MCMC Guide

Andrew Le

Contents

1 Required Libraries

```r
library(bayesplot)
library(LaplacesDemon)
library(coda)
library(xtable)
library(ggplot2)
library(dplyr)
library(tidyr)
```

2 Introduction

The purpose of this guide is to provide a reference for the basics behind implementing MCMC algorithms, checking your model, and summarizing the posterior. Although the examples here are based on one type of model, all of the code and techniques can be generalized to others.

3 General Steps

The general steps for MCMC that you should generally follow is:

1. Write down the prior probability model.
2. Write down the Gibbs sampling algorithm for sampling from the posterior.
3. Simulate data from the model by fixing the parameters to some truth.
4. Implement your MCMC algorithm on the simulated data
5. Summarize results and verify

4 Prior Probability Model

In this guide, we assume data $(y_1, y_2, \ldots, y_n)$ i.i.d. observed from a Normal distribution with unknown parameters (μ, σ^2) defined with a normal inverse-gamma prior.

The prior probability model is given by:

$$\begin{aligned} y_i \mid \mu, \sigma^2 &\sim \mathcal{N}(\mu,\ \sigma^2), \quad i = 1, \ldots, n \\ \mu|\sigma^2 &\sim \mathcal{N}(m, \kappa\sigma^2) \\ \sigma^2 &\sim \text{Inverse-Gamma}(\alpha, \beta) \end{aligned}$$

Any time you specify the prior model, it is important to recognize all of the parameters and hyperparameters so that you understand what is being fit vs tuned in your MCMC algorithm. The parameters in this case are μ and σ^2 and are estimated using MCMC. The hyperparameters in the model are α, β, m, and κ. These are fixed values to control the prior distribution (our beliefs about the parameters).

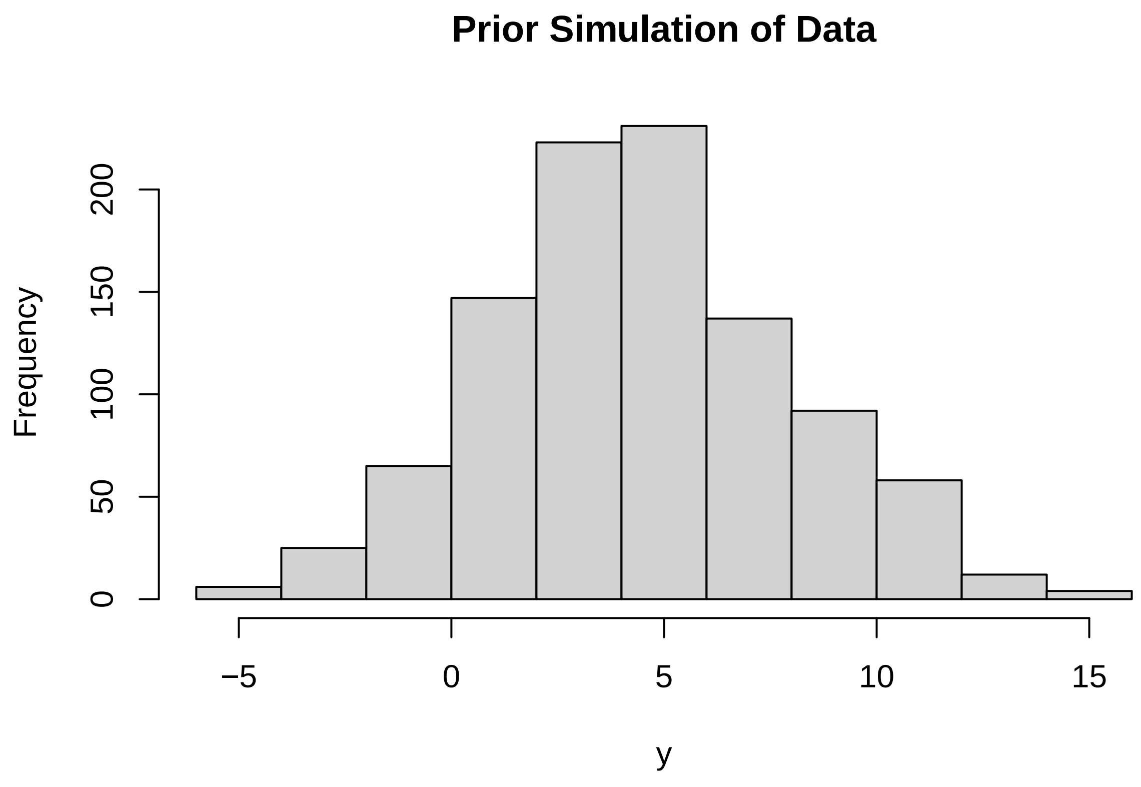
5 Simulating from the Prior Probability Model

This is useful for understanding your data looks like from the prior model. Always start at the very bottom of the prior probability model. In this case, the steps for sampling for the prior from fixed values of the hyperparameters α, β, κ is as follows:

1. Draw N samples from $\sigma^2 \sim \text{IG}(\alpha, \beta)$
2. Draw $\mu \sim \mathcal{N}(m, \kappa\sigma^2)$ (using the sampled σ^2 from earlier)
3. Draw $y \sim \mathcal{N}(\mu, \sigma^2)$ (using the sampled σ^2 and μ form earlier)

```r
simulate_prior <- function(n, alpha, beta, m, kappa) {
  # 1. Draw sigma^2 ~ InverseGamma(alpha, beta)
  sig_sq <- 1 / rgamma(1, shape = alpha, rate = beta)

  # 2. Conditional on sigma^2, draw mu ~ Normal(m, kappa * sigma^2)
  mu <- rnorm(1, mean = m, sd = sqrt(kappa * sig_sq))

  # 3. Given mu and sigma^2, simulate y_i ~ Normal(mu, sigma^2), i=1...n
  y <- rnorm(n, mean = mu, sd = sqrt(sig_sq))

  # Return a list containing the drawn parameters and simulated data
  list(mu = mu, sig_sq = sig_sq, y = y)
}
```

```r
hist(simulate_prior(n = 1000, alpha = 1, beta = 2, m = 3, kappa = 1.5)$y, main = "Prior Simulation of Da
```



6 Gibbs Sampler for Normal Model

We aim to sample from the joint posterior of μ, σ^2 given the data $y_1, \ldots, y_n$, using Gibbs sampling. ALWAYS write derive the posterior full conditionals first and then write down the steps in your sampler before implementing your sampling. Make sure that it is correct (otherwise your implementation will never give you the right results!)

