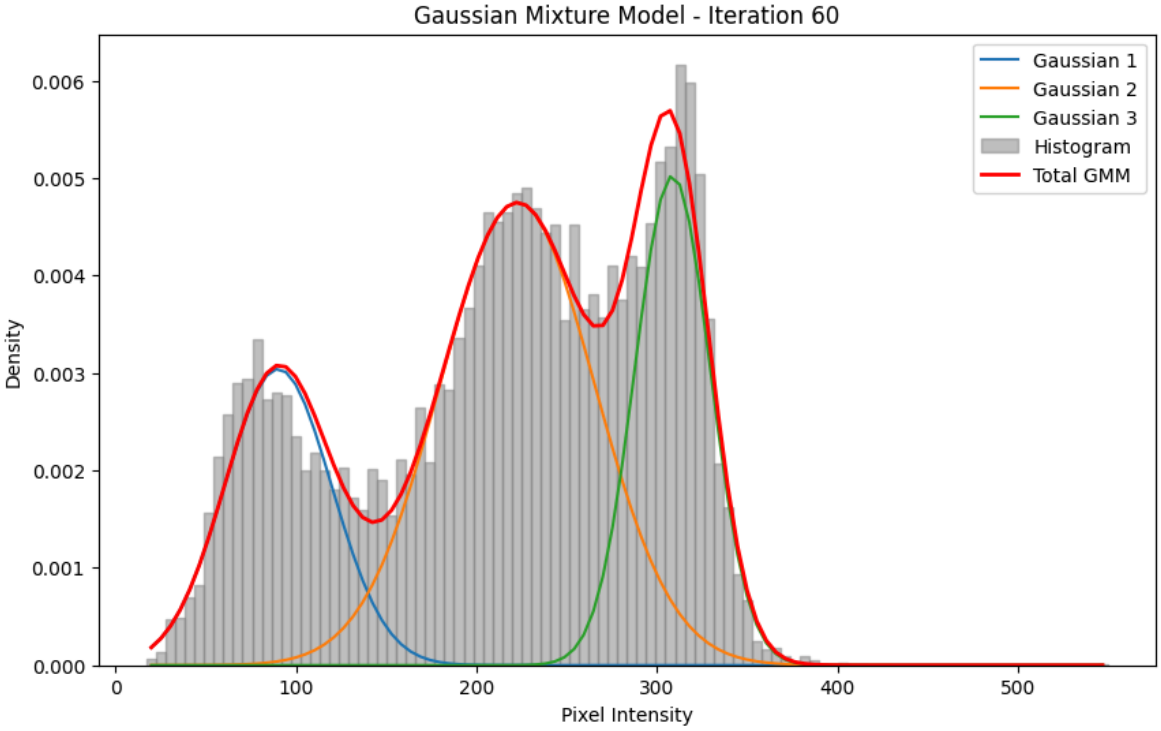

```
Iteration 40, Log-Likelihood: -14415.929691820758
```



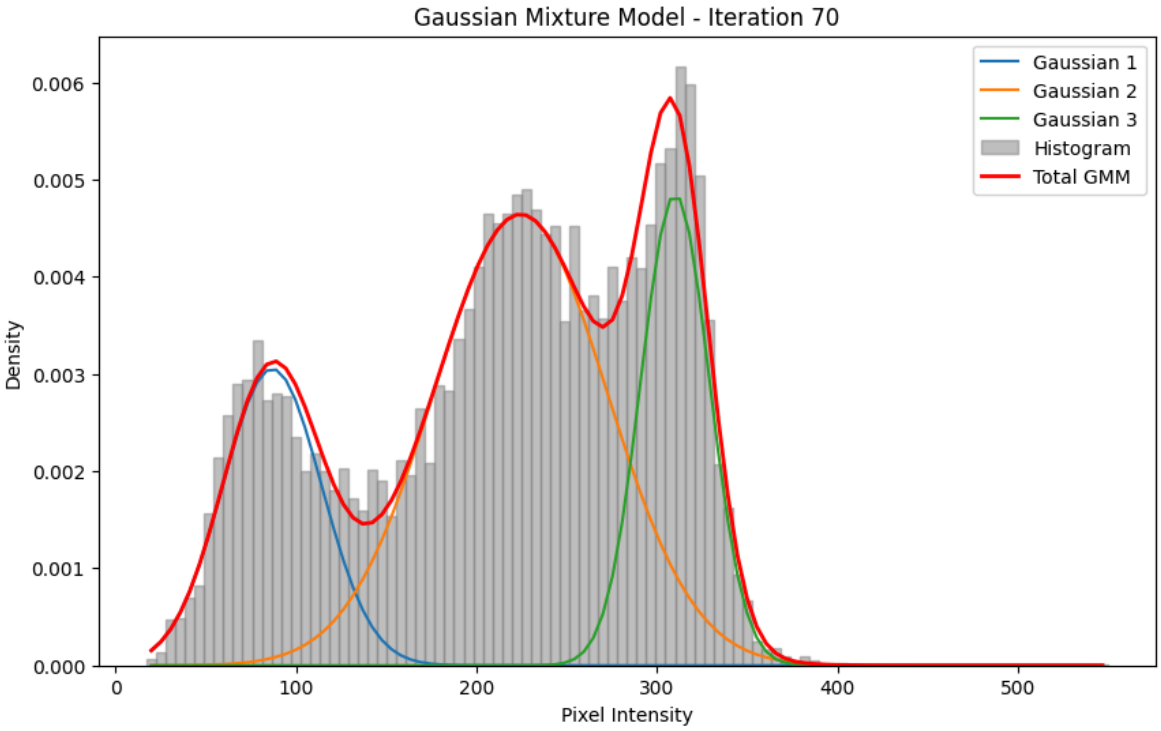

```
Iteration 50, Log-Likelihood: -14010.23263138193
```



