Random_Walk_Metropolis

Jose Michel Sammut

2024-07-17

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

Use Ctrl + Alt + I (Windows/Linux) to insert a new code chunk in your RMarkdown document.

Question 1

Markov Chain Monte Carlo (MCMC) methods are a class of algorithms used to sample from probability distributions when direct sampling is difficult. They are particularly useful for estimating the distribution of complex, high-dimensional spaces.

The **Metropolis-Hastings** algorithm is a specific type of MCMC method. It generates a sequence of sample values from a target probability distribution $f(x)$, even if the distribution is known only up to a normalizing constant.

$$\textbf{Target Distribution}: \quad f(x) = \frac{1}{2}e^{-|x|}$$

This is the probability density function of the **Laplace Distribution** (also known as the double-exponential distribution), centered at 0 with a scale parameter of 1.

```r
set.seed(123)  # Used to generate figures, for reproducibility.

# Define the Probability Density Function
f <- function(x) { 0.5 * exp(-abs(x)) }

# Metropolis-Hastings Algorithm
metropolis_hastings <- function(N, s, x0) {
  samples <- numeric(N)
  samples[1] <- x0

  for (i in 2:N) {
    x_star <- rnorm(1, mean = samples[i-1], sd = s)
    r <- f(x_star) / f(samples[i-1])
    u <- runif(1)

    if (log(u) < log(r)) {
      samples[i] <- x_star
```

```
    } else {
      samples[i] <- samples[i-1]
    }
  }

  return(samples)
}

# Generate samples
N <- 10000
s <- 1
x0 <- 0
samples <- metropolis_hastings(N, s, x0)

# Calculate sample mean and standard deviation
sample_mean <- mean(samples)
sample_sd <- sd(samples)

# Output the mean and standard deviation
sample_mean
```

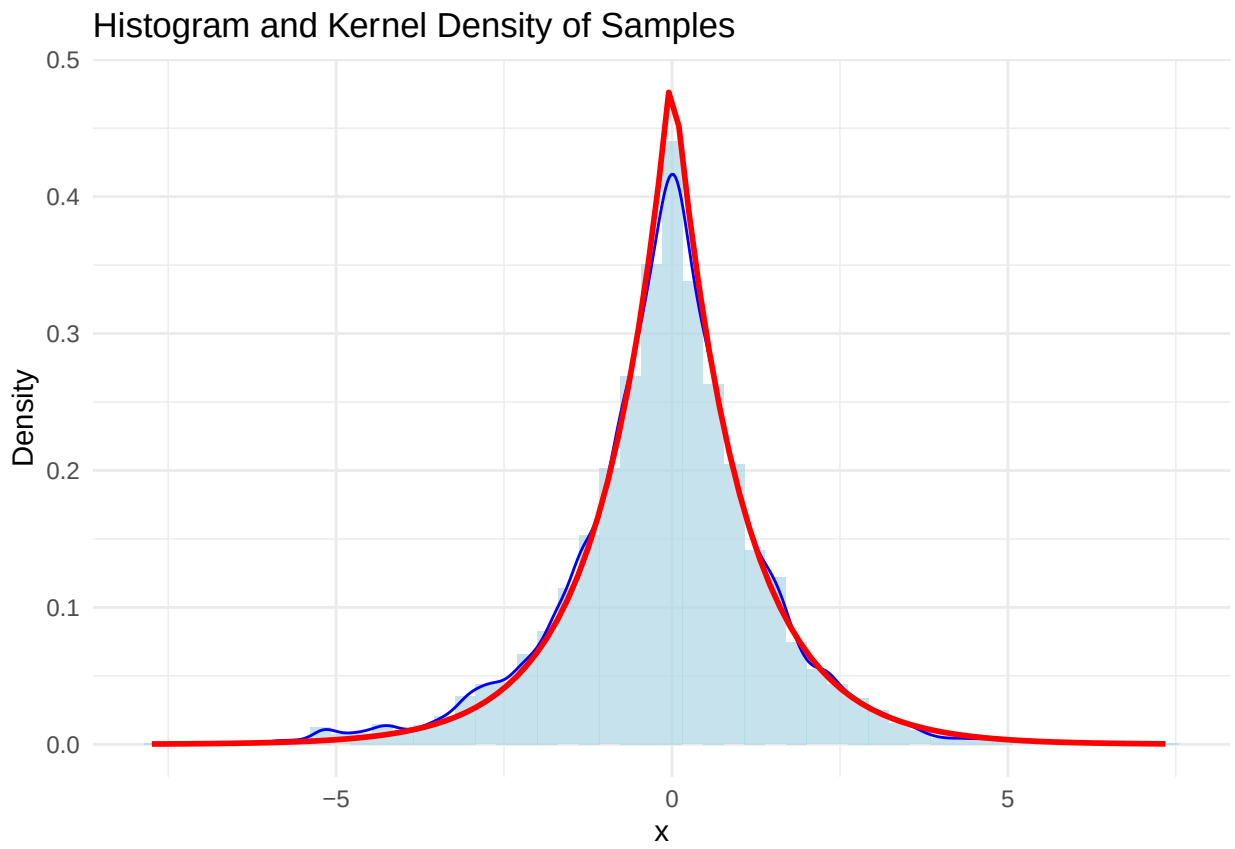
```
## [1] -0.0943715
```

```
sample_sd
```

```
## [1] 1.481383
```

```
library(ggplot2)

# Create histogram and density plot
ggplot(data.frame(samples = samples), aes(x = samples)) +
  geom_histogram(aes(y = after_stat(density)), bins = 50,
                 fill = 'lightblue', alpha = 0.7) +
  geom_density(color = 'blue') +
  stat_function(fun = f, color = 'red', linewidth = 1) +
  labs(title = "Histogram and Kernel Density of Samples",
       x = "x", y = "Density") +
  theme_minimal()
```



