Bayesian Statistical Modeling: Markov Chain Monte Carlo

Eason Cai

2025-02-14

1. Introduction

In this report, we are revisiting the safety concern an emergency landing system. In an emergency, the airplane system must quickly estimate the usable fuel remaining based on sensor readings and known consumption rates. You can find detailed information on Bayesian Monte Carlo and Grid Methods through https://github.com/easoncai999/bayesian-ps3.

Recall the given information and assumptions:

- **Fuel tank capacity:** 182 liters.
- **Fuel sensor reading:** 34 liters.
- **Fuel sensor error:** $\mathcal{N}(0, 20^2)$.
- **Fuel consumption rate:** $\mathcal{N}(18, 2^2)$.
- The fuel level and consumption are assumed **uncorrelated**.

In this report, we will do the following analysis:

1. Implement the MH algorithm to sample from the posterior:

$$\mathbb{P}(F \mid \text{data}) \propto \mathcal{N}(34, 20) \times \mathbb{1}_{\{0,182\}}(F).$$

2. Assess whether the numerical inference of the mode converges.
3. Define and use a positive control (a simple normal distribution with a known mode) to check our method's accuracy.
4. Compare the required number of runs for convergence with that of the Bayes Monte Carlo method.

---

2. Metropolis Hastings Algorithm Implementation

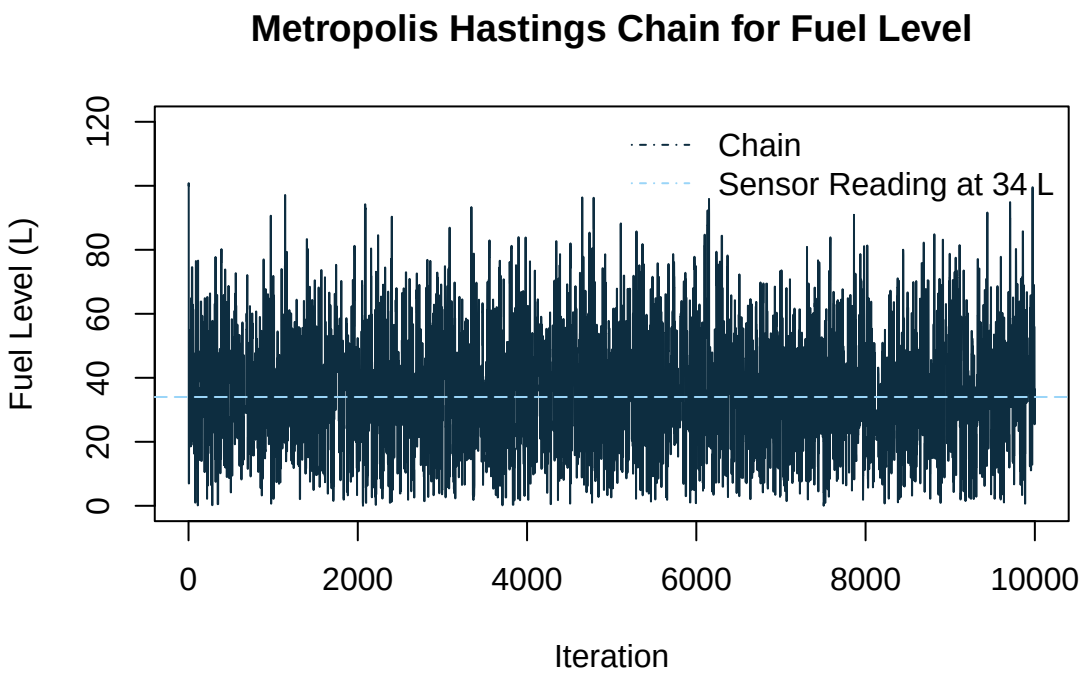
```
set.seed(107)

# Log posterior function that returns the log density for a given fuel level F
log_posterior <- function(F) {
  if (F <= 0 || F >= 182) {
    return(-Inf)
  }
  return(dnorm(F, mean = 34, sd = 20, log = TRUE))
```

```
}

# Setup initial states
n <- 10000
F_current <- 100
chain <- numeric(n)
chain[1] <- F_current
current_logp <- log_posterior(F_current)

# Tuning parameter for Metropolis-Hastings
proposal_sd <- 50

# Run the Metropolis-Hastings Algorithm
accepts <- 0
for (i in 2:n) {
  F_proposed <- rnorm(1, mean = F_current, sd = proposal_sd)
  proposed_logp <- log_posterior(F_proposed)
  log_acceptance <- proposed_logp - current_logp

  # Accept the proposed state
  if (log(runif(1)) < log_acceptance) {
    F_current <- F_proposed
    current_logp <- proposed_logp
    accepts <- accepts + 1
  }
  chain[i] <- F_current
}

# Find the acceptance rate
accept_rate <- (accepts / n) * 100
print(paste("Acceptance Rate of the Metropolis Hastings Algorithm:",
            round(accept_rate, 2), "%"))
```

```
## [1] "Acceptance Rate of the Metropolis Hastings Algorithm: 40.54 %"
```

