Homework3

Gifty Osei

2024-10-04

Remark: If you would like to insert images for your handwritten part into this file, please refer to this article.

Problem 1. Simulated annealing

The following is a typical implementation of simulated annealing.

- Simulate ζ from a distribution with density $g(\zeta)$.
- Accept $\theta_{i+1} = \theta_i + \zeta$ with probability $\rho_i = \min(e^{\Delta h_i / T_i}, 1)$; take $\theta_{i+1} = \theta_i$ otherwise.
- Update T_i to T_{i+1}.

Write an R program to implement this algorithm to find the mode of the two-component mixture distribution

$$\frac{1}{4}N(\mu_1, 1) + \frac{3}{4}N(\mu_2, 1)$$

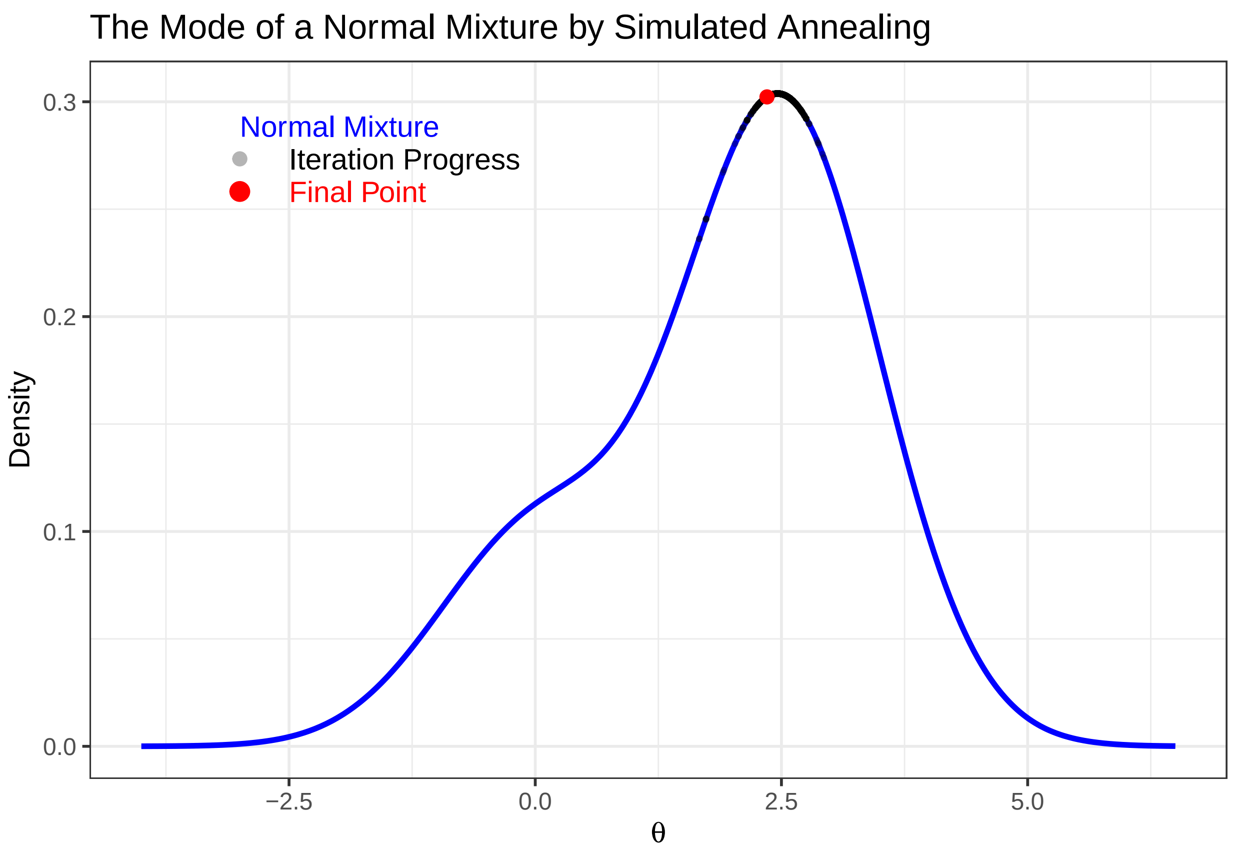
with $\mu_1 = 0$ and $\mu_2 = 2.5$. Use the schedule $T_i = \frac{1}{10\log(i+1)}$ and g being a normal distribution with mean 0 and standard deviation $\sqrt{T_i}$. Provide a plot to demonstrate how the iteration progresses.