6.1 Algorithm Outline

1. **Initialize** $\mu^{(0)}$ and $\sigma^{2(0)}$.
2. **For** iterations $t = 1, 2, \ldots, T$:
   a. Sample $\mu^{(t)}$ from its full conditional:
   $$\mu^{(t)}|- \sim \mathcal{N}\left(\mu^*, v_\mu^*\right),$$
   where
   $$v_\mu^* = \frac{\sigma^{2(t-1)}}{n + 1/\kappa}, \quad \mu^* = \frac{1}{n + 1/\kappa}\left(n\bar{y} + \frac{1}{\kappa}m\right).$$
   b. Sample $\sigma^{2(t)}$ from its full conditional:
   $$\sigma^{2(t)}|- \sim \text{InverseGamma}\left(\alpha + \frac{n}{2}, \beta + \frac{1}{2}\sum_{i=1}^{n}(y_i - \mu^{(t)})^2\right).$$
3. Repeat until convergence.

This algorithm alternates between sampling μ from its normal full conditional (given σ^2) and σ^2 from its Inverse-Gamma full conditional (given μ).

6.2 Introduction

We implement Gibbs sampler to sample from the joint posterior of μ, σ^2 given the data $y_1, \ldots, y_n$ using the algorithm outlined.

Every MCMC sampler has the following essential ingredients:

- A function for fitting the model with your data, the number of iterations, and the hyper parameters as arguments.
  - This makes it easier to fit the data with different hyperparameters.
- Initializing values for your parameters
  - Use empirical estimates for faster convergence
- A way to store your posterior draws (a vector for any scalar parameters, a matrix for vector parameters)

6.3 Gibbs Sampler Function

```r
fit_model <- function(y, n_iter, alpha, beta, m, kappa) {
  # y:        vector of data (y_1, ..., y_n)
  # n_iter:   number of Gibbs iterations
  # alpha, beta: parameters of InverseGamma for sigma^2
  # m, kappa:    parameters of Normal(m, kappa*sigma^2) for mu

  n <- length(y)
  y_bar <- mean(y)

  # Storage for draws
  mu_samp <- numeric(n_iter)
  sig_sq_samp <- numeric(n_iter)

  # Initialize (use empirical estimates for faster convergence)
  mu_curr <- y_bar
  sig_sq_curr <- var(y)

  for (iter in seq_len(n_iter)) {
    # --- [1] Update mu given sigma^2 ---
    # posterior variance
    v_star <- sig_sq_curr / (n + 1/kappa)
    # posterior mean
    mu_star <- (n * y_bar + (1/kappa) * m) / (n + 1/kappa)

    # draw mu
    mu_curr <- rnorm(1, mean = mu_star, sd = sqrt(v_star))

    # --- [2] Update sigma^2 given mu ---
    shape_post <- alpha + n/2
    # sum of squares around the current mu
    ss <- sum((y - mu_curr)^2)
    rate_post <- beta + ss/2

    # draw sigma^2 from Inverse-Gamma via reciprocal of Gamma
    sig_sq_curr <- 1 / rgamma(1, shape = shape_post, rate = rate_post)

    # Store draws
    mu_samp[iter] <- mu_curr
    sig_sq_samp[iter] <- sig_sq_curr
  }

  # Return posterior draws
  list(mu = mu_samp, sig_sq = sig_sq_samp)
}
```

6.4 Fitting data with Gibbs sampling

```r
# EXAMPLE
set.seed(123)

# Simulate some data
y_data <- rnorm(100, mean = 2, sd = sqrt(1.5))

# Run Gibbs sampler for 2000 iterations:
res <- fit_model(
  y        = y_data,
  n_iter   = 2000,
  alpha    = 2,      # InverseGamma shape
  beta     = 1,      # InverseGamma rate
  m        = 0,      # prior mean for mu
  kappa    = 0.5     # "prior precision" factor for mu
)

# Posterior draws:
mu_post <- res$mu
sig_sq_post <- res$sig_sq
```

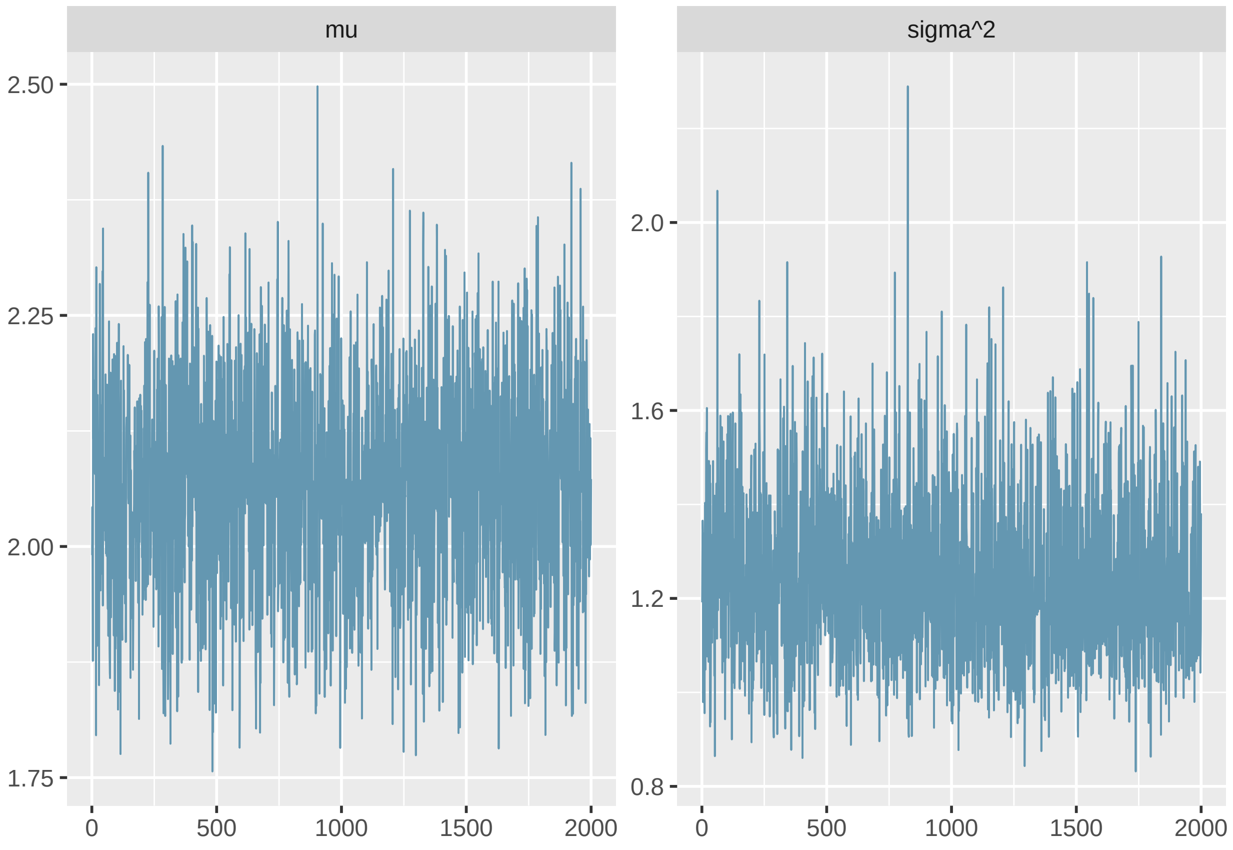
```r
# Combine into a matrix for summarizing (2 columns, mu and sigma2)
posterior_mat <- cbind(mu_post, sig_sq_post)
colnames(posterior_mat) <- c("mu", "sigma^2")
```

