ENG 107

Problem Set 4: Markov Chain Monte Carlo

Due: Friday, February 14

Hugh Shields

MCMC Implementation

In this case, we define our likelihood function as as the probability density of a given chain value x under a normal distribution $\mathcal{N}(\mu = 34, \sigma = 20)$. We let our prior be $\mathcal{U}(0, 182)$ so that the prior probability for a given chain value x is

$$\text{prior}(x) = \begin{cases} \frac{1}{182} & \text{if} \quad 0 \le x \le 182 \\ 0 & \text{else.} \end{cases}$$

We can convert these to log space and add them to get the log posterior of any given x. We then implement standard Metropolis Hastings with an initial value drawn from the prior distribution and a step size of 2σ. We choose this step size because it leads to an acceptance rate of $\sim$48% for each of the seeds tested, which is near ideal for one dimension. We choose $N = 10^5$ iterations (as this allows us to reach convergence) and show the resulting chains for three independent seeds in Figure 1:

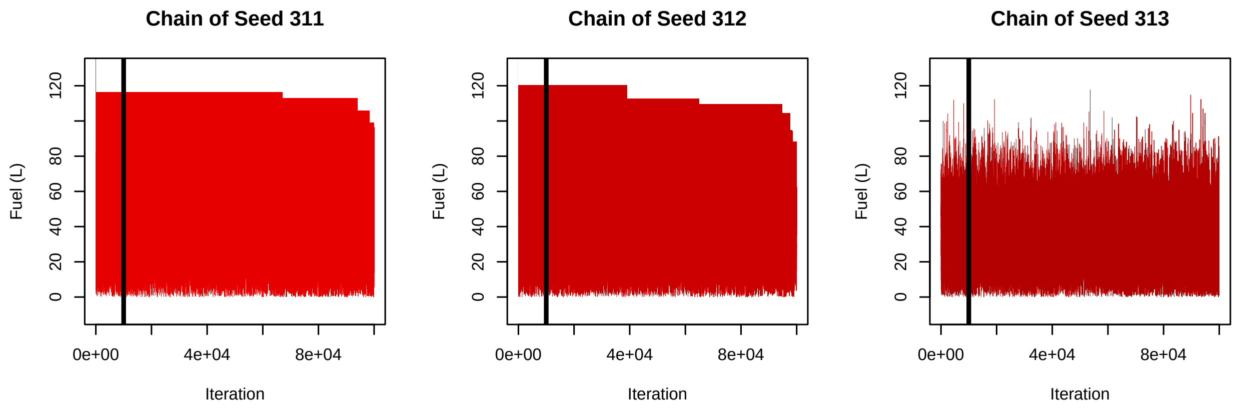

Figure 1: The MCMC chains for three separate seeds. The chosen burn-in value of 10^3 iterations is shown by the black vertical line.

Next, we show the generated posterior distributions in Figure 2. We also show the true posterior distribution, which is just a truncated normal distribution with mean $\mu = 34$ L, standard deviation $\sigma = 20$ L and bounds of $[0, 182]$ L. We implement this in code using the `truncnorm` package.