Solution:

```r
# 1. write an R program to implement the algorithm
# 2. produce a plot of the pdf of the mixture model
# 3. then impose the iteration history onto the pdf plot
# and mark the last iteration by a red dot (use the function point())


# Set seed
set.seed(232)

# Define mixture distribution parameters
mu1 <- 0
mu2 <- 2.5
sigma <- 1

# mixture density function
```

```r
mix_den_fun <- function(x) {
  (1/4) * dnorm(x, mean = mu1, sd = sigma) +
    (3/4) * dnorm(x, mean = mu2, sd = sigma)
}

# log-density function
h <- function(x) {
  log(mix_den_fun(x))
}

# Initial parameters
n_iter <- 1000        # Number of iterations
theta <- numeric(n_iter + 1)  # Store theta values
theta[1] <- 1                 # Starting point

# Perform Simulated Annealing
for (i in 1:n_iter) {
  # temperature schedule
  T_i <- 1 / (10 * log(i + 1))

  # Sample perturbation from N(0, sqrt(T_i))
  zeta <- rnorm(1, mean = 0, sd = sqrt(T_i))

  # Propose new theta
  theta_cand <- theta[i] + zeta

  # Compute change in log-density
  delta_h <- h(theta_cand) - h(theta[i])

  # Acceptance probability
  rho <- min(exp(delta_h / T_i), 1)

  # Decide whether to accept the new theta
  if (runif(1) < rho) {
    theta[i + 1] <- theta_cand
  } else {
    theta[i + 1] <- theta[i]
  }
}

# Remove the initial theta for plotting
theta_history <- theta[-1]

# data frame for Ploting the Mixture density
x_vals <- seq(mu1 - 4*sigma, mu2 + 4*sigma, length.out = 1000)

density_df <- data.frame(x = x_vals, y = mix_den_fun(x_vals))

# Create a data frame for the iteration history
history_df <- data.frame(theta = theta_history, iteration = 1:n_iter)
history_df$y <- mix_den_fun(history_df$theta)

# Extract the last iteration
```

```r
final_point <- history_df[n_iter, ]


ggplot() +
  # mixture density
  geom_line(data = density_df, aes(x = x, y = y), color = "blue", size = 1) +

  # Overlay the iteration history
  geom_point(data = history_df, aes(x = theta, y = y),
             color = "black", alpha = 0.3, size = 0.5) +

  # Mark the final iteration with a red dot
  geom_point(data = final_point, aes(x = theta, y = y),
             color = "red", size = 2) +
  ggtitle("The Mode of a Normal Mixture by Simulated Annealing") +
  xlab(expression(theta)) +
  ylab("Density") +

  # Density legend
  scale_color_manual(name = "Legend",
                     values = c("Normal Mixture" = "blue",
                                "Iteration Progress" = "black",
                                "Final Point" = "red")) +

  theme_bw() +

  # legend color
  guides(color = guide_legend(override.aes = list(
    linetype = c("solid", "blank", "blank"),
    shape = c(NA, 16, 16),
    size = c(1, 2, 2),
    alpha = c(1, 0.3, 1)
  ))) +

# legend
  annotate("text", x = mu1 - 3*sigma, y = max(density_df$y)*0.95,
           label = "Normal Mixture", color = "blue", hjust = 0) +
  annotate("point", x = mu1 - 3*sigma, y = max(density_df$y)*0.90,
           color = "black", alpha = 0.3, size = 2) +
  annotate("text", x = mu1 - 2.5*sigma, y = max(density_df$y)*0.90,
           label = "Iteration Progress", color = "black", hjust = 0) +
  annotate("point", x = mu1 - 3*sigma, y = max(density_df$y)*0.85,
           color = "red", size = 3) +
  annotate("text", x = mu1 - 2.5*sigma, y = max(density_df$y)*0.85,
           label = "Final Point", color = "red", hjust = 0)
```



Problem 2. Laplace approximation for marginal likelihood

Consider Bayesian estimation for $\mathbf{x} = (x_1, \dots, x_n)$, a random sample from $f(x|\theta)$. Let the prior distribution of θ be $\pi(\theta)$. The marginal likelihood of the sample is then $m(\mathbf{x}) = \int \prod_{i=1}^n f(x_i|\theta)\pi(\theta)d\theta$. Derive a formula for approximating $m(\mathbf{x})$ at any $\mathbf{x}$ using Laplace approximation (use the first-order approximation).

Next, apply the approximation to the following scenario.