7 Diagnostics

7.1 Trace Plots

Plotting a trace plot is a simple first step you can take to check for good mixing and convergence. It should look like a furry caterpillar.

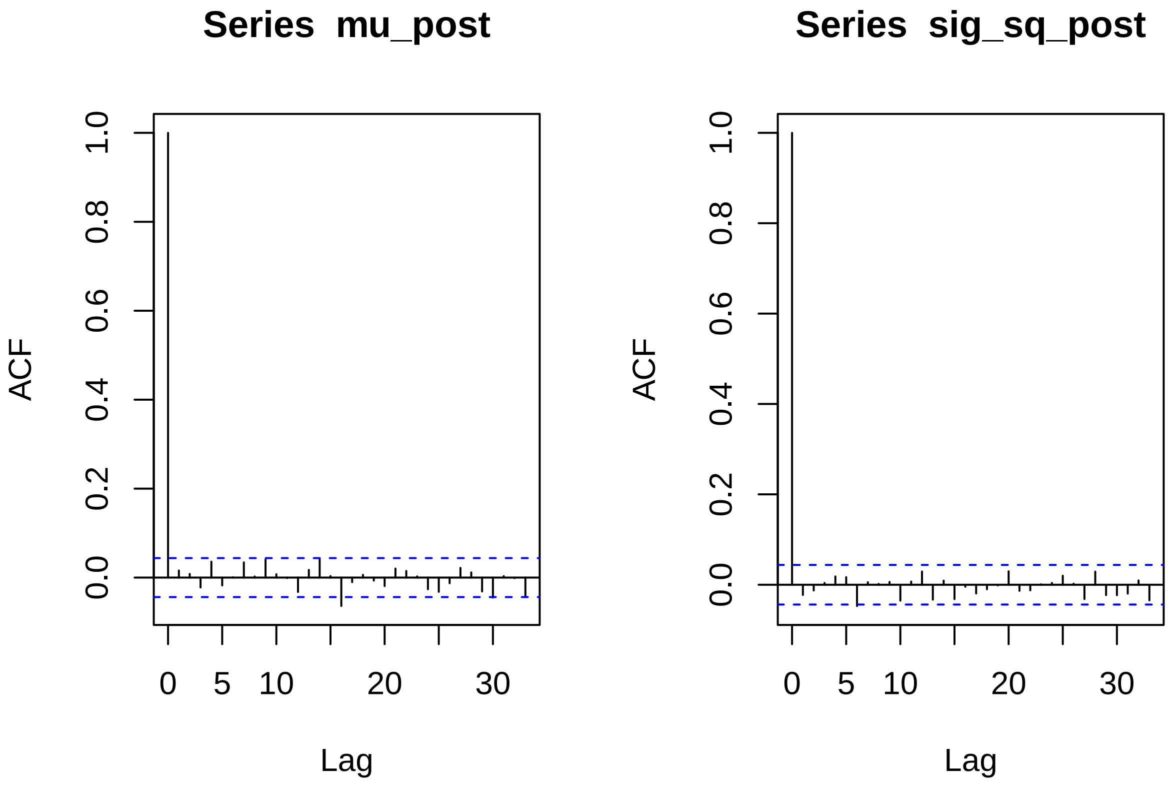
```
mcmc_trace(posterior_mat)
```



7.2 ACF Plots

Check the auto correlation function (ACF) of your posterior draws. You want as close to independent samples as possible so there should be little correlation between each draw