The **Gelman-Rubin Diagnostic** (denoted $\hat{R}$) is widely used in MCMC methods, to assess convergence by comparing within-chain and between-chain variances.

$$\hat{R} = \sqrt{\frac{B+W}{W}}$$

```r
# Function to calculate Rhat
calculate_rb <- function(N, s, J) {
  chains <- list()
  initial_values <- rnorm(J)

  for (j in 1:J) {
    chains[[j]] <- metropolis_hastings(N, s, initial_values[j])
  }

  Mjs <- sapply(chains, mean)
  Vjs <- sapply(chains, var)

  W <- mean(Vjs)
  M <- mean(Mjs)
  B <- sum((Mjs - M)^2) / J

  Rb <- sqrt((B + W) / W)

  return(Rb)
}
```

```r
# Parameters
N <- 2000
J <- 4

# Calculate Rhat for s = 0.001
s <- 0.001
Rb_specific <- calculate_rb(N, s, J)

# Output the Rhat value for specified parameters
Rb_specific
```

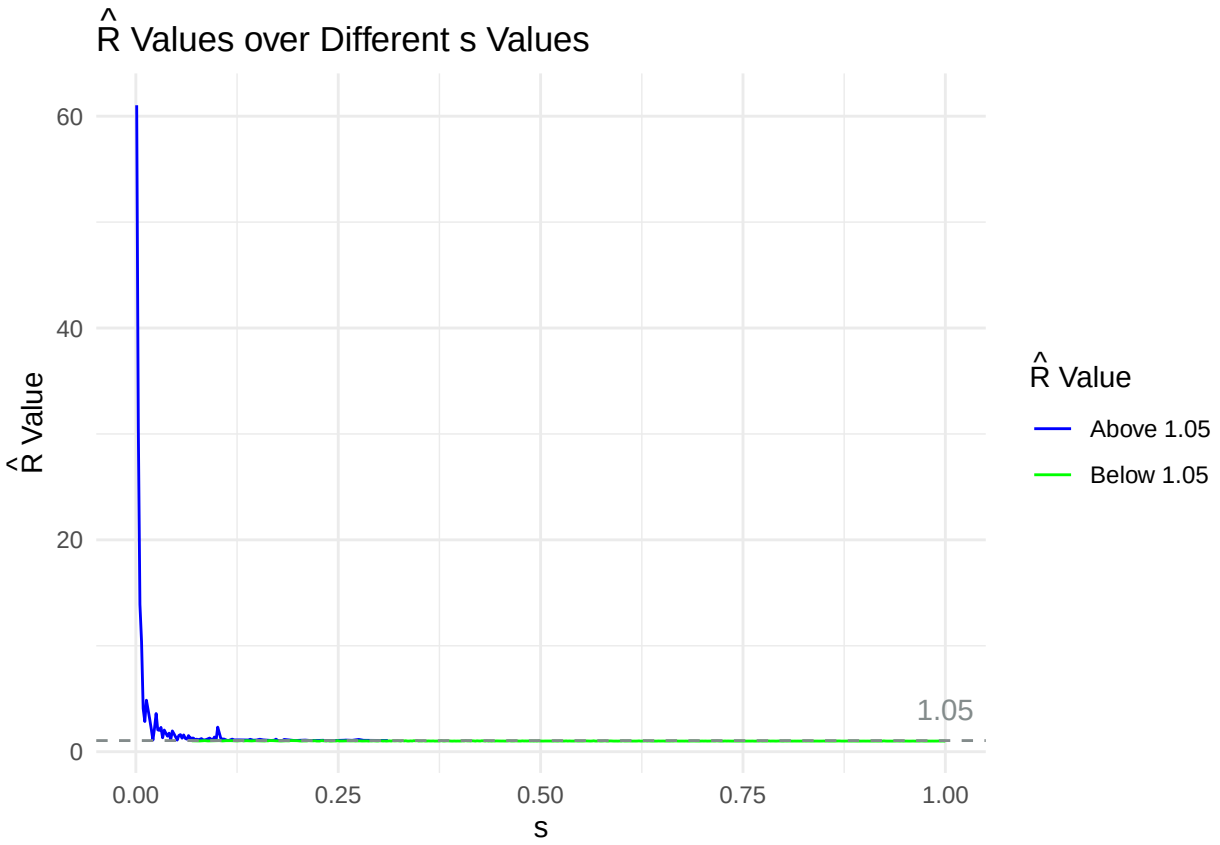
```
## [1] 18.95922
```

In the context of the Metropolis-Hastings algorithm, s is the standard deviation of the proposal distribution. It controls how far the algorithm proposes to move in each step. The choice of s affects the efficiency and convergence of the algorithm.

```r
# Calculate Rhat over a grid of s values
s_values <- seq(0.001, 1, length.out = 500)
Rb_values <- sapply(s_values, calculate_rb, N = N, J = J)

# Create a data frame for plotting
plot_data <- data.frame(s = s_values, Rb = Rb_values)
plot_data$color <- ifelse(plot_data$Rb <= 1.05, "Below 1.05", "Above 1.05")

# Plot Rhat values
ggplot(plot_data, aes(x = s, y = Rb, color = color)) +
  geom_line() +
  geom_hline(yintercept = 1.05, linetype = "dashed", color = "azure4") +
  annotate("text", x = max(s_values), y = 1.05,
           label = "1.05", vjust = -1, color = "azure4") +
  scale_color_manual(values = c("Below 1.05" = "green",
                                "Above 1.05" = "blue")) +
  labs(title = expression(paste(hat(R), " Values over Different s Values")),
       x = "s",
       y = expression(paste(hat(R), " Value")),
       color = expression(paste(hat(R), " Value"))) +
  theme_minimal()
```



Values of $\hat{R}$ close to 1 indicate convergence, and it is usually desired for $\hat{R}$ to be lower than 1.05. Small s values result in poor convergence due to insufficient exploration (small steps). Thus values of $s > 0.125$ are desired. Taking unnecessarily large s values may still yield good convergence but is potentially inefficient due to high rejection rates.