After tuning the proposed standard deviation, approximately 40.54% acceptance rate is good point to start.

```
plot(chain, type = "l", col = "#0D2D40", xlab = "Iteration",
     ylab = "Fuel Level (L)", main = "Metropolis Hastings Chain for Fuel Level",
     ylim = c(0, 120))
abline(h = 34, col = "#9AD5F8", lty = 5)
legend("topright", legend = c("Chain", "Sensor Reading at 34 L"),
       col = c("#0D2D40", "#9AD5F8"), lty = c(4, 4), bty = "n")
```



---

3. Mode Convergence by Numerical Inference

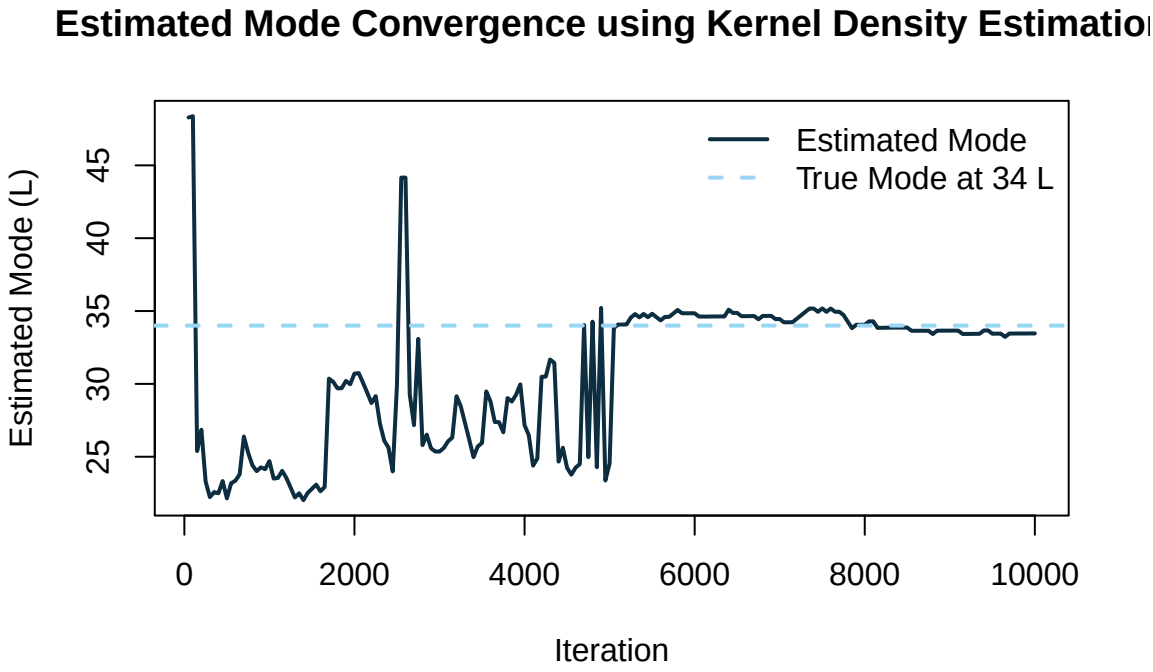
We now want to find the fuel level with the highest log posterior value observed so far and then assess whether this mode converges or not.

In our scenario, we know the unnormalized posterior is given by a normal density with $\mu = 34$, then the true mode is 34. Therefore, if the analysis shows it is converging, then the running estimate should be approximately 34.