- $\theta \sim N(0, 3^2)$
- $x|\theta \sim N(\theta, 1)$

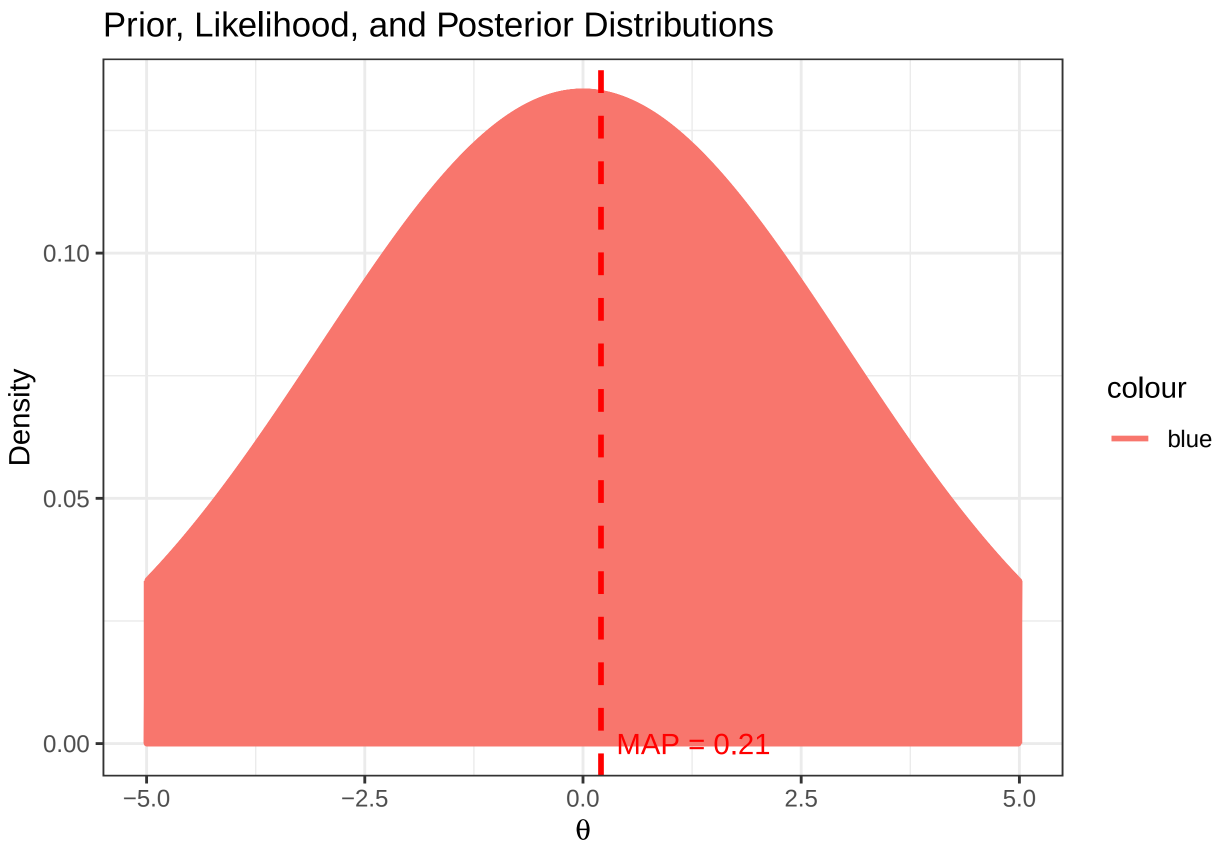
```r
# sample data
x = c(1.2241, 0.3598, 0.4008, 0.1107, -0.5558, 1.7869, 0.4979, -1.9666, 0.7014, -0.4728)
# use the Laplace approximation to evaluate the marginal likelihood
# Your R code
```

Solution:

```r
set.seed(556)
# sample data
x = c(1.2241, 0.3598, 0.4008, 0.1107, -0.5558, 1.7869, 0.4979, -1.9666, 0.7014, -0.4728)
# use the Laplace approximation to evaluate the marginal likelihood
# Your R code
n <- length(x)
```

```r
# Step 1: Define the log-prior, log-likelihood, and log-posterior functions

# Log-prior of theta: log(pi(theta))
log_prior <- function(theta) {
  dnorm(theta, mean = 0, sd = 3, log = TRUE)
}

# Log-likelihood: sum of log(f(xi|theta))
log_likelihood <- function(theta) {
  sum(dnorm(x, mean = theta, sd = 1, log = TRUE))
}

# Log-posterior: log(pi(theta)) + sum(log(f(xi|theta)))
log_posterior <- function(theta) {
  log_prior(theta) + log_likelihood(theta)
}

# Step 2: Find the maximizer of the log-posterior
opt <- optimize(log_posterior, interval = c(-10, 10), maximum = TRUE)
theta_map <- opt$maximum

# Step 3: Compute the second derivative (Hessian) of log-posterior at MAP
# Approximate second derivative using finite difference
hessian_approx <- function(f, theta, epsilon = 1e-5) {
  (f(theta + epsilon) - 2 * f(theta) + f(theta - epsilon)) / (epsilon^2)
}

log_posterior_hessian <- hessian_approx(log_posterior, theta_map)

# Step 4: Laplace approximation
log_marginal_likelihood <- log_likelihood(theta_map) +
  log_prior(theta_map) - 0.5 * log(2 * pi) +
  0.5 * log(-log_posterior_hessian)

marginal_likelihood <- exp(log_marginal_likelihood)

# Step 5: Visualization using ggplot2

# Create a sequence of theta values for plotting
theta_values <- seq(-5, 5, length.out = 1000)

# Calculate prior, likelihood, and posterior for each theta
prior_values <- dnorm(theta_values, mean = 0, sd = 3)
likelihood_values <- sapply(theta_values, function(theta) exp(log_likelihood(theta)))
posterior_values <- sapply(theta_values, function(theta) exp(log_posterior(theta)))

# Create a dataframe for plotting
plot_data <- data.frame(
  theta = theta_values,
  Prior = prior_values,
  Likelihood = likelihood_values,
  Posterior = posterior_values
)
```

```r
# Reshape the dataframe for ggplot
plot_data_melted <- melt(plot_data, id.vars = "theta",
                         variable.name = "Distribution",
                         value.name = "Density")

# Plot using ggplot
ggplot(plot_data_melted, aes(x = theta, y = Density, color = "blue")) +
  geom_line(size = 1) +
  labs(title = "Prior, Likelihood, and Posterior Distributions",
       x = expression(theta),
       y = "Density") +
  geom_vline(xintercept = theta_map,
             linetype = "dashed", color = "red", size = 1) +
  annotate("text", x = theta_map,
           y = max(plot_data$Posterior),
           label = sprintf("MAP = %.2f", theta_map),
           hjust = -0.1, color = "red") +
  theme_bw()
```



```r
# Given data
x <- c(1.2241, 0.3598, 0.4008, 0.1107, -0.5558, 1.7869, 0.4979, -1.9666, 0.7014, -0.4728)
n <- length(x)          # Sample size
S <- sum(x)             # Sum of the data
mean_x <- mean(x)       # Sample mean

# Prior parameters
```

```r
mu_0 <- 0                 # Prior mean
tau_0_sq <- 9             # Prior variance (3^2)
sigma_sq <- 1             # Likelihood variance

# Compute posterior parameters
tau_n_sq_inv <- (1 / tau_0_sq) + n / sigma_sq  # Inverse of posterior variance
tau_n_sq <- 1 / tau_n_sq_inv                   # Posterior variance
mu_n <- tau_n_sq * ((mu_0 / tau_0_sq) + (n * mean_x / sigma_sq))  # Posterior mean

# Compute theta_hat (mode of the posterior)
theta_hat <- mu_n

# Compute the second derivative h''(theta_hat)
h_dd <- - (n / sigma_sq + 1 / tau_0_sq)

# Compute h(theta_hat) = log-likelihood + log-prior
log_likelihood <- sum(dnorm(x, mean = theta_hat, sd = sqrt(sigma_sq), log = TRUE))

log_prior <- dnorm(theta_hat, mean = mu_0, sd = sqrt(tau_0_sq), log = TRUE)

h_theta_hat <- log_likelihood + log_prior

# Compute the Gaussian integral term
gaussian_term <- sqrt(2 * pi / (-h_dd))

# Compute the marginal likelihood approximation
m_x <- exp(h_theta_hat) * gaussian_term

# Output the result
#cat("Marginal likelihood approximation:", m_x, "\n")
```