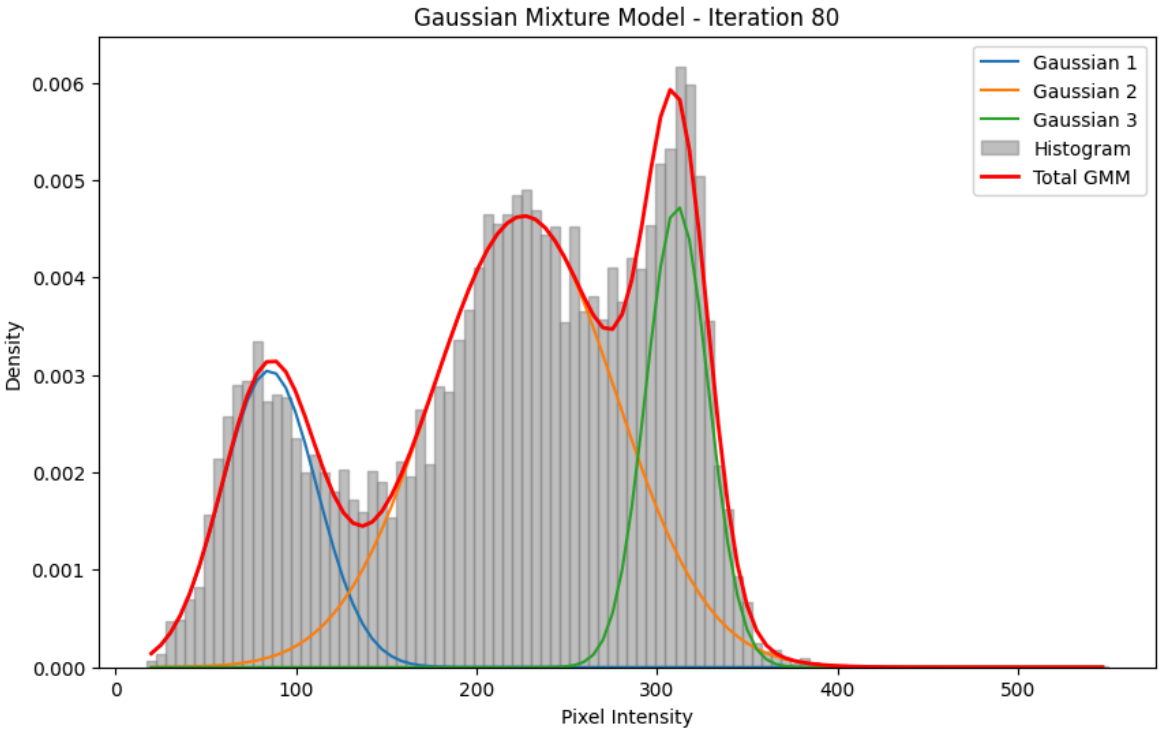

```
Iteration 60, Log-Likelihood: -13453.36002075683
```