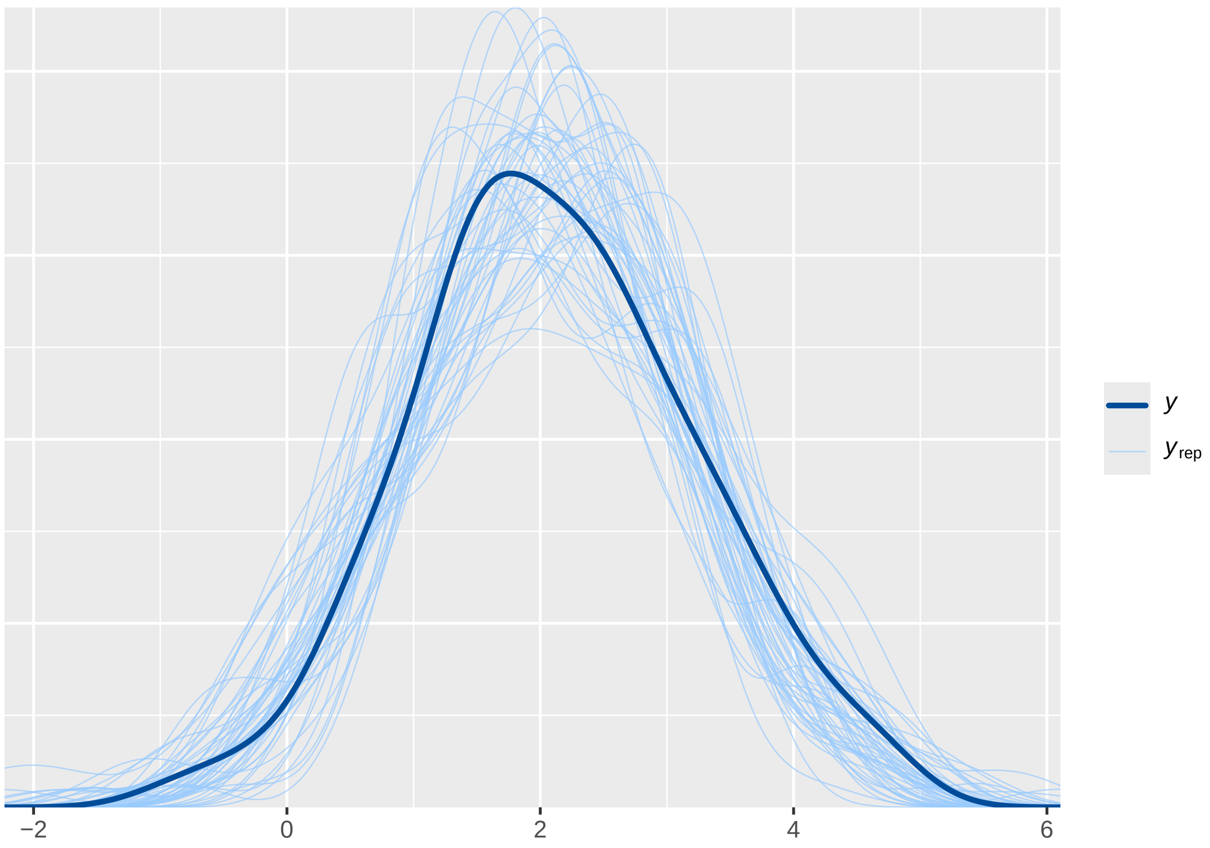
```r
par(mfrow = c(1,2))
acf(mu_post)
acf(sig_sq_post)
```



7.3 Posterior Predictive Check

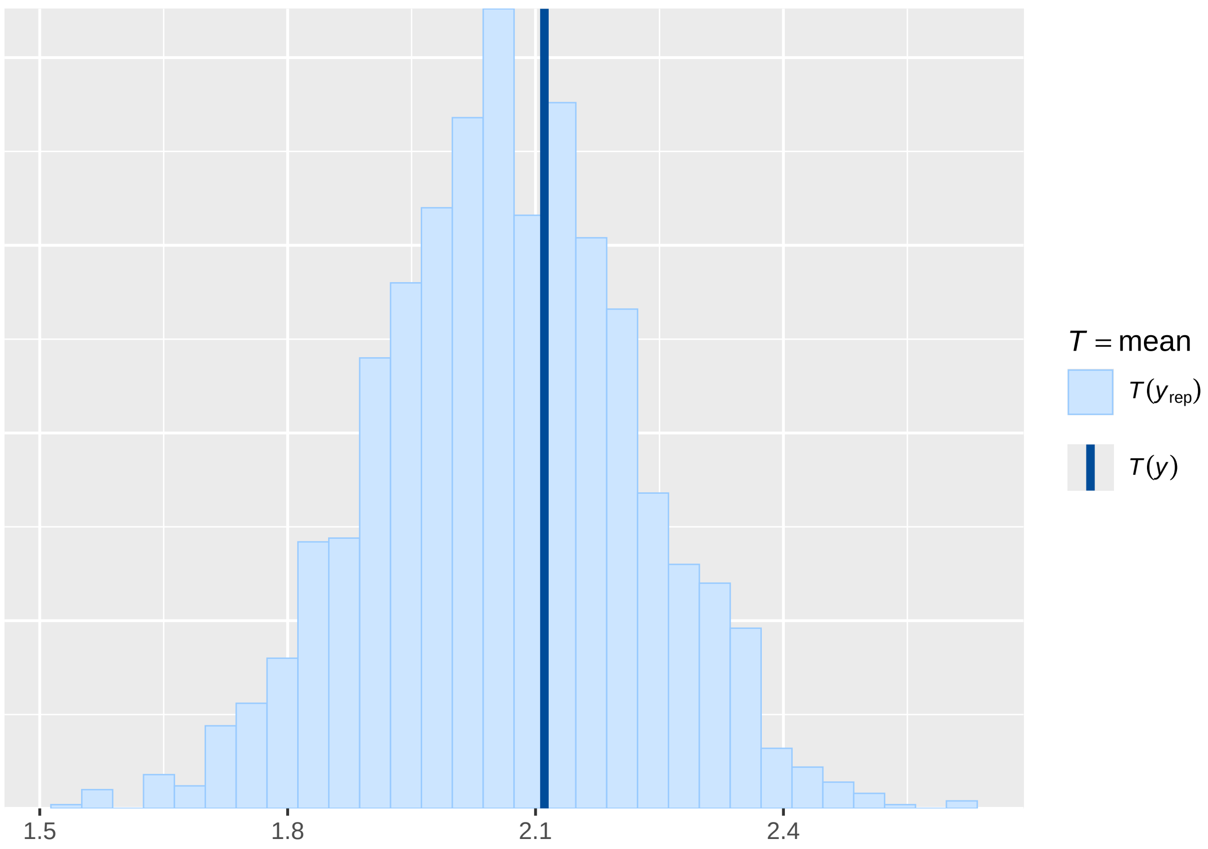
Posterior predictive checks are techniques to validate your model. Here, data is simulated using the posterior draws of the parameters and then compared to the observed data. If the posterior predictive distribution matches the observations, this is a good indication that your model is fitting well. You can also compare the empirical summaries of the posterior predictive distribution to the parameters to see if they are consistent.

```r
n_draws <- length(mu_post)
n_data  <- length(y_data)

# Generate posterior predictive draws
yrep <- matrix(NA, nrow = n_draws, ncol = n_data)

for (t in seq_len(n_draws)) {
  yrep[t, ] <- rnorm(n_data, mean = mu_post[t], sd = sqrt(sig_sq_post[t]))
}

# 4. Posterior Predictive Checks
color_scheme_set("brightblue")

# Density Overlay
ppc_dens_overlay(y = y_data, yrep = yrep[1:50, ])
```