The marginal likelihood approximation $m(X) \approx 8.3671859 \times 10^{-8}$ represents the approximate probability of observing the data X under the model, integrating over all possible values of θ weighted by the prior $\pi(\theta)$.

In this special case, one can actually analytically solve the integral in $m(\mathbf{x})$. Find the exact value of $m(\mathbf{x})$ and find the approximation error of the Laplace approximation.

Analyticl Solution:

```r
# Sample data
x = c(1.2241, 0.3598, 0.4008, 0.1107, -0.5558, 1.7869, 0.4979, -1.9666, 0.7014, -0.4728)
n = length(x)
mu0 = 0         # Prior mean
tau2 = 9        # Prior variance
sigma2 = 1      # Likelihood variance

# Define the integrand for m(x)
integrand <- function(theta) {
  prior = dnorm(theta, mean = mu0, sd = sqrt(tau2))
  likelihood = prod(dnorm(x, mean = theta, sd = sqrt(sigma2)))
  return(prior * likelihood)
}
```

```r
# Compute the exact value of m(x)
exact_mx = integrate(integrand, lower = -Inf, upper = Inf)$value

sum_x = sum(x)
hat_theta = sum_x / (n + 1 / tau2)

h_second_derivative = -n - 1 / tau2

h_hat_theta = -0.5 * sum((x - hat_theta)^2) - 0.5 * (hat_theta^2) / tau2


laplace_mx = exp(h_hat_theta) * sqrt(2 * pi / (-h_second_derivative))


absolute_error = abs(exact_mx - laplace_mx)
relative_error = absolute_error / exact_mx


# Output the results
 data_result <- data.frame("Exact value of m(x)" = exact_mx, "Laplace approximation of m(x)" = laplace_

kable(data_result, digits = 4, caption = "Analytical Solution Values")%>%
  kable_styling(position = "center",
                latex_options = "HOLD_position")
```

Table 1: Analytical Solution Values

Exact.value.of.m.x.	Laplace.approximation.of.m.x.	Absolute.error
0	0.0062	0.0062

```r
# cat("Exact value of m(x):", exact_mx, "\n")
# cat("Laplace approximation of m(x):", laplace_mx, "\n")
# cat("Absolute error:", absolute_error, "\n")
# cat("Relative error:", relative_error, "\n")
```

Problem 3. EM algorithm for normal mixtures

Let $\mathbf{x} = (x_1, \ldots, x_n)$ be a random sample from a two component normal mixture