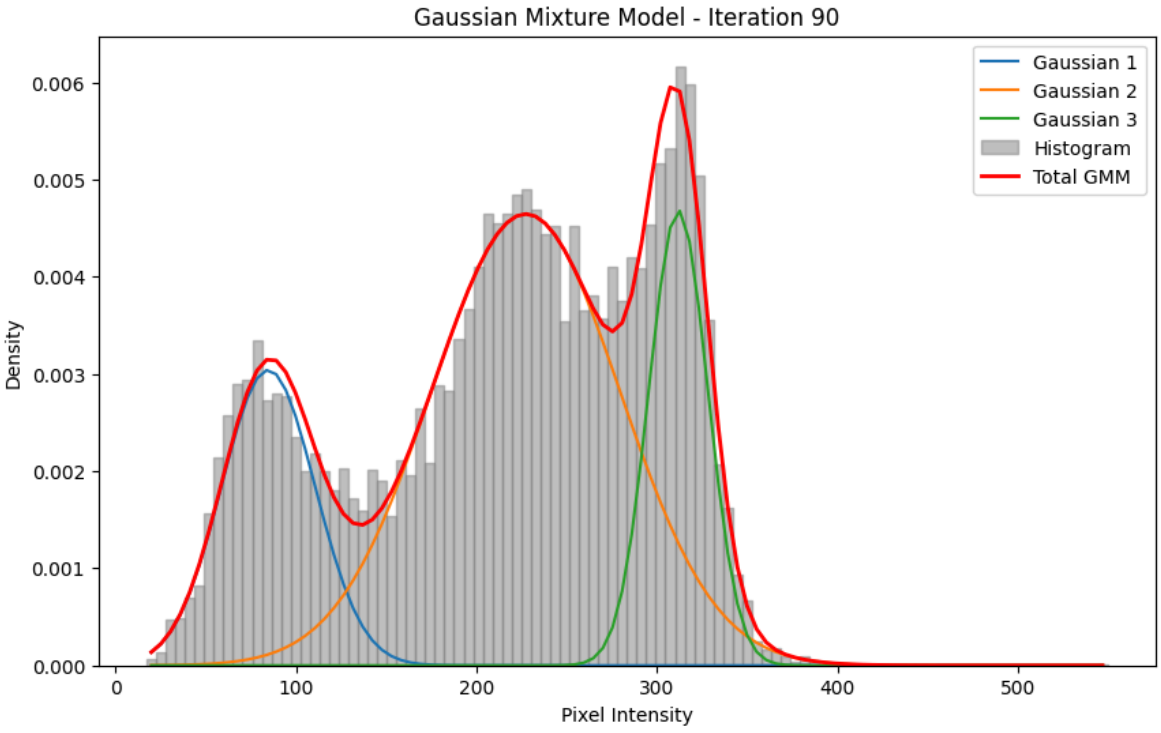

```
Iteration 70, Log-Likelihood: -12733.01756965972
```



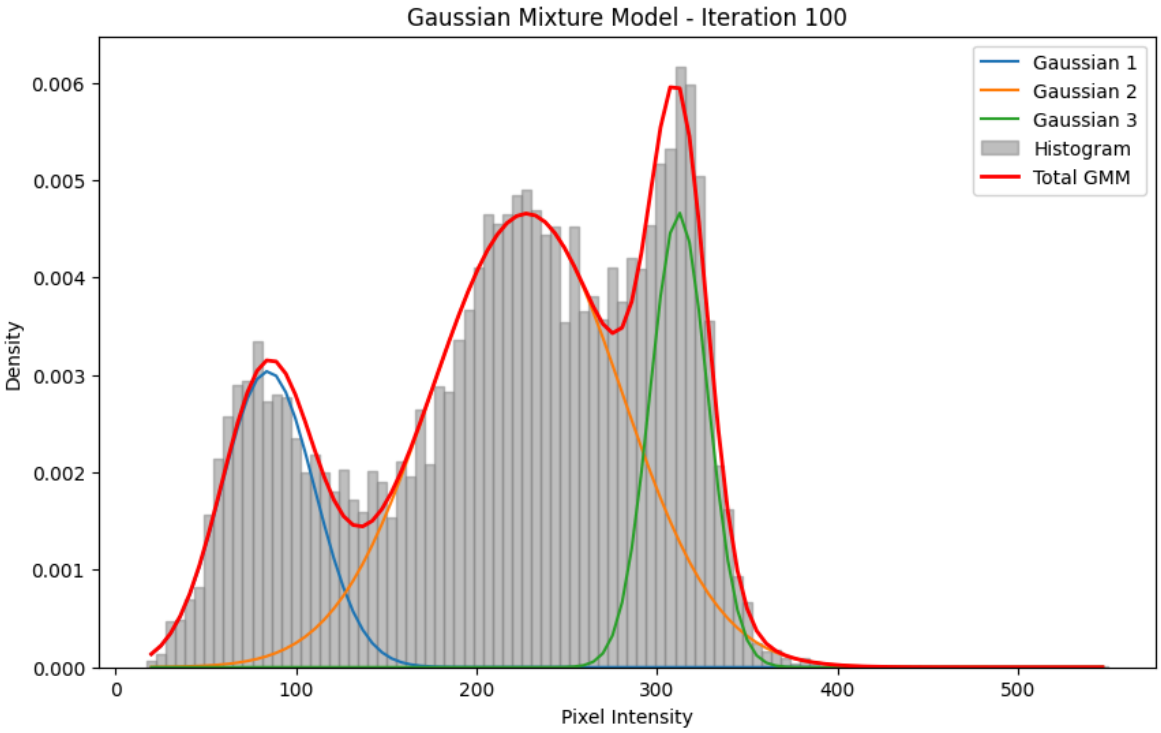

```
Iteration 80, Log-Likelihood: -12084.481577114322
```




```
Iteration 90, Log-Likelihood: -11730.264465014445
```