We will use kernel density estimation which means for a sample, we first estimates the underlying density. Then identify the x-value where the estimated density is maximized which gives the current mode estimate.

```
KDE <- function(iteration, MCMC_sample){
  modes <- numeric(length(iteration))

  # For each iteration step, find the estimate mode using KDE
  for (i in seq_along(iteration)) {
    index_i <- iteration[i]
    dens <- density(MCMC_sample[1:index_i])
    modes[i] <- dens$x[which.max(dens$y)]
  }
  return(modes)
}

iteration <- seq(50, length(chain), by = 50)
modes <- KDE(iteration, chain)

plot(iteration, modes, type = "l", col = "#0D2D40", lwd = 2,
     xlab = "Iteration", ylab = "Estimated Mode (L)",
```

```
     main = "Estimated Mode Convergence using Kernel Density Estimation")
abline(h = 34, col = "#9AD5F8", lty = 2, lwd = 2)
legend("topright", legend = c("Estimated Mode", "True Mode at 34 L"),
       col = c("#0D2D40", "#9AD5F8"), lty = c(1,2), lwd = c(2,2), bty = "n")
```



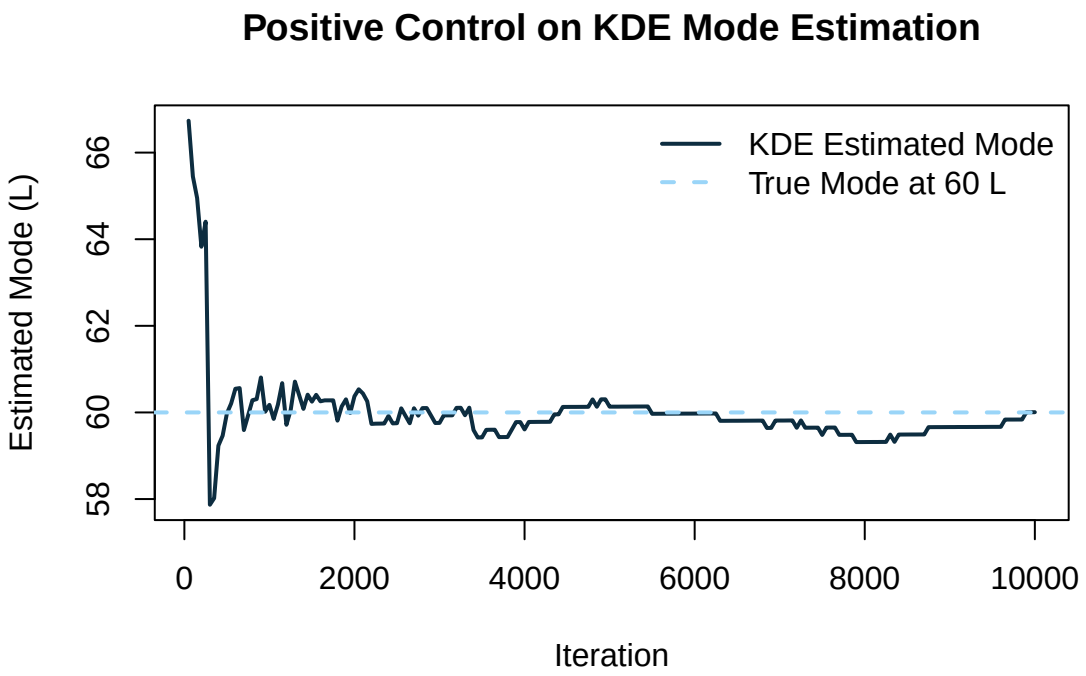
Therefore, we can observe that after approximately 5000 iterations, the estimated mode converges to the true mode.

---

4. Positive Control

```
# Positive control parameters
mu <- 60
std <- 10
lower_bound <- 0
upper_bound <- 182

# Generate samples from Normal(30, 10) and reject those outside (0,182)
pc_sample <- numeric(0)
while(length(pc_sample) < n) {
  temp <- rnorm(10000, mean = mu, sd = std)
  valid <- temp[temp > lower_bound & temp < upper_bound]
  pc_sample <- c(pc_sample, valid)
}

pc_sample <- pc_sample[1:n] # we want exactly n samples

# Using KDE function to find all estimated modes
```

```r
pc_modes <- KDE(iteration, pc_sample)

plot(iteration, pc_modes, type = "l", col = "#0D2D40", lwd = 2,
     xlab = "Iteration", ylab = "Estimated Mode (L)",
     main = "Positive Control on KDE Mode Estimation")
abline(h = mu, col = "#9AD5F8", lty = 2, lwd = 2)
legend("topright", legend = c("KDE Estimated Mode", "True Mode at 60 L"),
       col = c("#0D2D40", "#9AD5F8"), lty = c(1, 2), lwd = c(2, 2), bty = "n")
```




```r
# Print the final estimated mode and the absolute error from the true value.
final_estimate <- pc_modes[length(pc_modes)]
error <- abs(final_estimate - mu)
print(paste("Final KDE mode estimate:", round(final_estimate, 2)))
```

```
## [1] "Final KDE mode estimate: 60.01"
```

```r
print(paste("Absolute error:", round(error, 2)))
```

```
## [1] "Absolute error: 0.01"
```

By setting up the target distribution $\mathcal{N}(60, 10^2)$, we can see that KDE mode estimation is able to achieve 60.01 as the final estimation with 0.01 absolute error. Therefore, it passes the positive control test.

