Problem Set 2

Sungsik Hwang

February 17, 2025

Random-walk Metropolis algorithm

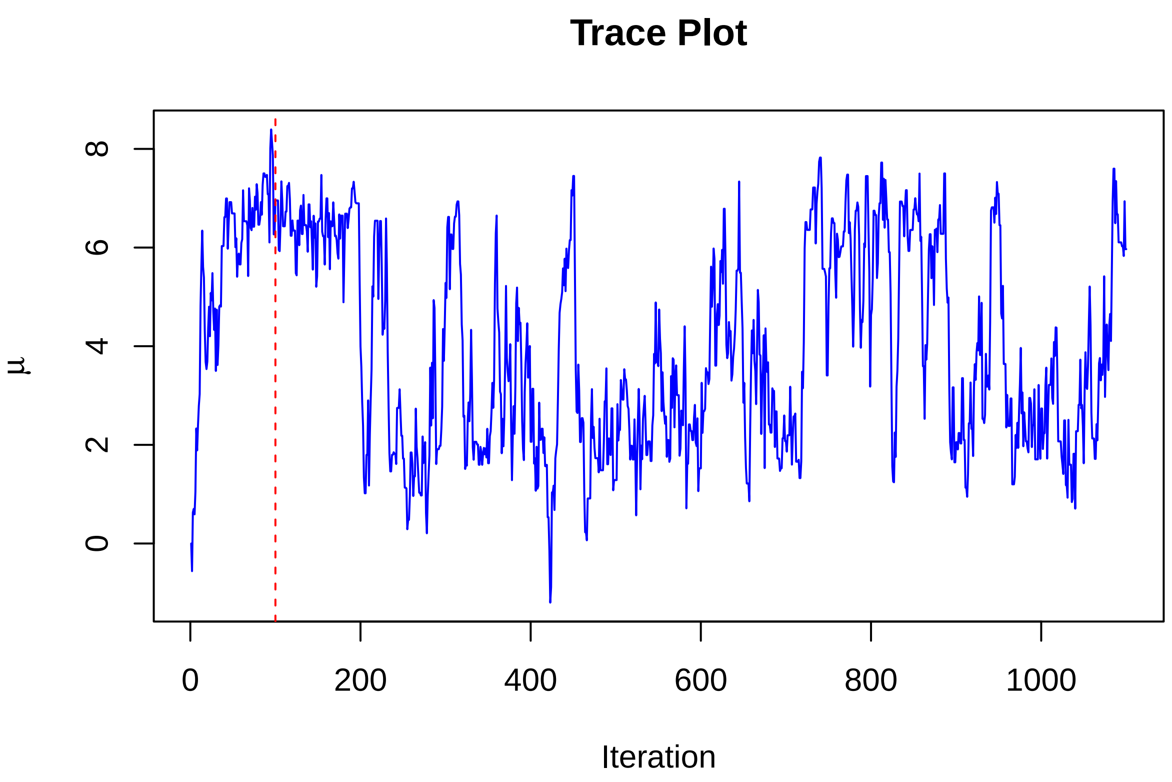
create the function

```r
library(coda)
library(ggplot2)
library(rstan)
y <- c(1.85, 7.05, 7.28, 22.616)
likelihood <- function(mu) {
  prod(dcauchy(y, location = mu, scale = 1))
}
prior <- function(mu) {
  dnorm(mu, mean = 0, sd = 2.5)
}
posterior <- function(mu) {
  likelihood(mu) * prior(mu)
}

set.seed(123)
n_iter <- 1100
burn_in <- 100
mu_current <- 0
samples <- numeric(n_iter)
samples[1] <- mu_current
for (i in 2:n_iter) {
  mu_prop <- rnorm(1, mean = mu_current, sd = 1)
  r <- posterior(mu_prop) / posterior(mu_current)
  if (runif(1) < min(1, r)) {
    mu_current <- mu_prop
  }
  samples[i] <- mu_current
}
posterior_samples <- samples[(burn_in + 1):n_iter]
```

(a) Trace Plot

```r
plot(samples, type = "l", col = "blue", main = "Trace Plot", xlab = "Iteration", ylab = expression(mu))
abline(v = burn_in, col = "red", lty = 2)
```