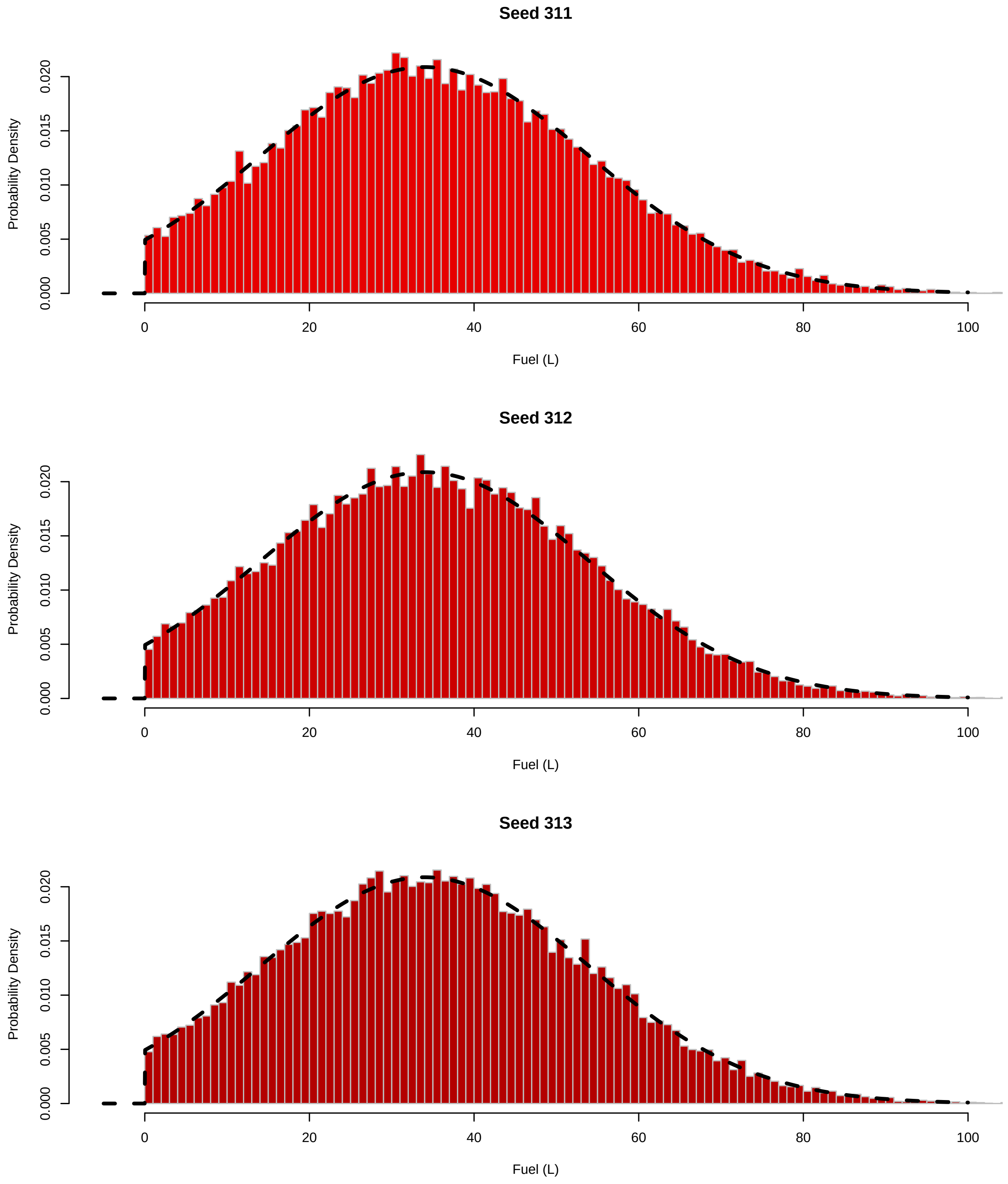

FIGURE 2: The MCMC generated posterior distributions for three separate seeds shown by the histograms. Histograms have 100 bins between 0 and 100. Overplotted in black is the true posterior distribution.

Assessing Convergence

As we can see in Figure 1, the trace plot shows that all three seeds appear to have reached a stationary distribution after a relatively small burn in period, with no drastic jumps or variability. Additionally, we see in Figure 2 that the mode of all three seeds have converged to around 34, as expected from likelihood. Thus, we can conclude that our numerical inference about the mode has converged.

Positive Control

For a positive control, we use our MCMC method to generate a normal distribution with mean $\mu = 0$ and standard deviation $\sigma = 1$. In this case, we set our posterior function to just this normal distribution. The number of iterations and burn in period remain the same, and the step size is again 2σ, leading to an acceptance rate of $\sim 50\%$. We show the histograms of generated distributions in Figure 3.

We also compute the modes of each of the histograms using the kernel density estimation via R's `density()` function. This function uses a Gaussian kernel to provide a continuous and smooth estimate of a PDF from the samples. Although this can smooth the PDF (e.g. in the fuel estimation case, using `density()` would lead to some density outside of the range (0,182)), here it allows us to give an exact estimate of the mode. The corresponding modes are -0.1313, -0.0023, and 0.0776; we see that at worst, we are about 0.1σ off of the true mode of 0. Thus, if we had used the same method to generate the modes for fuel estimation, we would expect our MCMC predictions of the mode to be within 2 L of the true mode.