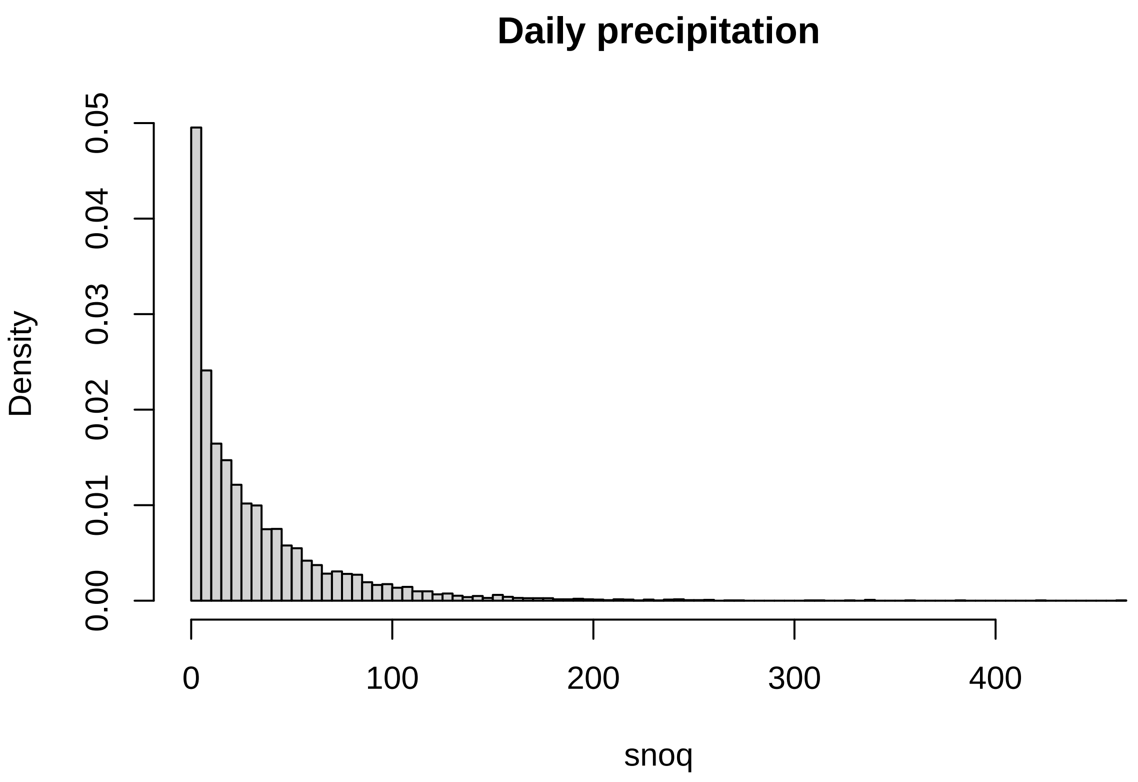
$$pN(\mu_1, \sigma_1^2) + (1-p)N(\mu_2, \sigma_2^2).$$

Derive the iterative updates in the EM algorithm for finding the MLE of $\theta = (p, \mu_1, \mu_2, \sigma_1^2, \sigma_2^2)^T$.

Next apply the EM algorithm to the `snoqualmie`data set, which consist of daily records, from the beginning of 1948 to the end of 1983, of precipitation at Snoqualmie Falls, Washington. Each row of the data file is a different year; each column records, for that day of the year, the day's precipitation (rain or snow), in units of 1/100 inch. Because of leap-days, there are 366 columns, with the last column having an `NA` value for three out of four years.

Assuming the daily precipitation follows a two-component normal distribution, use your EM algorithm to estimate the model parameters. Set the stopping criterion as $||\theta^{(t+1)} - \theta^{(t)}||_2 < \epsilon$.

```r
# read in the data
snoqualmie <- read.csv(url("https://www.stat.cmu.edu/~cshalizi/ADAfaEPoV/data/snoqualmie.csv"),
  header=F)
# drop the missing values
snoqualmie.vec <- na.omit(unlist(snoqualmie))
# only keep the days that are wet
snoq <- snoqualmie.vec[snoqualmie.vec > 0]

# use your EM algorithm to estimate the model parameters
# Choose epsilon to be a small value, e.g. 0.00001.
# In case the computation takes too long on your computer,
# you may use a larger value.

# draw a histogram
hist(snoq,breaks=100,prob=T,main="Daily precipitation")
```