---

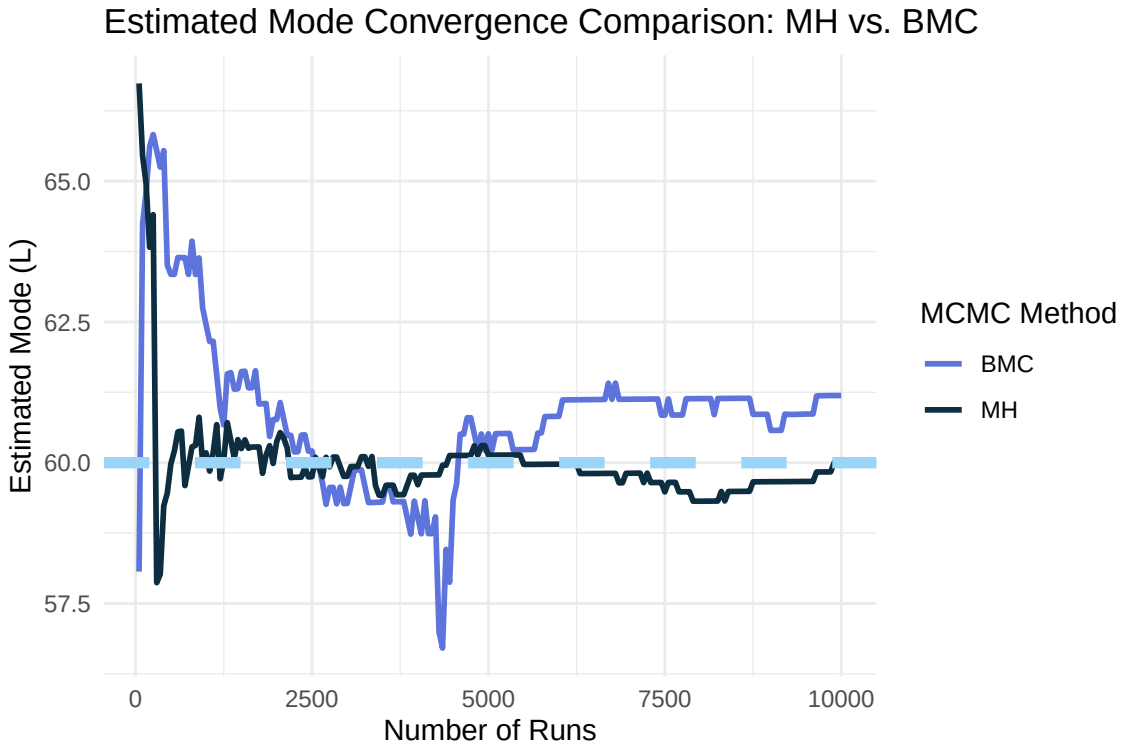
5. Bayes Monte Carlo Algorithm Comparison

```
library(ggplot2)
set.seed(107)

# True mode for the posterior
true_mode <- 60

# Define the BMC Positive Control
prior_sample <- runif(n, min = 0, max = 182)
w <- dnorm(prior_sample, mean = true_mode, sd = 20)
w <- w / sum(w)
bmc_samples <- sample(prior_sample, size = n, replace = TRUE, prob = w)
bmc_modes <- KDE(iteration, bmc_samples)

# Convergence threshold (in liters)
epsilon <- 0.05

convergence <- function(modes, iteration, true_value, epsilon, points = 10) {
  conv_run <- NA
  # Loop only up to the point where 10 consecutive values can be checked
  for (j in 1:(length(modes) - points + 1)) {
    # Check if the next 10 values are all within epsilon
    if (all(abs(modes[j:(j + points - 1)] - true_value) < epsilon)) {
      conv_run <- iteration[j]
      break
    }
    else
      conv_run <- "Did not converge"
  }
  return(conv_run)
}

# Example usage with your variables:
conv_mh  <- convergence(pc_modes, iteration, true_mode, epsilon)
conv_bmc <- convergence(bmc_modes, iteration, true_mode, epsilon)

comparison_table <- data.frame(
  Method = c("Metropolis Hastings", "Bayesian Monte Carlo"),
  Convergence_Runs = c(conv_mh, conv_bmc)
)
kable(comparison_table)
```