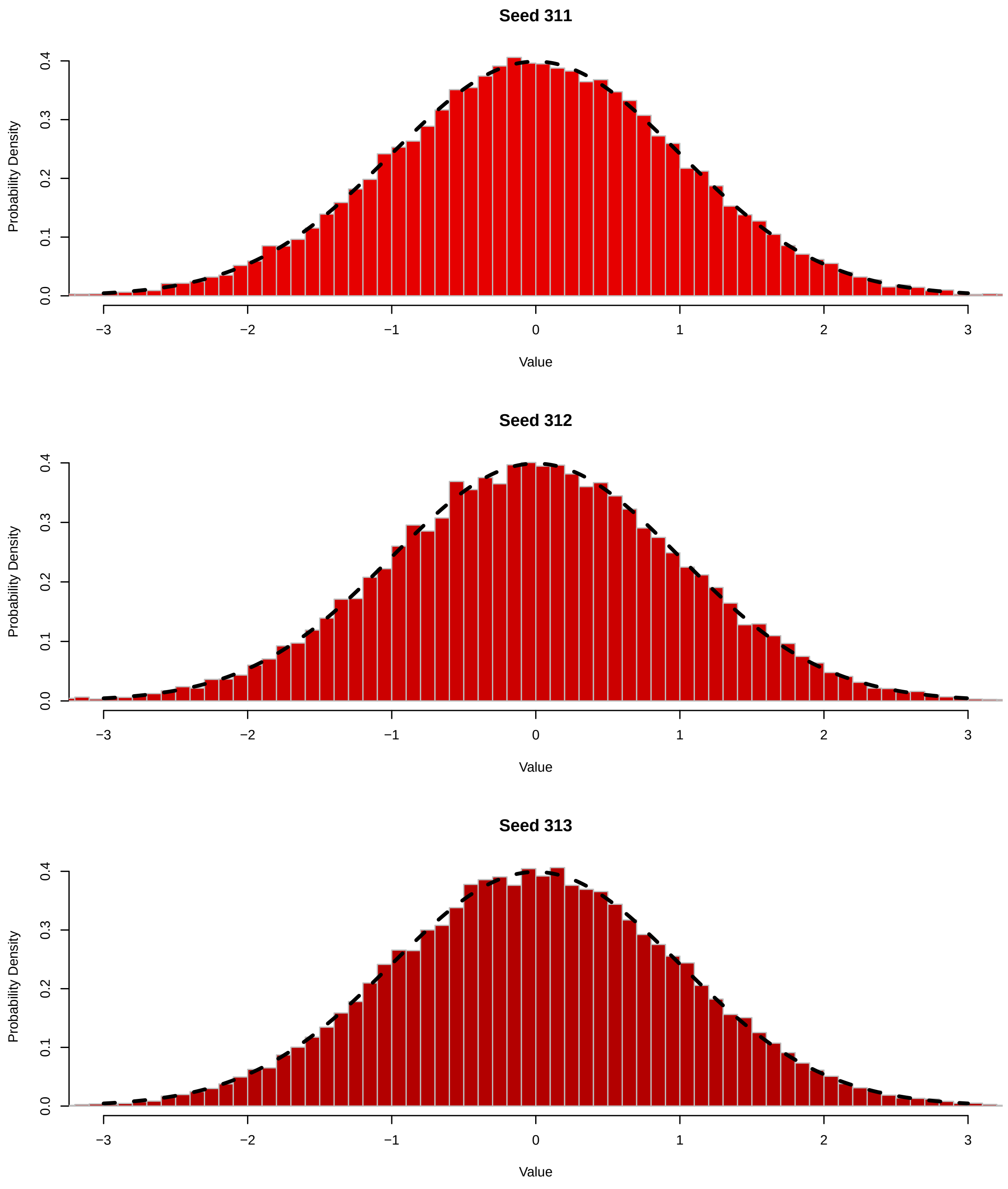

FIGURE 3: The MCMC generated posterior distributions of the positive control for three separate seeds shown by the histograms. Histograms have 100 bins. Overplotted in black is the true posterior distribution.

Metropolis Hasting versus Bayes Monte Carlo Convergence

Here, we see that we can generate posterior distributions using Bayes Monte Carlo with similar accuracy to those show in Figure 2 with just 10^4 iterations, as shown below in Figure 4.