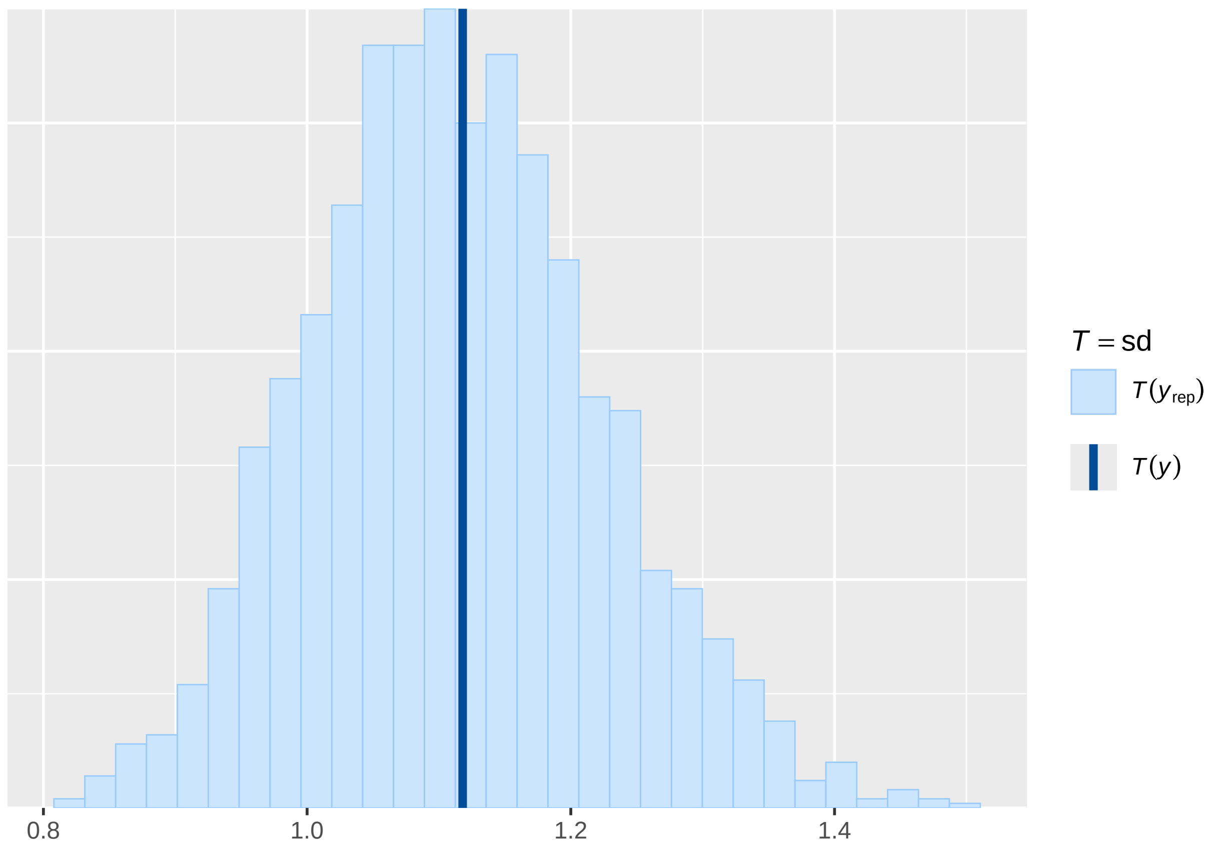
```r
# Compare mean
ppc_stat(y_data, yrep, stat = "mean")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```r
# Compare standard deviation
ppc_stat(y_data, yrep, stat = "sd")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```


0.8
1.0
1.2
1.4
$T = \mathrm{sd}$
$T(y_{\mathrm{rep}})$
$T(y)$

8 Posterior Summary

Once you sample from the posterior, you can use these methods to summarize them. Below are the common types of summaries that are expected.

8.1 Summary Table

```r
# Compute summary statistics for posterior samples
summary_stats <- data.frame(
  Parameter = c("mu", "sigma^2"),
  Mean = apply(posterior_mat, 2, mean),
  SD = apply(posterior_mat, 2, sd),
  `2.5%` = apply(posterior_mat, 2, quantile, probs = 0.025),
  `50%` = apply(posterior_mat, 2, quantile, probs = 0.5),  # Median
  `97.5%` = apply(posterior_mat, 2, quantile, probs = 0.975),
  check.names = FALSE # prevent automatic naming
)