Method	Convergence_Runs
Metropolis Hastings	5500
Bayesian Monte Carlo	Did not converge

```
# Now, visualize the comparison in a plot
df_compare <- data.frame(
  iter = rep(iteration, 2),
  mode_est = c(pc_modes, bmc_modes),
```

```
  method = rep(c("MH", "BMC"), each = length(iteration))
)

ggplot(df_compare, aes(x = iter, y = mode_est, color = method)) +
  geom_line(size = 1) +
  geom_hline(yintercept = true_mode, linetype = "dashed", color = "#9AD5F8", size = 2) +
  labs(title = "Estimated Mode Convergence Comparison: MH vs. BMC",
       x = "Number of Runs",
       y = "Estimated Mode (L)",
       color = "MCMC Method") +
  scale_color_manual(values = c("BMC" = "#5E74DD", "MH" = "#0D2D40")) +
  theme_minimal()
```



By defining a function where 10 continuous points are within 0.05 of the true value, we conclude the method converges to the true mode.

From both the plot and the table, we observe that Metropolis Hastings algorithm converges after 5500 runs, whereas Bayesian Monte Carlo did not converge after 10000 runs.

---

6. Summary

1. **Choice of Proposed Standard Deviation**:
    - Standard deviation 50 is chosen after tuning for the acceptance rate, where the plausible choices are any real number.
2. **Choice of Target Distribution Parameter Value in Positive Control**:
    - μ can be any number from 0 to 182 and σ could be any value.
    - $\mu = 60$ and $\sigma = 10$ are chosen in a way that is far from the model we have, thus, provide credibility when looking for results.
3. **Reproducibility**
    - All code and random seeds are included in this document.
    - By setting `seed`, anyone can reproduce the same numeric results.
    - This R Markdown file can be knitted to produce the same figures and estimates.
4. **Citations**
    - Qian, S. S., Stow, C. A., & Borsuk, M. E. (2003). On Monte Carlo methods for Bayesian inference. Ecological Modelling, 159(2-3), 269–277.
    - D'Agostini, G. (2003). Bayesian reasoning in data analysis: A critical introduction. Singapore: World Scientific Publishing. (Chapter 6)
    - Ruckert, K. L., Guan, Y., Bakker, A. M. R., Forest, C. E., & Keller, K. (2017). The effects of time-varying observation errors on semi-empirical sea-level projections. Climatic Change, 140(3-4), 349–360. https://doi.org/10.1007/s10584-016-1858-z
    - R Core Team. (n.d.). Kernel Density Estimation. In R: A Language and Environment for Statistical Computing (R-devel). Vienna, Austria: R Foundation for Statistical Computing. https://stat.ethz.ch/R-manual/R-devel/library/stats/html/density.html
5. **License**
    - GNU general public license v3.0.
6. **Copyright**
    -