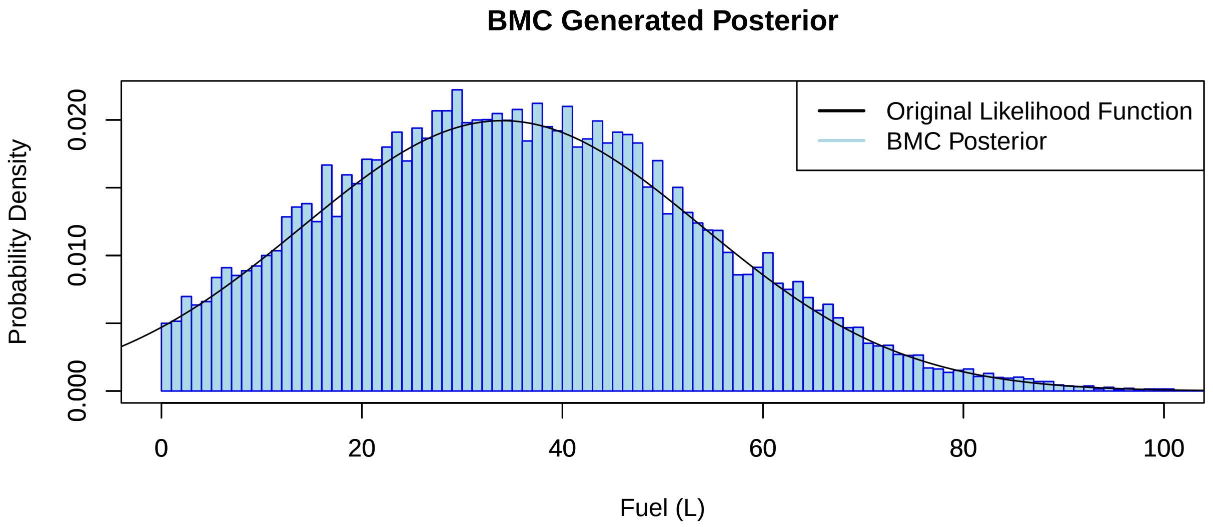

Figure 4: The BMC generated posterior distribution of available fuel with 10^4 iterations.

We see that we need an order of magnitude fewer iterations to reach convergence. This is because BMC does not require burn in, and the samples are not autocorrelated (i.e. they are independent) as in MCMC.