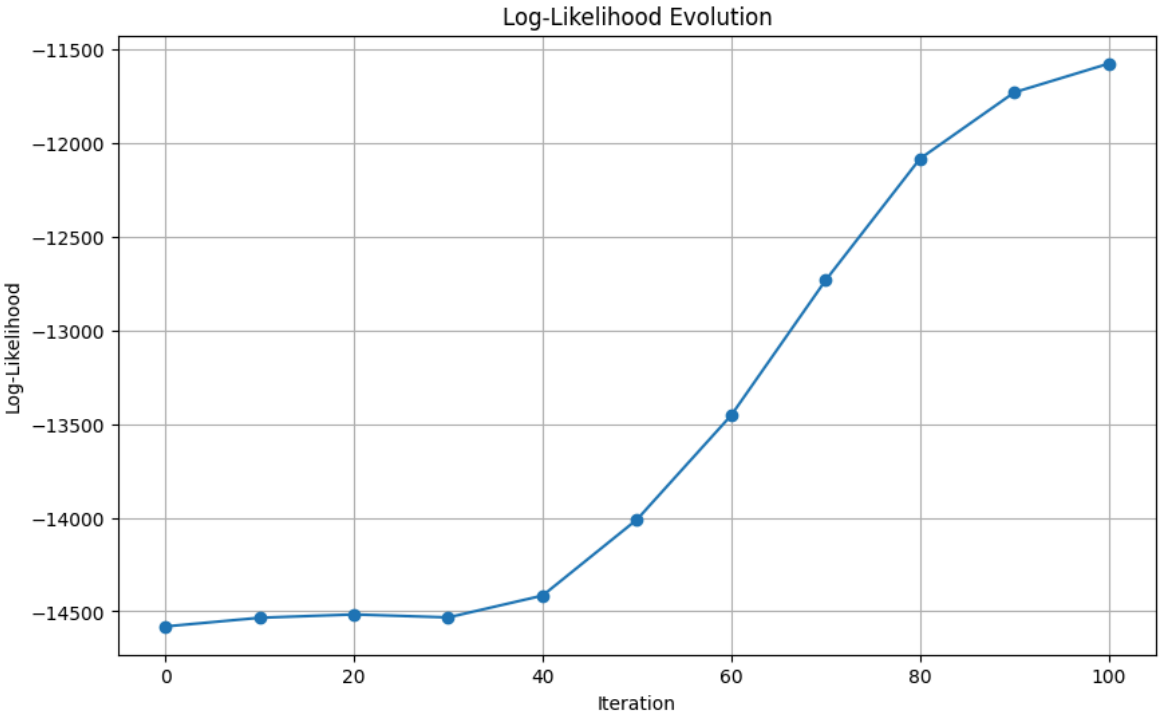

```
Iteration 100, Log-Likelihood: -11576.934243427593
```



In this code, we are implementing the Expectation-Maximization (EM) algorithm to estimate the maximum likelihood parameters for a Gaussian Mixture Model (GMM) that fits the distribution of pixel intensities in an image. The goal is to iteratively update the model's parameters: means, variances, and weights of the Gaussian components so that the GMM closely approximates the underlying histogram of pixel intensities. Each iteration consists of two steps: the E-step, where we calculate the responsibility weights (probabilities that each pixel belongs to each Gaussian component) based on the current parameters, and the M-step, where we update the parameters using these weights according to maximum likelihood estimation formula. This iterative process aims to maximize the log likelihood, which quantifies how well the GMM fits the data. As the iterations progress (by adding 10 iterations at a time), the GMM plot becomes increasingly similar to the histogram of pixel intensities, indicating a better fit as we can clearly see in iteration 100.

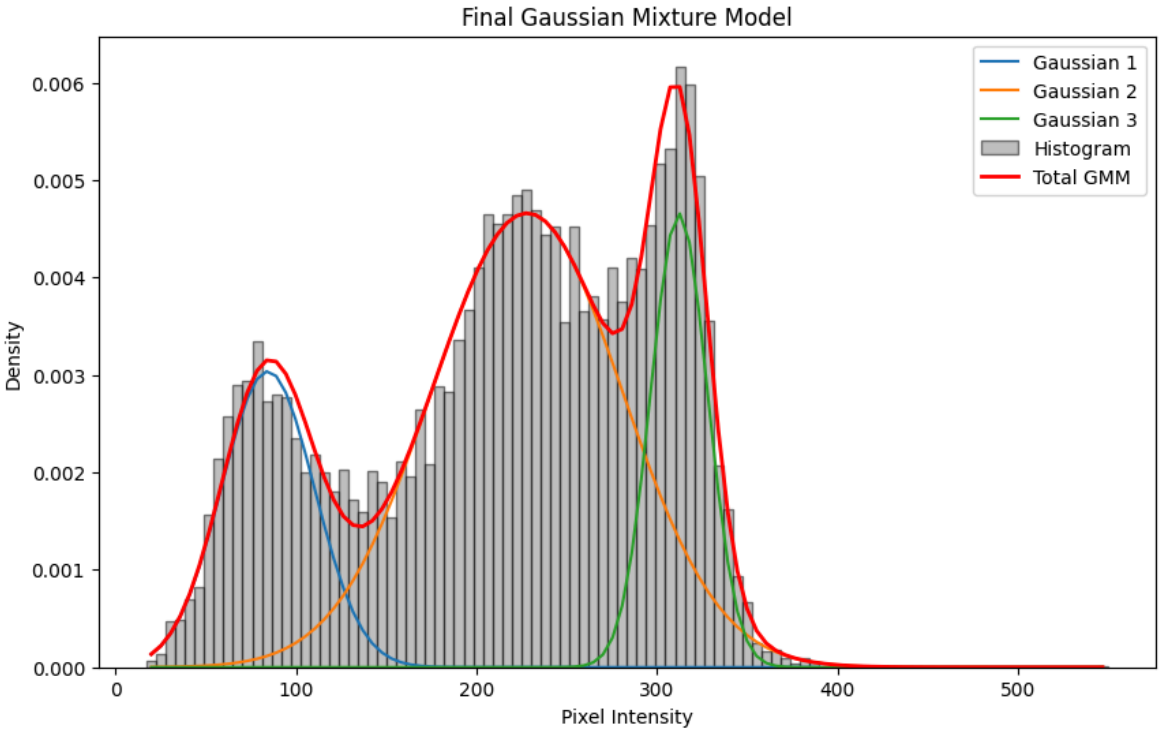
In [8]:
```
# Plot the final log-likelihood
plt.figure(figsize=(10, 6))
iterations = iterations = np.arange(0, 110, 10)
log_likelihood_subset = np.array(log_likelihoods)[iterations]
print(len(log_likelihoods))
plt.plot(iterations, log_likelihood_subset, marker='o')
plt.title("Log-Likelihood Evolution")
plt.xlabel("Iteration")
plt.ylabel("Log-Likelihood")
plt.grid(True)
plt.show()
```

```
110
```