Appendix: Code

You can also access the repository using the following URL: https://github.com/easoncai999/bayesian-ps4

```
### PART 2 ###
set.seed(107)

log_posterior <- function(F) {
  if (F <= 0 || F >= 182) {
    return(-Inf)
  }
  return(dnorm(F, mean = 34, sd = 20, log = TRUE))
}

n <- 10000
F_current <- 100
chain <- numeric(n)
chain[1] <- F_current
current_logp <- log_posterior(F_current)
proposal_sd <- 50
accepts <- 0
for (i in 2:n) {
  F_proposed <- rnorm(1, mean = F_current, sd = proposal_sd)
  proposed_logp <- log_posterior(F_proposed)
  log_acceptance <- proposed_logp - current_logp

  if (log(runif(1)) < log_acceptance) {
    F_current <- F_proposed
    current_logp <- proposed_logp
    accepts <- accepts + 1
  }
  chain[i] <- F_current
}

accept_rate <- (accepts / n) * 100
print(paste("Acceptance Rate of the Metropolis Hastings Algorithm:",
            round(accept_rate, 2), "%"))

plot(chain, type = "l", col = "#0D2D40", xlab = "Iteration",
     ylab = "Fuel Level (L)", main = "Metropolis Hastings Chain for Fuel Level",
     ylim = c(0, 120))
abline(h = 34, col = "#9AD5F8", lty = 5)
legend("topright", legend = c("Chain", "Sensor Reading at 34 L"),
       col = c("#0D2D40", "#9AD5F8"), lty = c(4, 4), bty = "n")

### PART 3 ###
KDE <- function(iteration, MCMC_sample){
  modes <- numeric(length(iteration))

  for (i in seq_along(iteration)) {
    index_i <- iteration[i]
    dens <- density(MCMC_sample[1:index_i])
    modes[i] <- dens$x[which.max(dens$y)]
  }
```

```r
  return(modes)
}

iteration <- seq(50, length(chain), by = 50)
modes <- KDE(iteration, chain)

plot(iteration, modes, type = "l", col = "#0D2D40", lwd = 2,
     xlab = "Iteration", ylab = "Estimated Mode (L)",
     main = "Estimated Mode Convergence using Kernel Density Estimation")
abline(h = 34, col = "#9AD5F8", lty = 2, lwd = 2)
legend("topright", legend = c("Estimated Mode", "True Mode at 34 L"),
       col = c("#0D2D40", "#9AD5F8"), lty = c(1,2), lwd = c(2,2), bty = "n")

### PART 4 ###
mu <- 60
std <- 10
lower_bound <- 0
upper_bound <- 182
pc_sample <- numeric(0)
while(length(pc_sample) < n) {
  temp <- rnorm(10000, mean = mu, sd = std)
  valid <- temp[temp > lower_bound & temp < upper_bound]
  pc_sample <- c(pc_sample, valid)
}

pc_sample <- pc_sample[1:n]
pc_modes <- KDE(iteration, pc_sample)

plot(iteration, pc_modes, type = "l", col = "#0D2D40", lwd = 2,
     xlab = "Iteration", ylab = "Estimated Mode (L)",
     main = "Positive Control on KDE Mode Estimation")
abline(h = mu, col = "#9AD5F8", lty = 2, lwd = 2)
legend("topright", legend = c("KDE Estimated Mode", "True Mode at 60 L"),
       col = c("#0D2D40", "#9AD5F8"), lty = c(1, 2), lwd = c(2, 2), bty = "n")

final_estimate <- pc_modes[length(pc_modes)]
error <- abs(final_estimate - mu)
print(paste("Final KDE mode estimate:", round(final_estimate, 2)))
print(paste("Absolute error:", round(error, 2)))

### PART 5 ###
library(ggplot2)
set.seed(107)
true_mode <- 60

prior_sample <- runif(n, min = 0, max = 182)
w <- dnorm(prior_sample, mean = true_mode, sd = 20)
w <- w / sum(w)
bmc_samples <- sample(prior_sample, size = n, replace = TRUE, prob = w)
bmc_modes <- KDE(iteration, bmc_samples)

epsilon <- 0.05
convergence <- function(modes, iteration, true_value, epsilon, points = 10) {
```

```r
  conv_run <- NA
  for (j in 1:(length(modes) - points + 1)) {
    if (all(abs(modes[j:(j + points - 1)] - true_value) < epsilon)) {
      conv_run <- iteration[j]
      break
    }
    else
      conv_run <- "Did not converge"
  }
  return(conv_run)
}

conv_mh  <- convergence(pc_modes, iteration, true_mode, epsilon)
conv_bmc <- convergence(bmc_modes, iteration, true_mode, epsilon)

comparison_table <- data.frame(
  Method = c("Metropolis Hastings", "Bayesian Monte Carlo"),
  Convergence_Runs = c(conv_mh, conv_bmc)
)
kable(comparison_table)

df_compare <- data.frame(
  iter = rep(iteration, 2),
  mode_est = c(pc_modes, bmc_modes),
  method = rep(c("MH", "BMC"), each = length(iteration))
)

ggplot(df_compare, aes(x = iter, y = mode_est, color = method)) +
  geom_line(size = 1) +
  geom_hline(yintercept = true_mode, linetype = "dashed", color = "#9AD5F8", size = 2) +
  labs(title = "Estimated Mode Convergence Comparison: MH vs. BMC",
       x = "Number of Runs",
       y = "Estimated Mode (L)",
       color = "MCMC Method") +
  scale_color_manual(values = c("BMC" = "#5E74DD", "MH" = "#0D2D40")) +
  theme_minimal()
```