A Code

See `ps3_shields.R` below. The analysis is reproducible, as seeds are set in the code and all figures and output statistics are generated by sourcing it. This file can be found at https://github.com/hughshields/ENG_107.

```
############################################
############################################
##  file: ps4_shields.R
##  ENG 107 Problem Set 4 Solutions
######################################################
##  author: Hugh Shields
##  copyright by the author
##  distributed under the GNU general public license
##  https://www.gnu.org/licenses/gpl.html
##  no warranty (see license details at the link above)
##################################################
# version 1: last changes: Feb. 14, 2025 by Hugh Shields
# contact: nikolaus.h.shields.gr@dartmouth.edu
##################################################
# citations:
# - R help files accessed through R-studio for syntax
# - discussions during ENG 107 class time
# - example codes from ENG 107 Canvas
# - much of the Metropolis-Hastings code was derived from
#   the example code Workflow_Example.R
# - code from my previous Problem Set (#3)
# - truncnorm package documentation
#   (https://cran.r-project.org/web/packages/truncnorm/truncnorm.pdf)
####################################################
# how to run:
# - save the file in a directory
# - go to the directory with this file
# - open R
# - type 'install.packages("truncnorm")'
# - type 'source(ps4_shields.R)'
# - read the printed results in the console and
#   open the new pdf files to analyze the results
###################################################

# Clear any existing variables and plots.
rm(list = ls())
graphics.off()
```

```
## 1a MCMC Posterior

# Define problem parameters
fuel.mean = 34
fuel.sd = 20
fuel.min = 0
fuel.max = 182

# Define the unnormalized log posterior
logp = function(x){

  # Get the log of the likelihood.
  likelihood = dnorm(x, mean = fuel.mean, sd = fuel.sd)
  log.likelihood = log(likelihood)

  # Use uniform prior, and get log
  # recall that a uniform distribution has value 1/(max-min) on [min, max] and
  # 0 elsewhere
  if (x >= fuel.min && x <= fuel.max) {
    prior = (1 / (fuel.max-fuel.min))
    log.prior = log(prior)
  } else {
    prior = 0
    log.prior = -Inf
  }

  # compute unormalized posterior and return value
  log.posterior = log.likelihood + log.prior
  return(log.posterior)
}


# Set the number of MCMC iterations.
NI = 100000

#Perform the analysis for three different seeds
seeds <-c(311,312,313)


chain<-mat.or.vec(NI, length(seeds))

for (s in 1:length(seeds)) {
```

```r
set.seed(seeds[s])


# Start with some initial state of fuel level estimate drawn from prior
x = runif(1, fuel.min, fuel.max)

# Evaluate the unnormalized posterior of the parameter values
# P(theta^initial | y)
lp = logp(x)

# Setup some variables and arrays to keep track of:
x.best = x         # the best parameter estimates
lp.max = lp                # the maximum of the log posterior
x.new = NA      # proposed new parameters (theta^new)
accepts = 0                # how many times the proposed new parameters are accepted
mcmc.chain = array(dim=c(NI,1)) # and a chain of accepted parameters

# Set the step size for the MCMC.
step = fuel.sd*2

# Metropolis-Hastings algorithm MCMC; the proposal distribution proposes the next
# point to which the random walk might move.
for(i in 1:NI) {
  # Propose a new state (theta^new) based on the current parameter values
  # theta and the transition probability / step size
  x.new = rnorm(1, x, sd = step)

  # Evaluate the new unnormalized posterior of the parameter values
  # and compare the proposed value to the current state
  lp.new = logp(x.new)
  lq = lp.new - lp
  # Metropolis test; compute the acceptance ratio
  # Draw some uniformly distributed random number 'lr' from [0,1];
  lr = log(runif(1))

  # If lr < the new proposed value, then accept the parameters setting the
  # proposed new theta (theta^new) to the current state (theta).
  if(lr < lq) {
    # Update the current theta and log posterior to the new state.
    x = x.new
    lp = lp.new
```

```
      # If this proposed new parameter value is \better" (higher posterior probability
      # than the current state, then accept with a probability of 1. Hence, increase
      # the number of acceptations by 1.
      accepts = accepts + 1

      # Check if the current state is the best, thus far and save if so.
      if(lp > lp.max) {
        x.best = x
        lp.max = lp
      }
    }
    # Append the parameter estimate to the chain. This will generate a series of param
    # values (theta_0, theta_1, ...).
    mcmc.chain[i] = x
  }


  #Checking the acceptance rate
  # Calculate the parameter acceptance rate; it should be higher than 23.4%.
  accept.rate <- (accepts/NI) * 100
  print(accept.rate)

  chain[,s] <- mcmc.chain
}

# Define seed colors
colors_red <- mat.or.vec(length(seeds)+1, 3)
for (i in 1:length(seeds)){
  colors_red[i, ] <- c((1-0.1*i), 0, 0)
}

burnin = 10000

# plot chain values for each seed
pdf(file="chains.pdf",8.5,3)
# subplots
par(mfrow = c(1, 3))

# loop over the seeds
for (i in 1:length(seeds)) {
  plot(chain[, i], type = "l", col = rgb(matrix(colors_red[i,], ncol = 3)),
       lwd = 0.1,ylim=c(-10,130), xlab = "Iteration", ylab = "Fuel (L)",
       main = paste("Chain of Seed", seeds[i]))
  abline(v = burnin, lwd = 3,lty = 1,col="black");
}
dev.off()

# plot posteriors for each seed
pdf(file="posteriors.pdf",8.5,10)