(b) Convergence & autocorrelation

```
# It does not seem to converge and shows strong autocorrelation.
```

(c) Effective Sample Size

```
mcmc_samples <- as.mcmc(posterior_samples)
ess <- effectiveSize(mcmc_samples)
ess
```

```
##     var1
## 36.18585
```

(d) Raftery-Lewis Diagnostic

```
raftery <- raftery.diag(mcmc_samples, q = 0.5, r = 0.01, s = 0.95)
print(raftery)
```

```
##
## Quantile (q) = 0.5
## Accuracy (r) = +/- 0.01
## Probability (s) = 0.95
##
## You need a sample size of at least 9604 with these values of q, r and s
```

(e) Rerun the Chain

```r
N_required <- 9604
burn_in2 <- floor(0.1 * N_required)
mu_current2 <- 0
samples2 <- numeric(N_required)
samples2[1] <- mu_current2
for (i in 2:N_required) {
  mu_prop <- rnorm(1, mean = mu_current2, sd = 1)
  r <- posterior(mu_prop) / posterior(mu_current2)
  if (runif(1) < min(1, r)) {
    mu_current2 <- mu_prop
  }
  samples2[i] <- mu_current2
}
posterior_samples2 <- samples2[(burn_in2 + 1):N_required]
posterior_mean <- mean(posterior_samples2)
posterior_median <- median(posterior_samples2)
cred_interval <- quantile(posterior_samples2, probs = c(0.025, 0.975))
cat("Posterior mean:", posterior_mean, "\n")
```

```
## Posterior mean: 4.327586
```

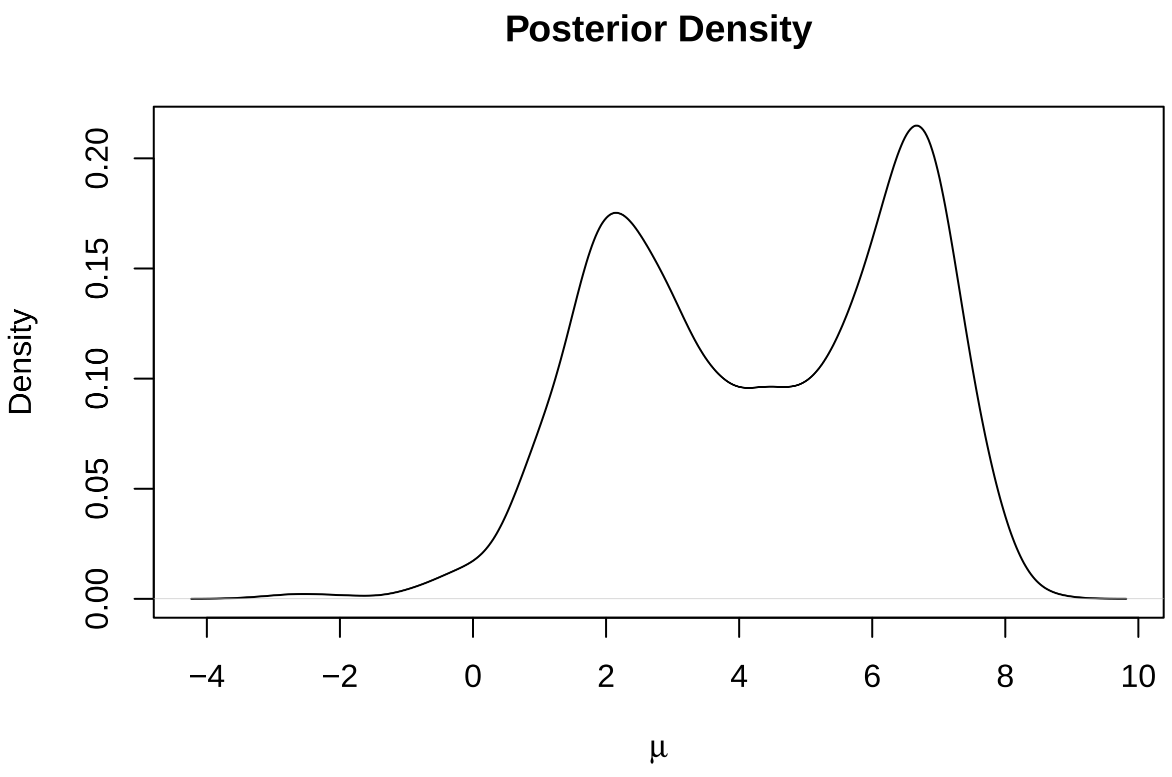
```r
cat("Posterior median:", posterior_median, "\n")
```

```
## Posterior median: 4.424224
```

```r
cat("95% CI:", cred_interval, "\n")
```

```
## 95% CI: 0.5443187 7.633466
```

```r
plot(density(posterior_samples2), main = "Posterior Density", xlab = expression(mu), ylab = "Density")
```