# Print summary table
summary_stats
```

```
##         Parameter     Mean        SD      2.5%       50%    97.5%
## mu             mu 2.066667 0.1111576 1.8425890 2.068150 2.282428
## sigma^2   sigma^2 1.248574 0.1768884 0.9557287 1.231083 1.632152
```

You can output the contents of the table to Latex code.

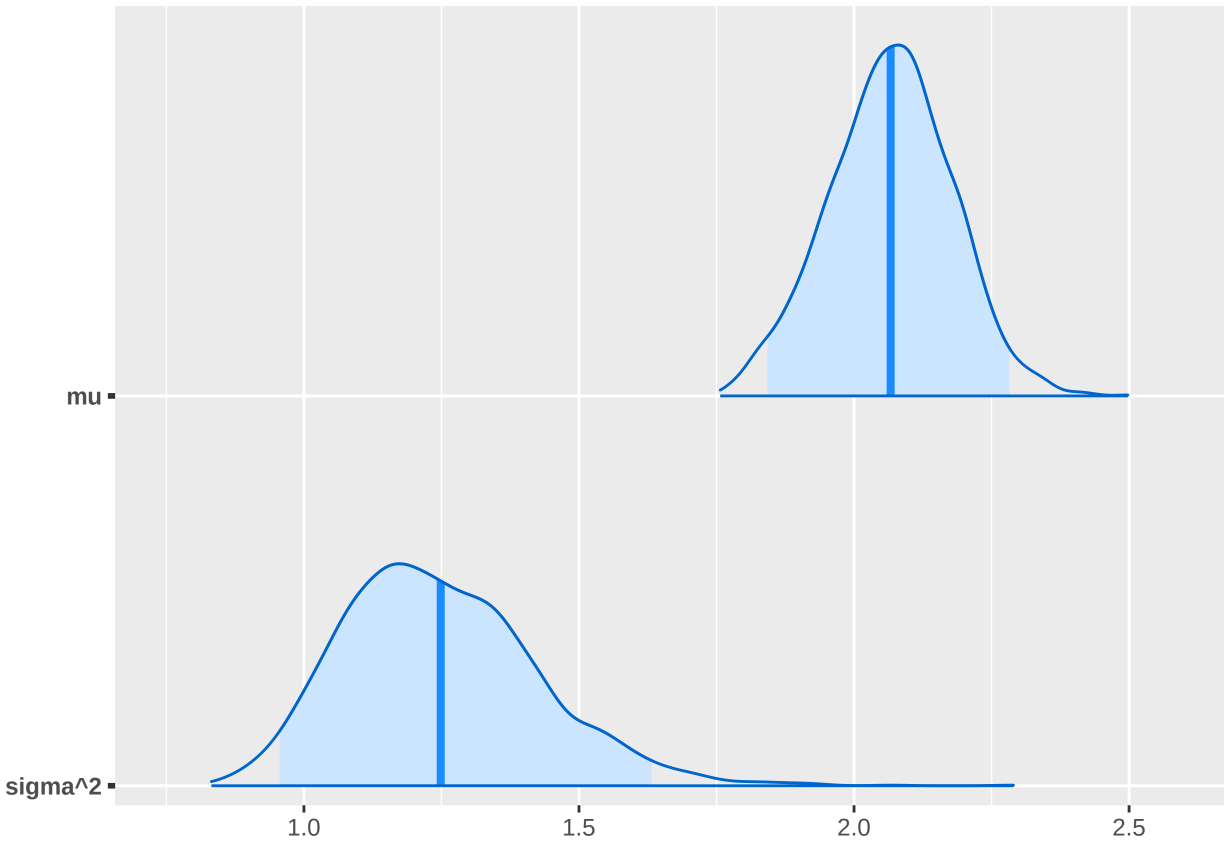
```r
# Convert to LaTeX code
latex_table <- xtable(summary_stats, digits = 3)

# Print the LaTeX code
print(latex_table, type = "latex", include.rownames = FALSE)
```

```
## % latex table generated in R 4.2.1 by xtable 1.8-4 package
## % Tue Feb 18 16:42:04 2025
## \begin{table}[ht]
## \centering
## \begin{tabular}{lrrrrr}
##   \hline
## Parameter & Mean & SD & 2.5\% & 50\% & 97.5\% \\
##   \hline
## mu & 2.067 & 0.111 & 1.843 & 2.068 & 2.282 \\
##   sigma\verb|^|2 & 1.249 & 0.177 & 0.956 & 1.231 & 1.632 \\
##    \hline
## \end{tabular}
## \end{table}
```

8.2 Density Estimate

You can control the point estimate and width of the credible intervals using the "prob" and "point_est" arguments.

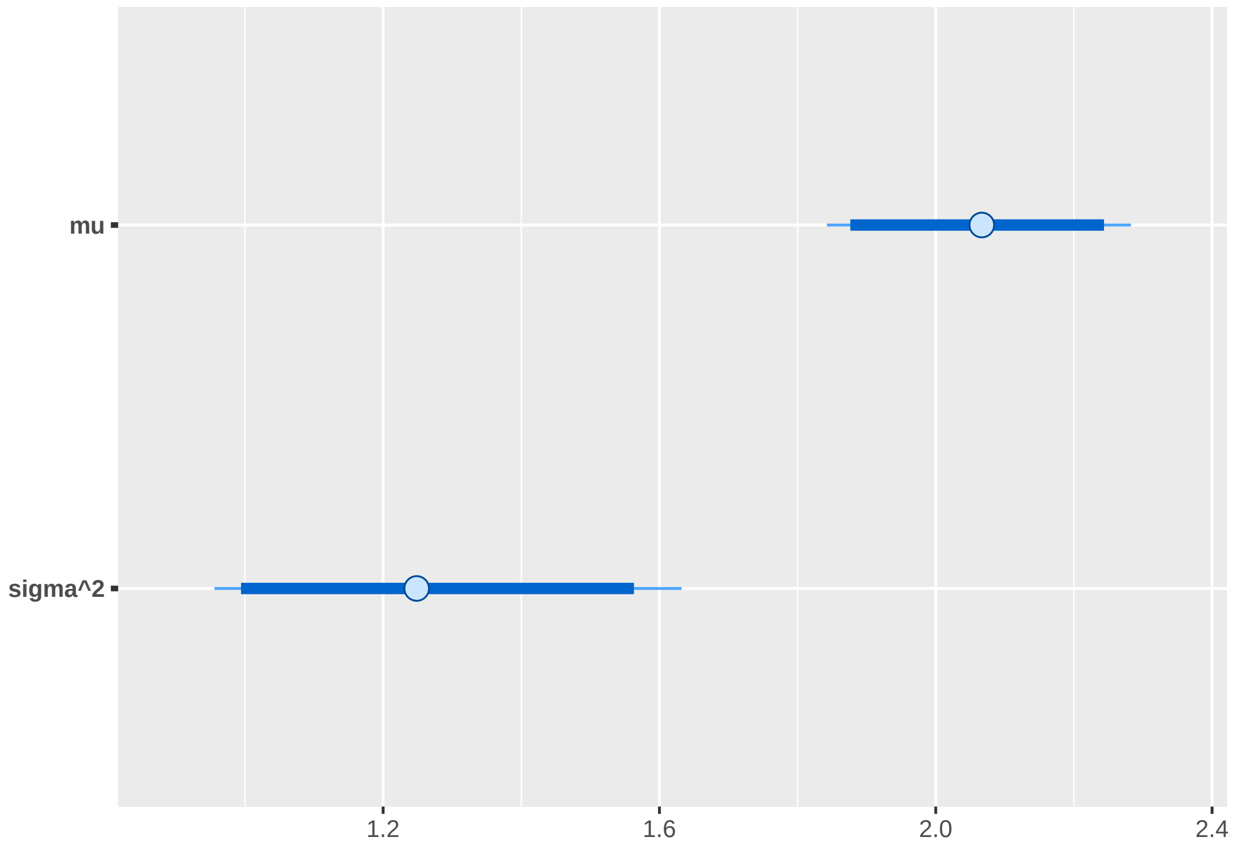
```r
mcmc_areas(posterior_mat, prob = 0.95, point_est = "mean") # 95% credible interval
```



8.3 Box plots