This convergence is also reflected in the evolution of the log likelihood, which becomes progressively less negative with more iterations, meaning the likelihood of the observed data given the model parameters is increasing. This trend signifies that the GMM is improving its representation of the data distribution. By the final iterations, the total GMM model effectively approximates the histogram, indicating that the algorithm has reached a point where the model parameters best describe the data distribution.

In [9]:
```
# Final GMM after convergence
plt.figure(figsize=(10, 6))
hist, bin_edges = np.histogram(data_no_background, bins=100, density=True)
x = (bin_edges[:-1] + bin_edges[1:]) / 2  # Bin centers
total_gmm = np.zeros_like(x)

for k in range(3):
    gaussian = weights[k] * norm.pdf(x, means[k], np.sqrt(variances[k]))
    total_gmm += gaussian
    plt.plot(x, gaussian, label=f"Gaussian {k+1}")

plt.hist(data_no_background, bins=100, density=True, alpha=0.5, color='gray', ed
plt.plot(x, total_gmm, color='red', linewidth=2, label="Total GMM")
plt.title("Final Gaussian Mixture Model")
plt.xlabel("Pixel Intensity")
plt.ylabel("Density")
plt.legend()
plt.show()
```



This code generates a plot showing the final Gaussian Mixture Model (GMM) after the EM algorithm has converged. The purpose of this plot is to visualize how well the GMM fits the distribution of pixel intensities in the data. For each component, norm.pdf calculates the Gaussian PDF at each x point, and this PDF is scaled by the component's weight. The weighted Gaussian PDF for each component is then added to total_gmm, which accumulates the contributions from all components to create the final GMM distribution. Each individual Gaussian component is plotted as a separate line on the graph to show its shape and position within the overall GMM just like before. Ideally, the "Total GMM" curve should align with the histogram, indicating a good fit which in our case we could say it does.

Task 6: Vary only one model parameter

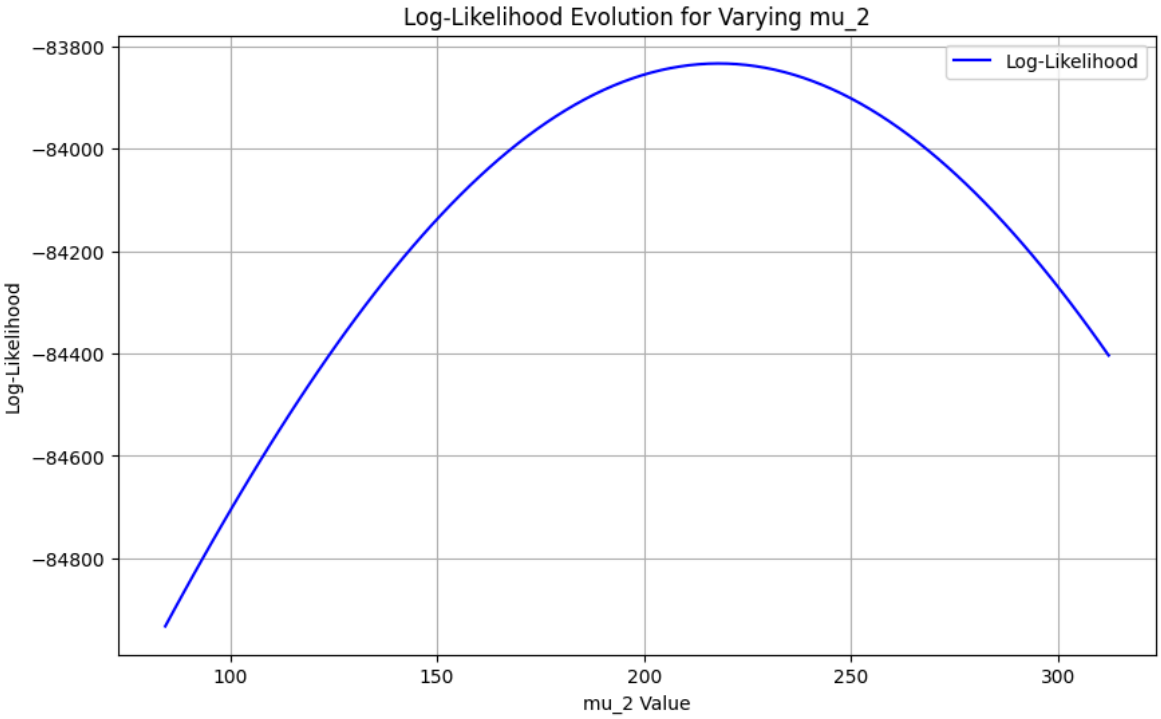
Keeping all other parameters fixed to their estimated values, vary only μ_2 (the mean of the middle Gaussian distribution) between the estimated values of μ_1 and μ_3 in about 100 steps, and plot for each step the log likelihood function.