Stan

Stan model

```r
stan_code <- "
data {
  int<lower=0> N;
  real y[N];
}
parameters {
  real mu;
}
model {
  mu ~ normal(0, 2.5);
  y ~ cauchy(mu, 1);
}
"
stan_data <- list(
  N = length(y),
  y = y
)

fit <- stan(model_code = stan_code, data = stan_data,
            iter = 600, warmup = 100, chains = 2, seed = 123)
```

```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 4.1e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.41 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: WARNING: There aren't enough warmup iterations to fit the
## Chain 1:          three stages of adaptation as currently configured.
## Chain 1:          Reducing each adaptation stage to 15%/75%/10% of
## Chain 1:          the given number of warmup iterations:
## Chain 1:            init_buffer = 15
## Chain 1:            adapt_window = 75
## Chain 1:            term_buffer = 10
## Chain 1:
## Chain 1: Iteration:   1 / 600 [  0%]  (Warmup)
## Chain 1: Iteration:  60 / 600 [ 10%]  (Warmup)
## Chain 1: Iteration: 101 / 600 [ 16%]  (Sampling)
## Chain 1: Iteration: 160 / 600 [ 26%]  (Sampling)
## Chain 1: Iteration: 220 / 600 [ 36%]  (Sampling)
## Chain 1: Iteration: 280 / 600 [ 46%]  (Sampling)
## Chain 1: Iteration: 340 / 600 [ 56%]  (Sampling)
## Chain 1: Iteration: 400 / 600 [ 66%]  (Sampling)
## Chain 1: Iteration: 460 / 600 [ 76%]  (Sampling)
## Chain 1: Iteration: 520 / 600 [ 86%]  (Sampling)
## Chain 1: Iteration: 580 / 600 [ 96%]  (Sampling)
## Chain 1: Iteration: 600 / 600 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.001 seconds (Warm-up)
## Chain 1:                0.005 seconds (Sampling)
## Chain 1:                0.006 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 4e-06 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.04 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: WARNING: There aren't enough warmup iterations to fit the
## Chain 2:          three stages of adaptation as currently configured.
## Chain 2:          Reducing each adaptation stage to 15%/75%/10% of
## Chain 2:          the given number of warmup iterations:
## Chain 2:            init_buffer = 15
## Chain 2:            adapt_window = 75
## Chain 2:            term_buffer = 10
## Chain 2:
## Chain 2: Iteration:   1 / 600 [  0%]  (Warmup)
## Chain 2: Iteration:  60 / 600 [ 10%]  (Warmup)
## Chain 2: Iteration: 101 / 600 [ 16%]  (Sampling)
## Chain 2: Iteration: 160 / 600 [ 26%]  (Sampling)
## Chain 2: Iteration: 220 / 600 [ 36%]  (Sampling)
## Chain 2: Iteration: 280 / 600 [ 46%]  (Sampling)
## Chain 2: Iteration: 340 / 600 [ 56%]  (Sampling)
## Chain 2: Iteration: 400 / 600 [ 66%]  (Sampling)
## Chain 2: Iteration: 460 / 600 [ 76%]  (Sampling)
## Chain 2: Iteration: 520 / 600 [ 86%]  (Sampling)
## Chain 2: Iteration: 580 / 600 [ 96%]  (Sampling)
## Chain 2: Iteration: 600 / 600 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.001 seconds (Warm-up)
## Chain 2:                0.006 seconds (Sampling)
## Chain 2:                0.007 seconds (Total)
## Chain 2:
```

```r
samples <- extract(fit)$mu
mcmc_samples <- as.mcmc(as.matrix(samples))
```