```r
mcmc_intervals(posterior_mat, prob = 0.95, point_est = "mean")
```

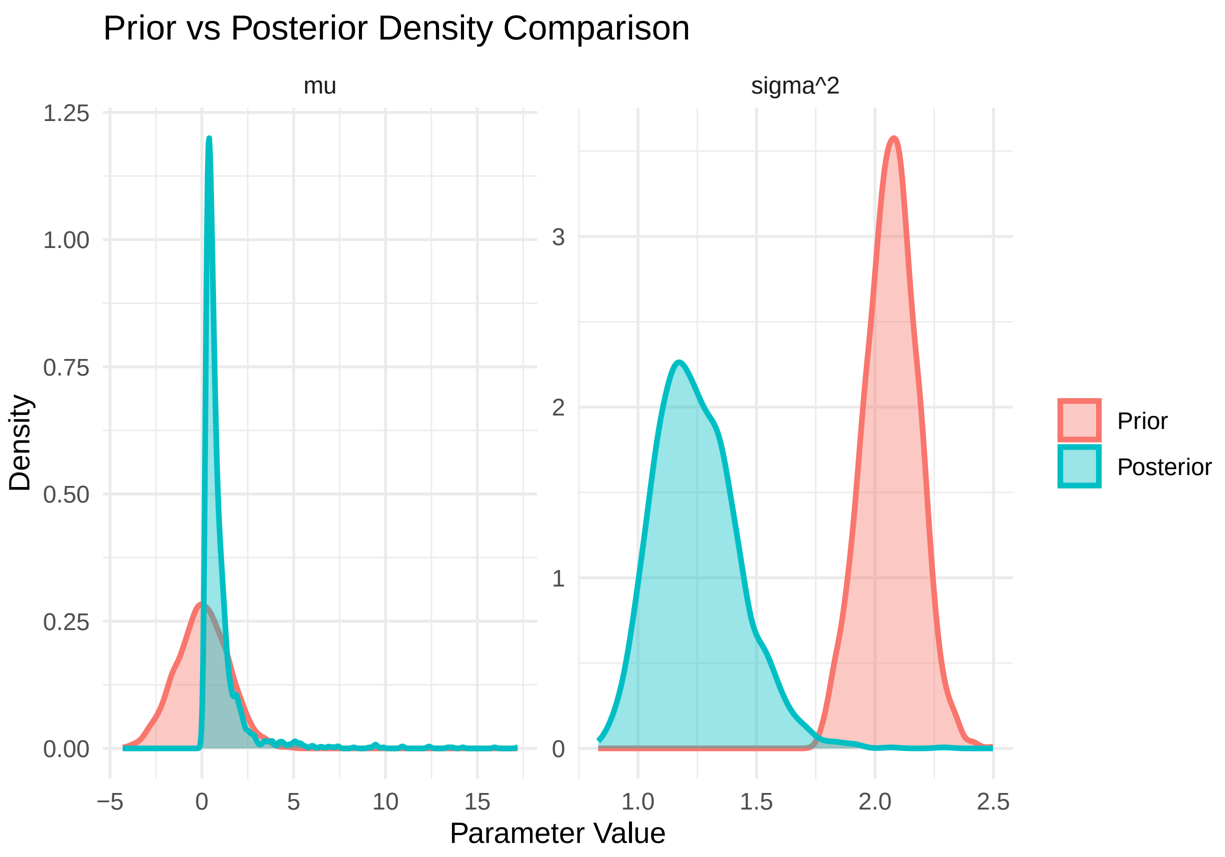
```
## Warning: `prob_outer` (0.9) is less than `prob` (0.95)
## ... Swapping the values of `prob_outer` and `prob`
```



9 Prior Sensitivity Analysis

Comparing the the prior and posterior distribution of the parameters allows us to see how much "learning" there was from the data. If the posterior drastically changes under different priors, then you know that parameters are sensitive to prior specification.

```r
set.seed(123)

# Define prior distributions
prior_mu <- rnorm(length(mu_post), mean = 0, sd = sqrt(1 / 0.5))  # Normal prior for mu
prior_sigma2 <- 1 / rgamma(length(sig_sq_post), shape = 2, rate = 1)  # Inverse Gamma prior for sigma^2

# Create data frame for ggplot
df <- data.frame(
  value = c(prior_mu, mu_post, prior_sigma2, sig_sq_post),
  type = rep(c("Prior", "Posterior"), each = length(mu_post) * 2),
  parameter = rep(c("mu", "sigma^2"), each = length(mu_post), times = 2)
)

df$type <- factor(df$type, levels = c("Prior", "Posterior"))