In [12]:
```
variances_i=[sigma1_2, sigma2_2, sigma3_2]
weights_i=[pi1, pi2, pi3]

# Compute responsibilities with mu2 = mu1 for lower bound
fixed_means = np.array([means[0], means[0], means[2]])
fixed_responsibilities = computing_weights(data_no_background, fixed_means, vari

# Define the range for mu2
mu2_range = np.linspace(means[0], means[2], 100)
```

In [13]:
```
def compute_log_likelihood(data, means, variances, weights):
    N = data.shape[0]
    K = means.shape[0]
```

```
    log_likelihood = 0.0
    for n in range(N):
        tmp = 0.0
        for k in range(K):
            tmp += weights[k] * norm.pdf(data[n], loc=means[k], scale=np.sqrt(va
        log_likelihood += np.log(tmp + 1e-12)  # Numerical stability

    return log_likelihood
```

In [16]:
```
log_likelihoods_mu2 = []
for mu2 in mu2_range:
    temp_means = np.array([means[0], mu2, means[2]])

    # Compute log-likelihood
    log_likelihood = compute_log_likelihood(data_no_background, temp_means, vari
    log_likelihoods_mu2.append(log_likelihood)

plt.figure(figsize=(10, 6))
plt.plot(mu2_range, log_likelihoods_mu2, label="Log-Likelihood", color="blue")
plt.title("Log-Likelihood Evolution for Varying mu_2")
plt.xlabel("mu_2 Value")
plt.ylabel("Log-Likelihood")
plt.legend()
plt.grid(True)
plt.show()
```



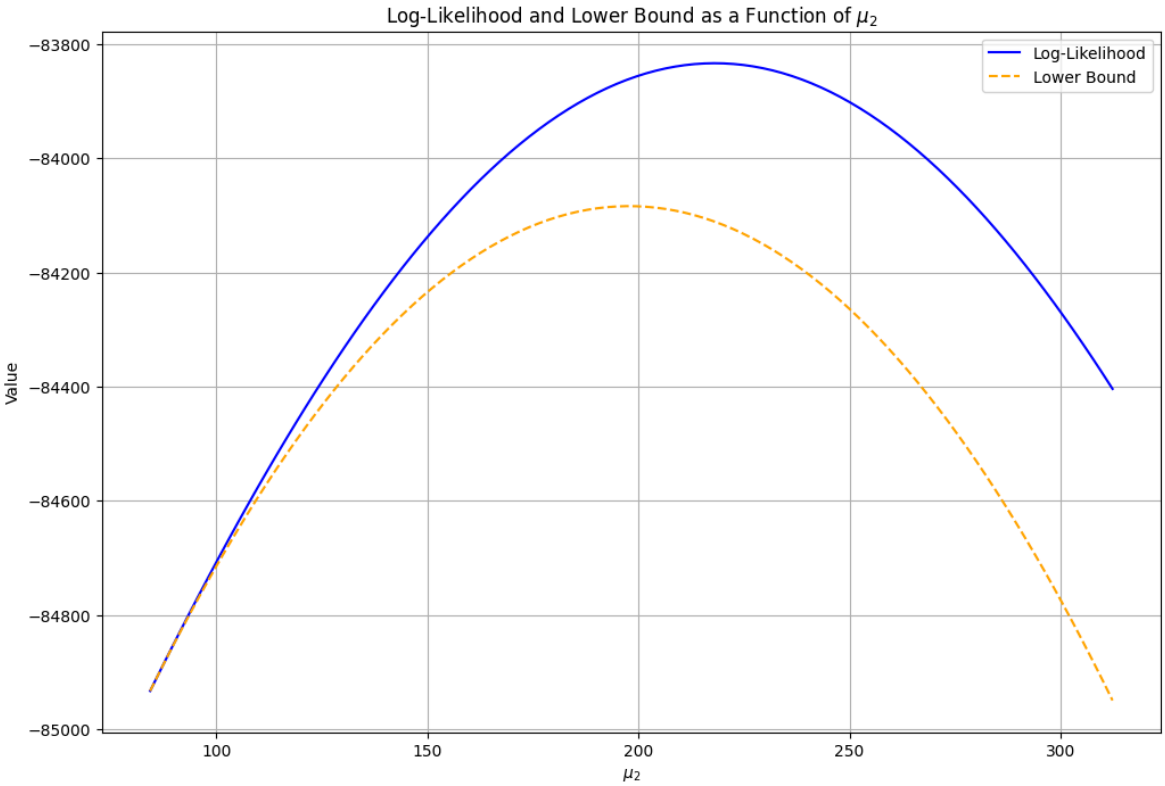
We defined a range of values for $\mu 2$ between the first and third elements of the means array, with a step size of 0.01. In a loop, we iteratively updated $\mu 2$ in the means array and computed the log-likelihood for each corresponding value by calling the compute_log_likelihood function and the resulting log-likelihood values were stored in the log_likelihoods list. Finally, we plotted the log-likelihoods against the values of $\mu 2$. The log-likelihood's peak suggests an optimal region for μ2 where the GMM fits the data most effectively. Moving it too close to $\mu 1$ or $\mu 3$ reduces the fit, as indicated by lower log-likelihood values.