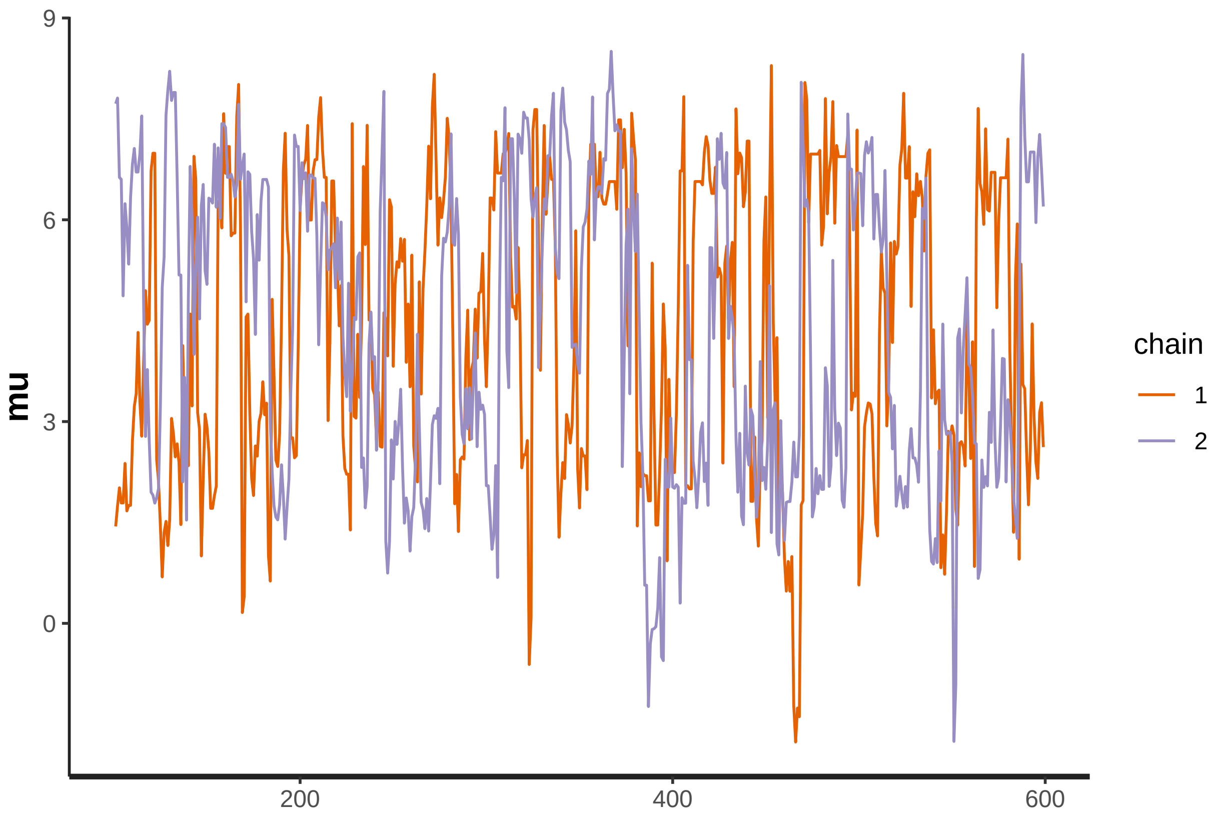
(a) effective sample size

```r
ess <- effectiveSize(mcmc_samples)
ess
```

```
## var1
## 1000
```

(b) Produce a trace plot of the samples

```r
stan_trace(fit, pars = "mu")
```



(c) Gelman-Rubin diagnostic

```
mcmc_samples <- As.mcmc.list(fit, pars = "mu")
gelman_diag <- gelman.diag(mcmc_samples, autoburnin = FALSE)
print(gelman_diag)
```

```
## Potential scale reduction factors:
##
##    Point est. Upper C.I.
## mu          1       1.01
```