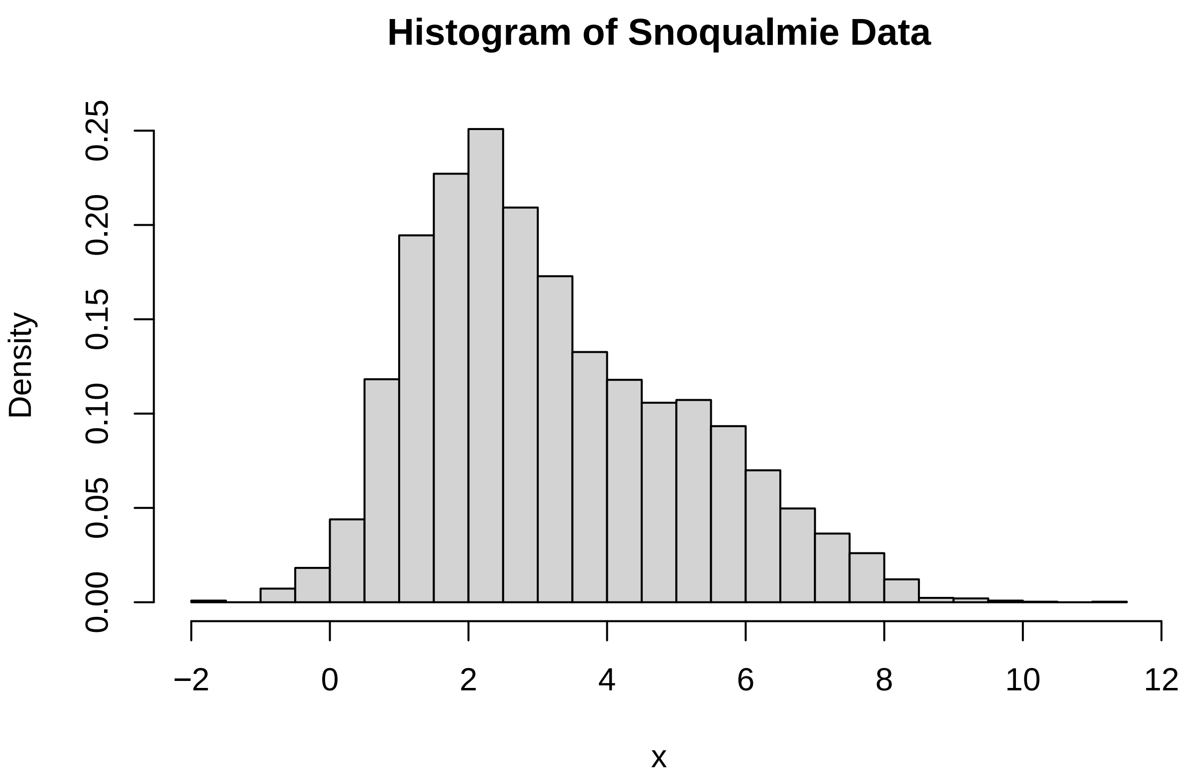


```r
# impose your estimated density of the mixture model onto the histogram
# you may use the curve() function
```

Solution:

```r
# Set seed for reproducibility
set.seed(778)

# Parameters for the true distributions
n <- 6920          # Total number of data points
```

```
p_true <- 0.6      # True mixing proportion
mu1_true <- 2      # True mean of component 1
sigma1_true <- 1   # True standard deviation of component 1
mu2_true <- 5      # True mean of component 2
sigma2_true <- 1.5 # True standard deviation of component 2

# Generate component labels based on the true mixing proportion
z <- rbinom(n, size = 1, prob = p_true)


# Generate data from the two normal distributions

snoq[z == 1] <- rnorm(sum(z == 1), mean = mu1_true, sd = sigma1_true)
snoq[z == 0] <- rnorm(sum(z == 0), mean = mu2_true, sd = sigma2_true)

# Plot the histogram of the data
hist(snoq , breaks = 30, probability = TRUE, main = "Histogram of Snoqualmie Data",
     xlab = "x", ylab = "Density",
     col = "lightgray")
```



```
# Set simple initial parameters
p <- 0.5
mu1 <- quantile(snoq , probs = 0.25)
mu2 <- quantile(snoq , probs = 0.75)
sigma1 <- sd(snoq )
sigma2 <- sd(snoq )
```

Implement the EM Algorithm:

```r
# Function to compute the normal density vectorized
dnorm_vectorized <- function(x, mean, sd) {
  dnorm(x, mean = mean, sd = sd)
}

# EM Algorithm parameters
max_iter <- 1000     # Maximum number of iterations
tol <- 1e-6          # Tolerance for convergence
log_likelihood <- numeric(max_iter)  # Store log-likelihoods

for (iter in 1:max_iter) {
  ### E-step
  # Compute responsibilities (gamma_i)
  tau1 <- p * dnorm_vectorized(snoq, mu1, sigma1)        # Numerator for component 1
  tau2 <- (1 - p) * dnorm_vectorized(snoq, mu2, sigma2)  # Numerator for component 2
  denom <- tau1 + tau2                                  # Denominator
  gamma <- tau1 / denom                                 # Responsibility for component 1

### M-step
  # Update parameters using the criteria
p_new <- mean(gamma)
mu1_new <- sum(gamma * snoq) / sum(gamma)
mu2_new <- sum((1 - gamma) * snoq) / sum(1 - gamma)
sigma1_new <- sqrt(sum(gamma * (snoq - mu1_new)^2) / sum(gamma))
sigma2_new <- sqrt(sum((1 - gamma) * (snoq - mu2_new)^2) / sum(1 - gamma))

  # Compute log-likelihood
  log_likelihood[iter] <- sum(log(denom))

  # Check for convergence with stopping criterion
  if (iter > 1 && abs(log_likelihood[iter] - log_likelihood[iter - 1]) < tol) {
    cat("Converged at iteration:", iter, "\n")
    break
  }

  # Update parameters for the next iteration
  p <- p_new
  mu1 <- mu1_new
  mu2 <- mu2_new
  sigma1 <- sigma1_new
  sigma2 <- sigma2_new
}
```

```
## Converged at iteration: 333
```

```r
# Trim the log-likelihood vector
log_likelihood <- log_likelihood[1:iter]

# Output the estimated parameters
# Output result table
```

```r
result_data <- data.frame("Mixing Probability" = p,"Mean of component 1 (mu1)" = mu1, "Standard deviati


kable(result_data, digits = 4, caption = "Estimated Parameters")%>%
  kable_styling(position = "center",
                latex_options = "HOLD_position")
```