par(mfrow = c(3, 1))
# loop over the  seeds
for (i in 1:length(seeds)) {
  # plot histogram for each seed
  hist(chain[burnin:nrow(chain), i], breaks = 100, probability = TRUE,
       xlab = "Fuel (L)", ylab = "Probability Density",
       main = paste("Seed", seeds[i]),
       col = rgb(matrix(colors_red[i,], ncol = 3)), border = "grey",
       xlim = c(-5, 100), ylim = c(0, 0.022))

  # overlay the true distribution
  curve(dtruncnorm(x, mean = fuel.mean, sd = fuel.sd,
                   a = fuel.min, b = fuel.max),
        add = TRUE, col = "black", lwd = 3, lty = 2, n = 2000)
}
par(mfrow = c(1, 1))
dev.off()


## 1b Positive Control

# redefine posterior
logp_control = function(x){

  # Get the log of the likelihood.
  posterior = dnorm(x, mean = 0, sd = 1)
  log.posterior = log(posterior)

  return(log.posterior)
}

# Set the number of MCMC iterations.
NI = 100000

#Perform the analysis for three different seeds
seeds <-c(311,312,313)
```

```
chain_control<-mat.or.vec(NI, length(seeds))

for (s in 1:length(seeds)) {

  set.seed(seeds[s])


  # Start with some initial state
  x = rnorm(1, 0, 1)

  # Evaluate the unnormalized posterior
  lp = logp(x)

  # Setup some variables and arrays to keep track of:
  x.best = x          # the best parameter estimates
  lp.max = lp                 # the maximum of the log posterior
  x.new = NA      # proposed new parameters (theta^new)
  accepts = 0                 # how many times the proposed new parameters are accepted
  mcmc.chain = array(dim=c(NI,1)) # and a chain of accepted parameters

  # Set the step size for the MCMC.
  step = 2

  # Metropolis-Hastings algorithm MCMC; the proposal distribution proposes the next
  # point to which the random walk might move.
  for(i in 1:NI) {
    # Propose a new state (theta^new) based on the current parameter values
    # theta and the transition probability / step size
    x.new = rnorm(1, x, sd = step)

    # Evaluate the new unnormalized posterior of the parameter values
    # and compare the proposed value to the current state
    lp.new = logp_control(x.new)
    lq = lp.new - lp
    # Metropolis test; compute the acceptance ratio
    # Draw some uniformly distributed random number 'lr' from [0,1];
    lr = log(runif(1))

    # If lr < the new proposed value, then accept the parameters setting the
    # proposed new theta (theta^new) to the current state (theta).
    if(lr < lq) {
      # Update the current theta and log posterior to the new state.
      x = x.new
      lp = lp.new

      # If this proposed new parameter value is \better" (higher posterior probability
      # than the current state, then accept with a probability of 1. Hence, increase
      # the number of acceptations by 1.
      accepts = accepts + 1

      # Check if the current state is the best, thus far and save if so.
      if(lp > lp.max) {
        x.best = x
        lp.max = lp
      }
    }
    # Append the parameter estimate to the chain. This will generate a series of param
    # values (theta_0, theta_1, ...).
    mcmc.chain[i] = x
  }


  #Checking the acceptance rate
  # Calculate the parameter acceptance rate; it should be higher than 23.4%.
  accept.rate <- (accepts/NI) * 100
  print(accept.rate)

  chain_control[,s] <- mcmc.chain
}

burnin = 10000


# plot posteriors for each seed
pdf(file="posterior_positivecontrol.pdf",8.5,10)
par(mfrow = c(3, 1))
# loop over the seeds
for (i in 1:3) {
  # plot histogram for each seed
  hist(chain_control[burnin:nrow(chain_control), i], breaks = 100, probability = TRUE,
       xlab = "Value", ylab = "Probability Density",
       main = paste("Seed", seeds[i]),
       col = rgb(matrix(colors_red[i,], ncol = 3)), border = "grey",
       xlim = c(-3,3))
  # overlay the true distribution
  curve(dnorm(x, mean = 0, sd = 1),
        add = TRUE, col = "black", lwd = 3, lty = 2, n = 2000)
}
par(mfrow = c(1, 1))
dev.off()

# estimate modes via kernel density estimation
modes = numeric(length(seeds))
for (i in 1:length(seeds)) {
  # get density estimate
  density_estimate = density(chain_control[burnin:NI, i])
  # find mode in each
  modes[i] = density_estimate$x[which.max(density_estimate$y)]
  # print
  cat("Seed:", seeds[i], "- Mode:", modes[i], "\n")
}

## 1c Bayesian Monte Carlo

# From PS#3: Bayesian Monte Carlo to determine posterior
set.seed(0)
N = 40000
# set up likelihood distribution
fuel = seq(-36, 104, length=10000)
prob = dnorm(fuel, mean=fuel.mean, sd=fuel.sd)

# sample N values from the prior
prior_sample = runif(N, min = fuel.min, max = fuel.max)
# compute the likelihood values from the prior sampling
likelihood_values = dnorm(prior_sample, mean = fuel.mean, sd = fuel.sd)
weights = likelihood_values/sum(likelihood_values)
# resample from the prior sample using the weights (with replacement)
posterior_samples = sample(prior_sample, size = N, replace = TRUE, prob = weights)

# plot histogram (probability=TRUE shows density) of BMC results and overplot original
pdf(file="BMC_posterior.pdf", 8.5,4)
hist(posterior_samples, breaks=100, probability=TRUE, xlab="Fuel (L)",
     ylab = "Probability Density", main = "BMC Generated Posterior",
     col="lightblue", border="blue", xlim=c(0, 100), ylim=c(0, 0.022))
par(new = TRUE)
plot(fuel, prob, type="l", lwd=1, xlab="", ylab = "", main = "",
     xlim = c(0, 100), ylim=c(0, 0.022))
```

```
legend("topright", legend = c("Original Likelihood Function", "BMC Posterior"), col = c(
dev.off()
```