Task 7: Locate Lower bound

On the same figure, also plot the lower bound corresponding to the parameter vector in which all parameters are set to their estimated values, except μ_2 which is set to the estimated value of μ_1.

In [18]:
```
def compute_lower_bound(data, means, variances, weights, responsibilities):
    N, K = responsibilities.shape

    Q = 0.0

    # Weighted log priors and Gaussian probabilities
    for k in range(K):
        Q += np.sum(responsibilities[:, k] * (np.log(weights[k] + 1e-12) +
                                              -0.5 * np.log(2 * np.pi * variances
                                              0.5 * ((data - means[k]) ** 2) / va

    # Entropy
    Q -= np.sum(responsibilities * np.log(responsibilities + 1e-12))

    return Q
```

In [ ]:
```
# Initialize lists to store log-likelihoods and lower bounds
lower_bounds_mu2 = []

for mu2 in mu2_range:
    temp_means = np.array([means[0], mu2, means[2]])

    # Compute lower bound
    lower_bound_value = compute_lower_bound(data_no_background, temp_means, vari
    lower_bounds_mu2.append(lower_bound_value)

# Convert lists to numpy arrays for easier handling
log_likelihoods_mu2 = np.array(log_likelihoods_mu2)
lower_bounds_mu2 = np.array(lower_bounds_mu2)

# Plotting Tasks 6-7
plt.figure(figsize=(12, 8))
plt.plot(mu2_range, log_likelihoods_mu2, label="Log-Likelihood", color='blue')
plt.plot(mu2_range, lower_bounds_mu2, label="Lower Bound", color='orange', lines

# Labels and Title
plt.title("Log-Likelihood and Lower Bound as a Function of $\\mu_2$")
plt.xlabel("$\\mu_2$")
plt.ylabel("Value")
plt.legend()
plt.grid(True)
plt.show()
```



In this task we demonstrated the relationship between the log-likelihood and its lower bound. The lower bound provides a valid approximation to the log-likelihood, which is essential for optimization in EM algorithms. The visualization confirms that the lower bound is tightest at the estimated optimal parameter values.

Task 8: Maximize lower bound

Compute the value for μ_2 that maximizes this lower bound (first line of eq. 3.35), indicate its location on the figure, and comment on the result.