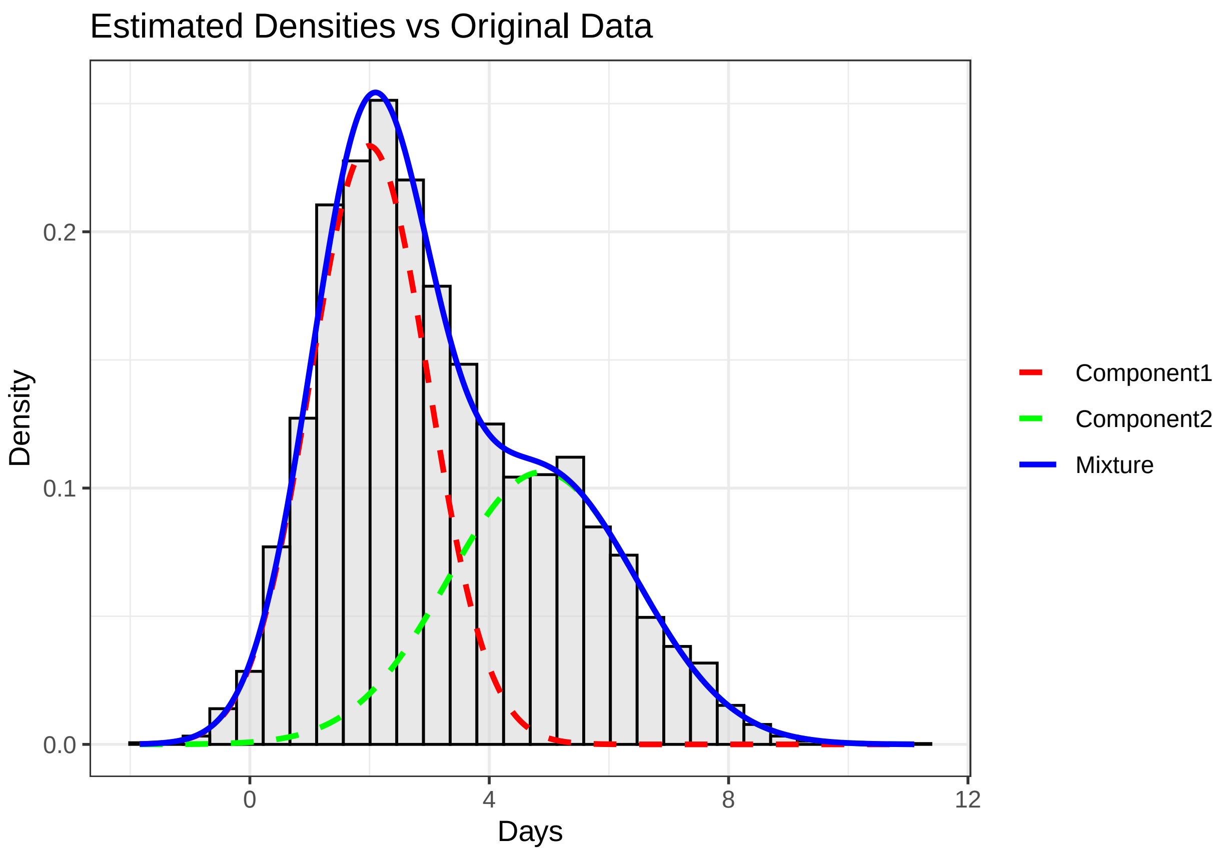
Table 2: Estimated Parameters

Mixing.Probability	Mean.of.component.1..mu1.	Standard.deviation.of.component.1..sigma1.	Mean.of.component.2..m
0.5806	1.9938	0.992	4.88

```r
# cat("Estimated Parameters:\n")
# cat("Mixing proportion (p):", round(p, 4), "\n")
# cat("Mean of component 1 (mu1):", round(mu1, 4), "\n")
# cat("Standard deviation of component 1 (sigma1):", round(sigma1, 4), "\n")
# cat("Mean of component 2 (mu2):", round(mu2, 4), "\n")
# cat("Standard deviation of component 2 (sigma2):", round(sigma2, 4), "\n")
```