(d) Extract posterior statistics

```
posterior_samples <- extract(fit)$mu
posterior_mean <- mean(posterior_samples)
posterior_median <- median(posterior_samples)
cred_interval <- quantile(posterior_samples, probs = c(0.025, 0.975))
cat("Posterior mean:", posterior_mean, "\n")
```

```
## Posterior mean: 4.384296
```

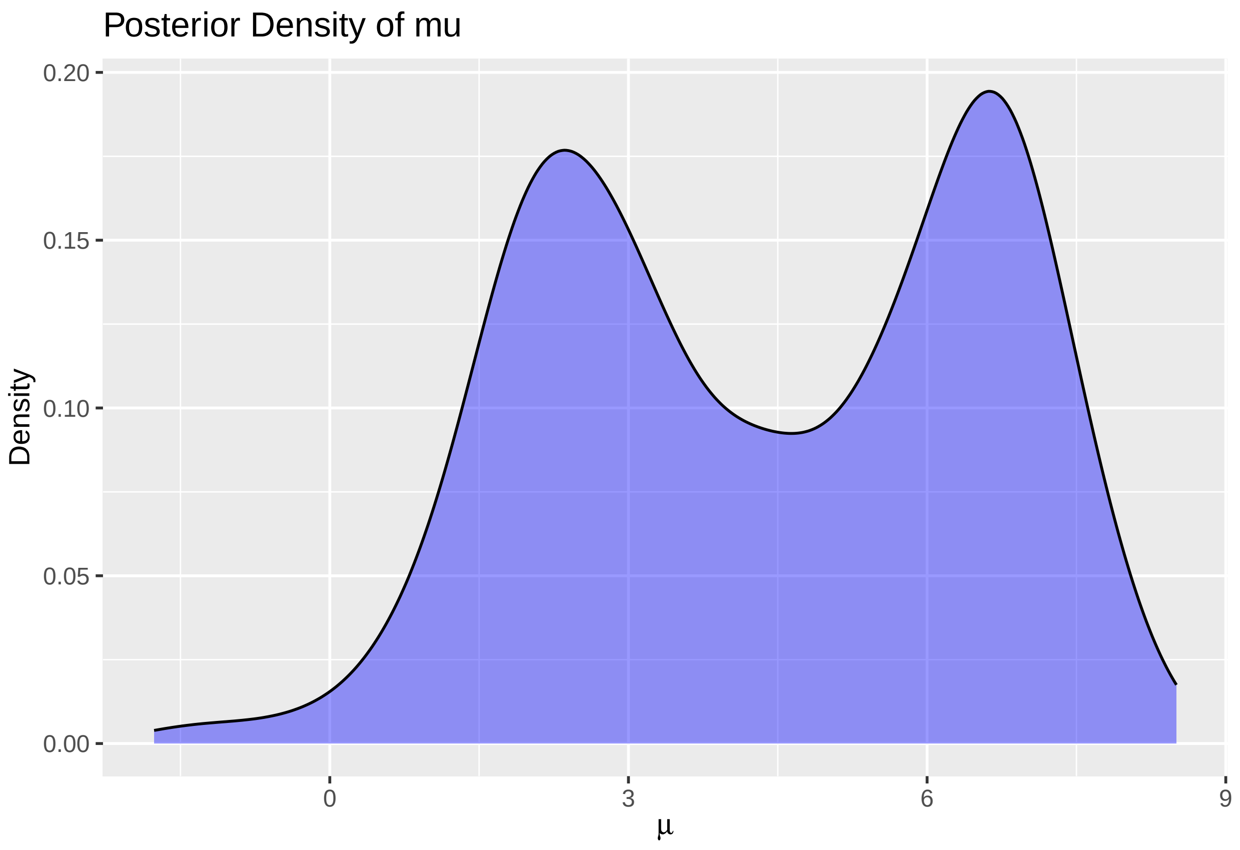
```
cat("Posterior median:", posterior_median, "\n")
```

```
## Posterior median: 4.406973
```

```
cat("95% CI:", cred_interval, "\n")
```

```
## 95% CI: 0.6667264 7.75784
```

```r
ggplot(data.frame(mu = posterior_samples), aes(x = mu)) +
  geom_density(fill = "blue", alpha = 0.4) +
  labs(title = "Posterior Density of mu", x = expression(mu), y = "Density")
```


Posterior Density of mu
Density
μ