ggplot(df, aes(x = value, fill = type, color = type)) +
  geom_density(alpha = 0.4, linewidth = 1) +
  facet_wrap(~parameter, scales = "free") +
  labs(title = "Prior vs Posterior Density Comparison",
       x = "Parameter Value", y = "Density") +
  theme_minimal() +
  theme(legend.title = element_blank())
```



10 Metropolis Hastings

If you need to sample from a posterior distribution that does not follow a form that can be directly sampled from, you can use the Metropolis-Hastings (MH) algorithm.

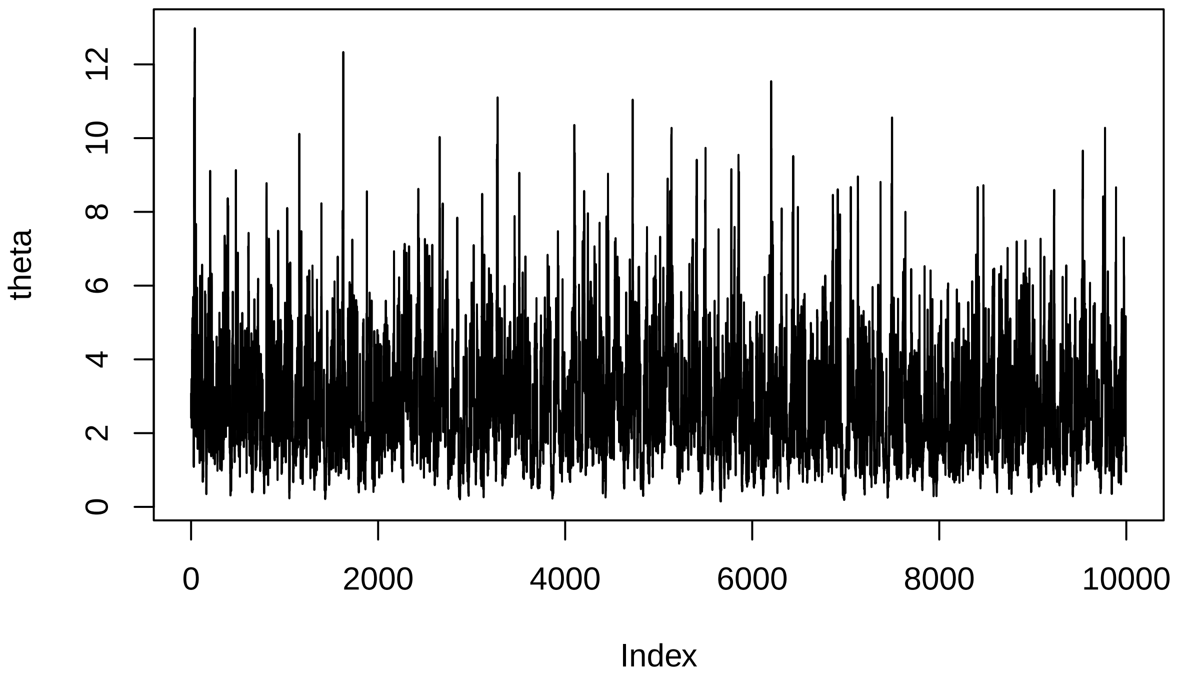
Notes:

- If the posterior has positive support, define the proposal distribution in terms of the log transformation of the random variable, and then transform back.
- Generally you should define your acceptance ratio in terms of a log scale to avoid numerical issues that may arise.
- You may need to adjust the acceptance ratio to achieve better mixing. A rule of thumb to follow is that the acceptance rate should be between 20% to 30%.
  - If your trace plot has a lot of stagnant horizontal lines, decrease the step size. Otherwise if it looks fuzzy with lots of small jumps, increase the step size.

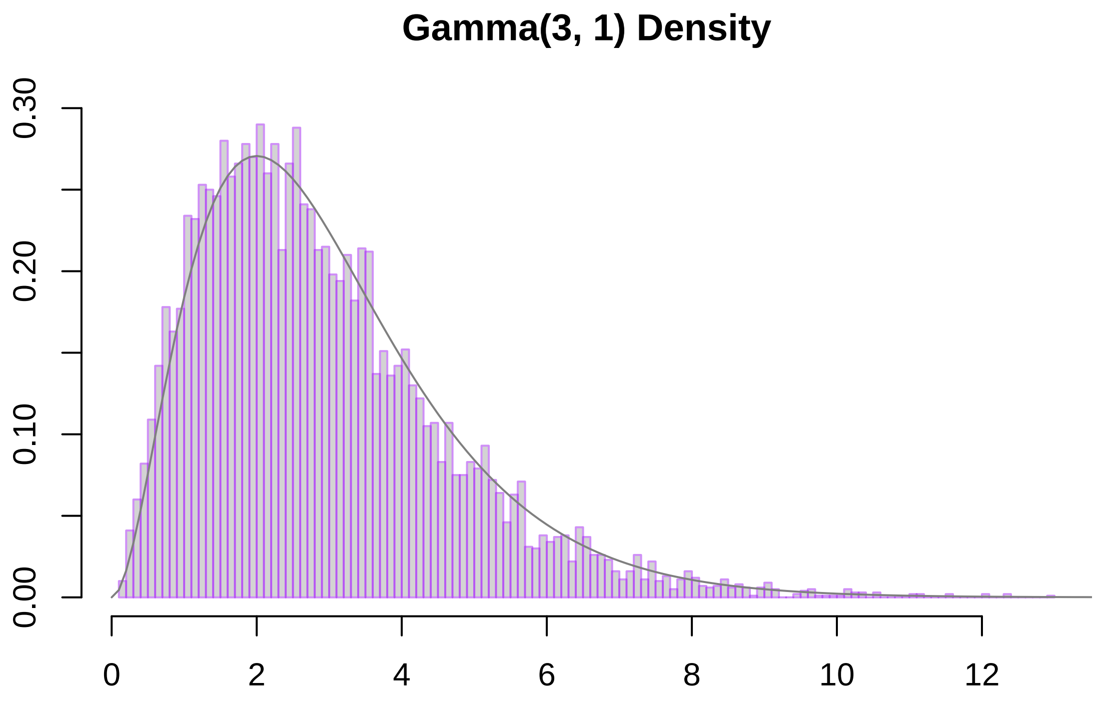
Below is a simple example of the MH algorithm for sampling a Gamma(3, 1) distribution. Adapt this code by changing the log target function to the log posterior you want to sample from.

```r
log_jacobian <- function(theta) {
  return(-log(theta))
}

#####
# Example: Target distribution <- Gamma(3, 1)
###

log_target <- function(theta) {
  return(dgamma(theta, shape = 3, rate = 1, log = TRUE))
}

theta <- rep(NA, 10000)

num_sim <- 10000

theta[1] <- rgamma(1, shape = 3, rate = 1) #initialize theta

for (t in 2:num_sim) {
  theta_curr <- theta[t - 1]
  v_star <- rnorm(1, mean = log(theta_curr), sd = 0.5)
  theta_prop <- exp(v_star)

  log_A_num <- log_target(theta_prop)

  log_A_denom <- log_target(theta_curr)

  log_ratio <- min(0, (log_A_num + log_jacobian(theta_curr)
                       - (log_A_denom + log_jacobian(theta_prop))))

  if (log_ratio >= log(runif(1))) {
    theta[t] <- theta_prop
  } else{
    theta[t] <- theta_curr
  }
}
```

```r
plot(theta, type = "l")
```



```r
gd = seq(0, 15, .1)
purple.5 = scales::alpha("purple", .5)

hist(theta, freq = FALSE, xlab = "", ylab = "", breaks = 100,
     border = purple.5, main = "Gamma(3, 1) Density")
lines(gd, dgamma(gd, 3, 1), col = "grey50")
```


Gamma(3, 1) Density