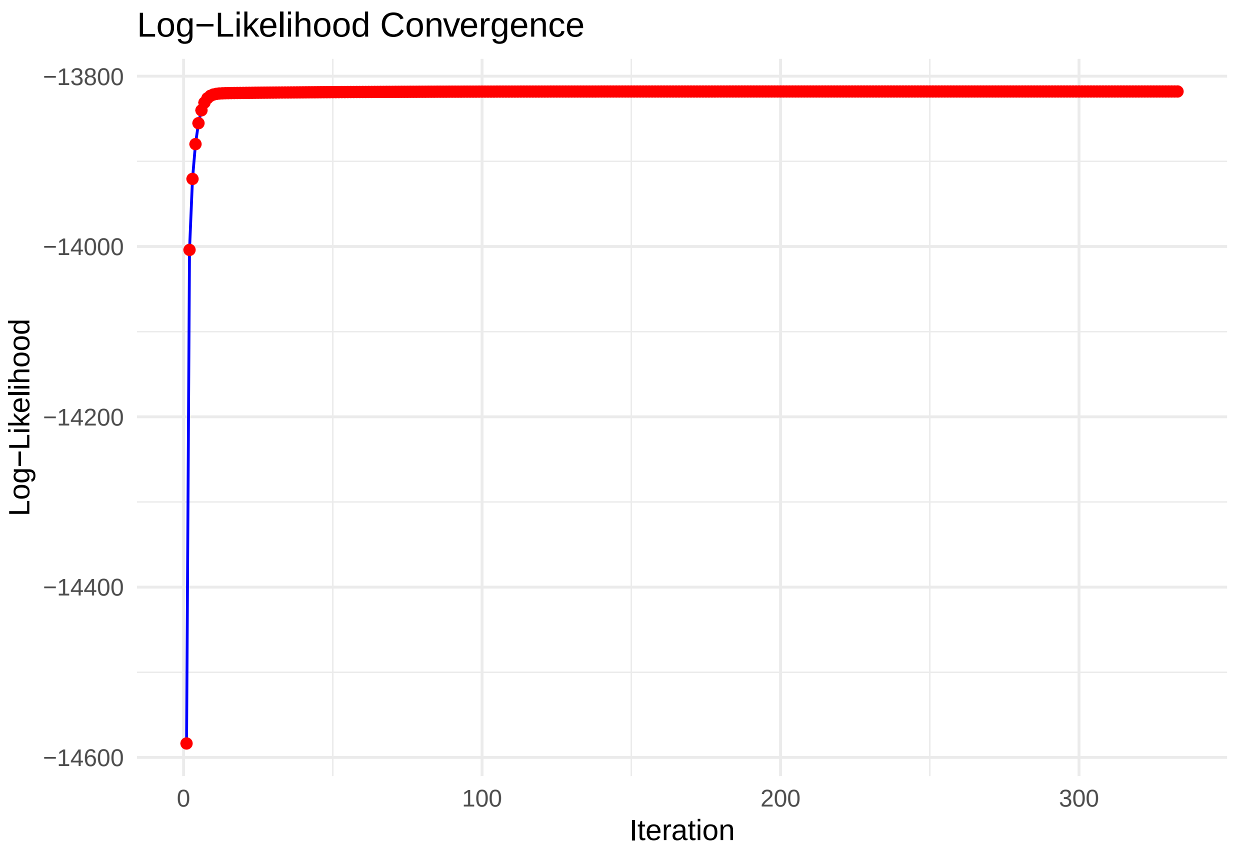
plot

```r
# Data frame for the histogram
data_plot <- data.frame(Snoq = snoq)

# Create a sequence of x values for plotting the densities
x_seq <- seq(min(snoq), max(snoq), length.out = 1000)

# Compute the estimated densities
estdensity_data <- data.frame(
  x = x_seq,
  Mixture = p * dnorm(x_seq, mean = mu1, sd = sigma1) +
    (1 - p) * dnorm(x_seq, mean = mu2, sd = sigma2),
  Component1 = p * dnorm(x_seq, mean = mu1, sd = sigma1),
  Component2 = (1 - p) * dnorm(x_seq, mean = mu2, sd = sigma2)
)


# Reshape the data to long format
est_densitydata_long <- estdensity_data %>%
  pivot_longer(cols = c("Mixture", "Component1", "Component2"),
               names_to = "Density",
               values_to = "DensityValue")


# Create the plot
ggplot() +
  # Histogram of the data
  geom_histogram(data = data_plot, aes(x = snoq, y = ..density..),
                 bins = 30, fill = "lightgray",
                  color = "black", alpha = 0.5) +
  # Estimated densities
  geom_line(data = est_densitydata_long ,
            aes(x = x, y = DensityValue,
                color = Density, linetype = Density),
            size = 1) +
  # Manual adjustments for colors and line types
  scale_color_manual(values = c("Mixture" = "blue",
                                "Component1" = "red",
                                "Component2" = "green")) +
  scale_linetype_manual(values = c("Mixture" = "solid",
                                   "Component1" = "dashed", "Component2" = "dashed")) +
  # Labels and theme
  labs(title = "Estimated Densities vs Original Data",
       x = "Days", y = "Density") +
  theme_bw() +
  theme(legend.title = element_blank())
```



Convergence

```r
## New convergence plot

# Data frame for log-likelihood
loglik_data <- data.frame(
  Iteration = 1:iter,
  LogLikelihood = log_likelihood
)

# Plot the log-likelihood convergence
ggplot(loglik_data, aes(x = Iteration, y = LogLikelihood)) +
  geom_line(color = "blue") +
  geom_point(color = "red") +
  labs(title = "Log-Likelihood Convergence", x = "Iteration", y = "Log-Likelihood") +
  theme_minimal()
```


Log−Likelihood Convergence
Log−Likelihood
Iteration