Formulate the lower bound as

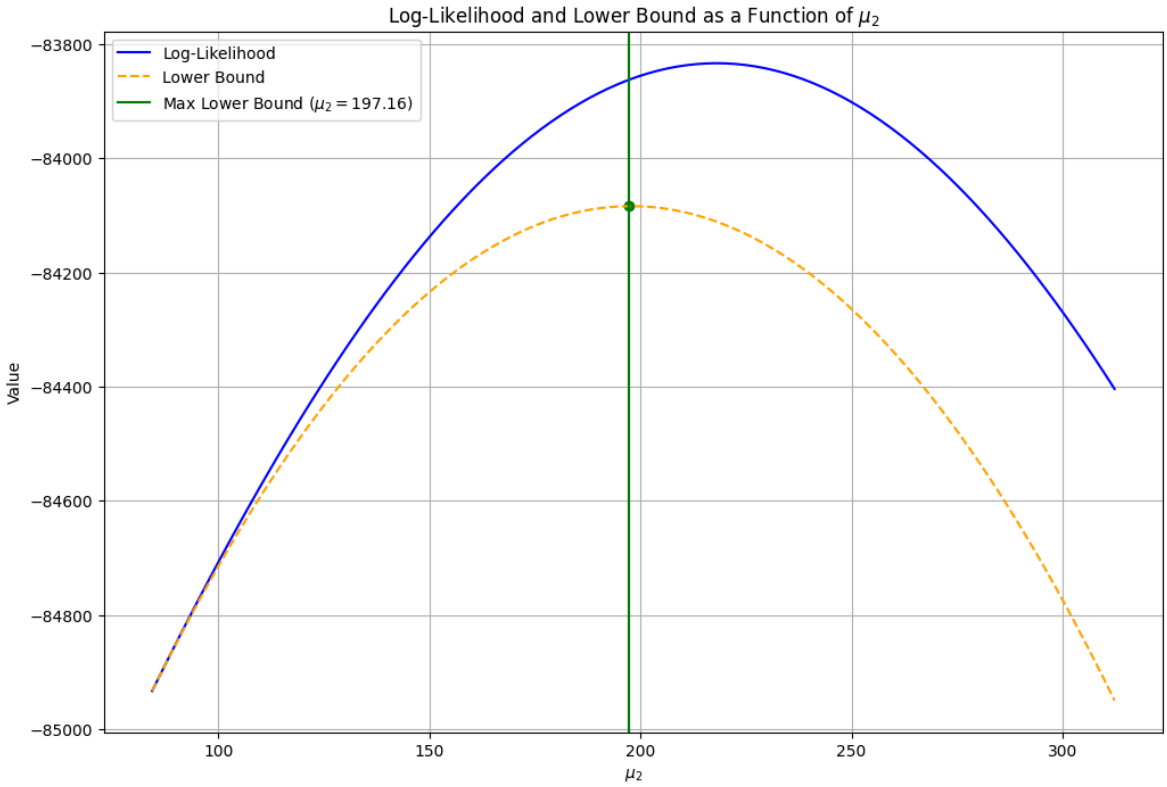
$$Q(\theta|\tilde{\theta}) = -\frac{1}{2}\sum_{k=1}^{K}[\frac{1}{\sigma_k^2}\sum_{n=1}^{N} w_{n,k}(d_n - \mu_k - \sum_{m-1}^{M} c_m\phi_{n,m})^2 + (\sum_{n-1}^{N} w_{n,k})\log\sigma_k^2]$$

$$+\sum_{k=1}^{K}[(\sum_{n=1}^{N} w_{n,k})\log\pi_k]$$

$$-\sum_{n=1}^{N}\sum_{k=1}^{K} w_{n,k}\log w_{n,k} - \frac{N}{2}\log(2\pi)$$

In [20]:
```
# Locate the mu2 that maximizes the lower bound
max_lb_index = np.argmax(lower_bounds_mu2)
optimal_mu2 = mu2_range[max_lb_index]
max_lower_bound = lower_bounds_mu2[max_lb_index]

# Plotting Tasks 6-8
plt.figure(figsize=(12, 8))
plt.plot(mu2_range, log_likelihoods_mu2, label="Log-Likelihood", color='blue')
```

```
plt.plot(mu2_range, lower_bounds_mu2, label="Lower Bound", color='orange', lines
plt.axvline(x=optimal_mu2, color='green', linestyle='-', label=f"Max Lower Bound
plt.scatter(optimal_mu2, max_lower_bound, color='green')  # Highlight the point

# Labels and Title
plt.title("Log-Likelihood and Lower Bound as a Function of $\\mu_2$")
plt.xlabel("$\\mu_2$")
plt.ylabel("Value")
plt.legend()
plt.grid(True)
plt.show()

# Print the optimal mu2
print(f"The value of mu2 that maximizes the lower bound is: {optimal_mu2:.4f}")
print(f"The maximum lower bound value is: {max_lower_bound:.4f}")
```




```
The value of mu2 that maximizes the lower bound is: 197.1571
The maximum lower bound value is: -84083.9771
```

The last task gave us the best value for μ_2 that maximizes the lower bound as aproximately 197,16 and the maximum lower bound value at this point is about -84083,98. This shows that optimizing the lower bound is a reliable way to find good parameter values for the model. The graph clearly demonstrates how the lower bound tracks the log-likelihood. This means that by maximizing the lower bound, we can efficiently improve the model's performance without directly calculating the more complex log-likelihood.

Conclusion

In this assignment, we learned about the Expectation-Maximization (EM) algorithm for fitting model parameters to MRI data.
First, the MRI image was displayed and a histogram of pixel intensities was created. Peaks

in the histogram highlight dominant intensity regions corresponding to different tissue types.

Then a 3-component Gaussian mixture model (GMM) was initialized by dividing the intensity range into three intervals. Each Gaussian's mean, variance, and weight were set based on these intervals. The initial GMM was plotted over the histogram, showing each Gaussian weighted by its prior probability π_k and the overall mixture model. Probabilities $w_{n,k}$ were computed for each pixel's class membership. The visualization showed how the pixels were segmented into three classes, highlighting regions corresponding to different Gaussian components.

Then the EM algorithm iteratively updated the model's parameters to improve the model. The evolution of the log-likelihood was plotted, confirming convergence, and the final segmentation results were visualized. By varying only μ_2, the log-likelihood function was plotted. This showed how sensitive the model's fit was to changes in this parameter while others were fixed.

Finally a lower bound of the log-likelihood was computed and plotted alongside the log-likelihood. The lower bound tracked the log-likelihood closely but always remained slightly below it, as expected. The value of μ_2 that maximized the lower bound was identified as 197.16 and its location was marked on the graph. This demonstrated the lower bound's role in guiding parameter optimization.