Part 1: Technical Details

Metropolis-Hastings MCMC Sampling Model:

```r
set.seed(1234)
library(MCMCpack)
```

```
## Warning: package 'MCMCpack' was built under R version 4.3.3

## Loading required package: coda

## Loading required package: MASS

## Warning in .recacheSubclasses(def@className, def, env): undefined subclass
## "ndiMatrix" of class "replValueSp"; definition not updated

## ##
## ## Markov Chain Monte Carlo Package (MCMCpack)

## ## Copyright (C) 2003-2024 Andrew D. Martin, Kevin M. Quinn, and Jong Hee Park

## ##
## ## Support provided by the U.S. National Science Foundation

## ## (Grants SES-0350646 and SES-0350613)
## ##
```

```r
library(tinytex)

# Given Data
y = c(12.1, 4.7, 8.6, 2.3, 14.7, 4.4, 6.8, 10.2, 17.8, 3.5)
n = c(14, 16, 7, 10, 33, 21, 14, 13, 18, 10)
J = length(y)

# Unnormalized Posterior
p = function(a, b, y, n) {
    J = length(n)
    dens = (b^a/gamma(a))^J * prod(gamma(n + a)/(sum(y) + b)^(n + a))
    return(dens)
}

# Proposal Distribution
library(TruncatedNormal)
```

```
## Warning: package 'TruncatedNormal' was built under R version 4.3.3
```

```r
Q = function(mean.a, mean.b, prop.var) {
    rtmvnorm(1, mu = c(mean.a, mean.b), sigma = prop.var, lb = c(0, 0))
}

qdens = function(a, b, mean.a, mean.b, prop.var) {
    dtmvnorm(c(a, b), mu = c(mean.a, mean.b), sigma = prop.var, lb = c(0, 0))
}

# Actual 2-Dimensional Metropolis-Hastings Algorithm
mh2D = function(param1, param2, y, prop.var, n.iter) {
    results = matrix(nrow = n.iter, ncol = 3)
    colnames(results) = c("param1", "param2", "accept")

    # initialize model
    param1.t = param1
    param2.t = param2

    # draw proposed alpha and beta
    for (t in 1:n.iter) {
        new = Q(param1.t, param2.t, prop.var)
        param1.new = new[1]
        param2.new = new[2]

        # acceptance probability calculation
        alpha = min(1, p(param1.new, param2.new, y, n)/p(param1.t, param2.t, y, n) *
            (qdens(param1.t, param2.t, param1.new, param2.new, prop.var)/qdens(param1.ne
                param2.new, param1.t, param2.t, prop.var)))

        # accept/reject decision
        u = runif(1, 0, 1)

        # implementation of accepting/rejecting
        if (u <= alpha) {
            param1.t = param1.new
            param2.t = param2.new
        } else {
            param1.t = param1.t
            param2.t = param2.t
        }
        # store results for alpha and beta
        results[t, "param1"] = param1.t
        results[t, "param2"] = param2.t
        results[t, "accept"] = ifelse(u <= alpha, T, F)
    }
    return(results)
}
```

Run algorithm to get results for α and β:

```r
# Run Algorithm
n.iter = 20000
result1 = mh2D(1, 1, y, prop.var = diag(4, 2), n.iter = n.iter)  #chain1
result2 = mh2D(5, 10, y, prop.var = diag(4, 2), n.iter = n.iter)  #chain2
result3 = mh2D(10, 5, y, prop.var = diag(4, 2), n.iter = n.iter)  #chain3
result4 = mh2D(18, 10, y, prop.var = diag(4, 2), n.iter = n.iter)  #chain4
result5 = mh2D(18, 18, y, prop.var = diag(4, 2), n.iter = n.iter)  #chain5

alpha = c(result1[, "param1"], result2[, "param1"], result3[, "param1"], result4[,
    "param1"], result5[, "param1"])
bbeta = c(result1[, "param2"], result2[, "param2"], result3[, "param2"], result4[,
    "param2"], result5[, "param1"])
```

Use samples of α and β to draw θ_j:

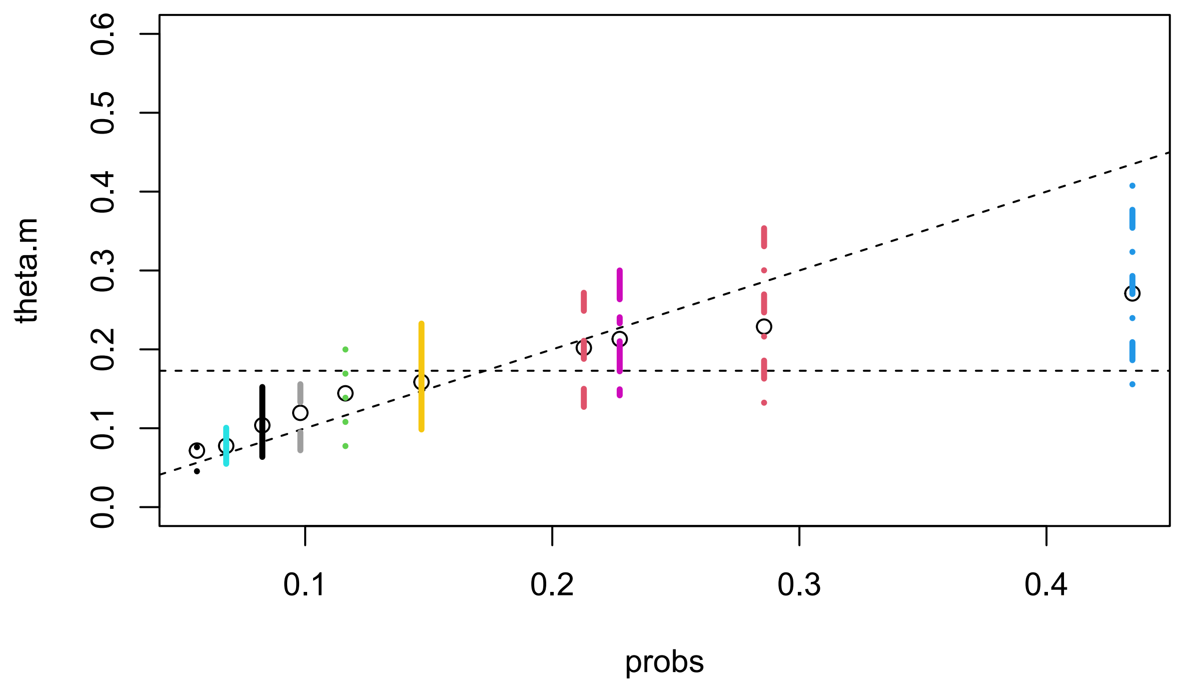
```r
# Sample Thetas
theta = matrix(NA, nrow = n.iter, ncol = J)
for (j in 1:J) {
    theta[, j] = rgamma(n.iter, n[j] + alpha, n[j] * sum(y[j]) + bbeta)
}
wt = 1/theta  #waiting time estimates
```

Summary Statistics for θ_j

```r
# Calculate Summary Statistics for Thetas
theta.m = apply(theta, 2, mean)  #mean for each theta
theta.l = apply(theta, 2, quantile, 0.025)  #95% lower bound for each theta
theta.h = apply(theta, 2, quantile, 0.975)  #95% upper bound for each theta
results = matrix(NA, J + 2, 3)
results[, 1] = c(theta.m, mean(alpha), mean(bbeta))  #store in results matrix
results[, 2] = c(theta.l, quantile(alpha, 0.025), quantile(bbeta, 0.025))
results[, 3] = c(theta.h, quantile(alpha, 0.975), quantile(bbeta, 0.975))
colnames(results) = c("Mean", "Lower Bound", "Upper Bound")
rownames(results) = c("NYC", "Boston", "Baltimore", "Charlotte", "Miami", "Denver",
    "Seattle", "San Diego", "Los Angeles", "Las Vegas", "Alpha", "Beta")
library(knitr)
kable(results)
```

	Mean	Lower Bound	Upper Bound
NYC	0.1037288	0.0636589	0.1522686
Boston	0.2019569	0.1271004	0.2935485
Baltimore	0.1442891	0.0773483	0.2283302
Charlotte	0.2709363	0.1556982	0.4340168
Miami	0.0776313	0.0548870	0.1038918
Denver	0.2130853	0.1417138	0.3000404
Seattle	0.1585039	0.0984476	0.2326756
San Diego	0.1195413	0.0720559	0.1774228
Los Angeles	0.0716628	0.0453609	0.1031637
Las Vegas	0.2288195	0.1323069	0.3554275
Alpha	8.1775831	3.1684232	12.0745310
Beta	37.6121916	4.8234941	68.5876358

```r
# Summary Plot for Thetas
probs = 1/y
plot(probs, theta.m, ylim = c(0, 0.6))
abline(a = 0, b = 1, lty = 2)
abline(h = mean(probs), lty = 2)
for (j in 1:J) {
    segments(probs[j], theta.l[j], probs[j], theta.h[j], col = j, lty = j, lwd = 3)
}
```



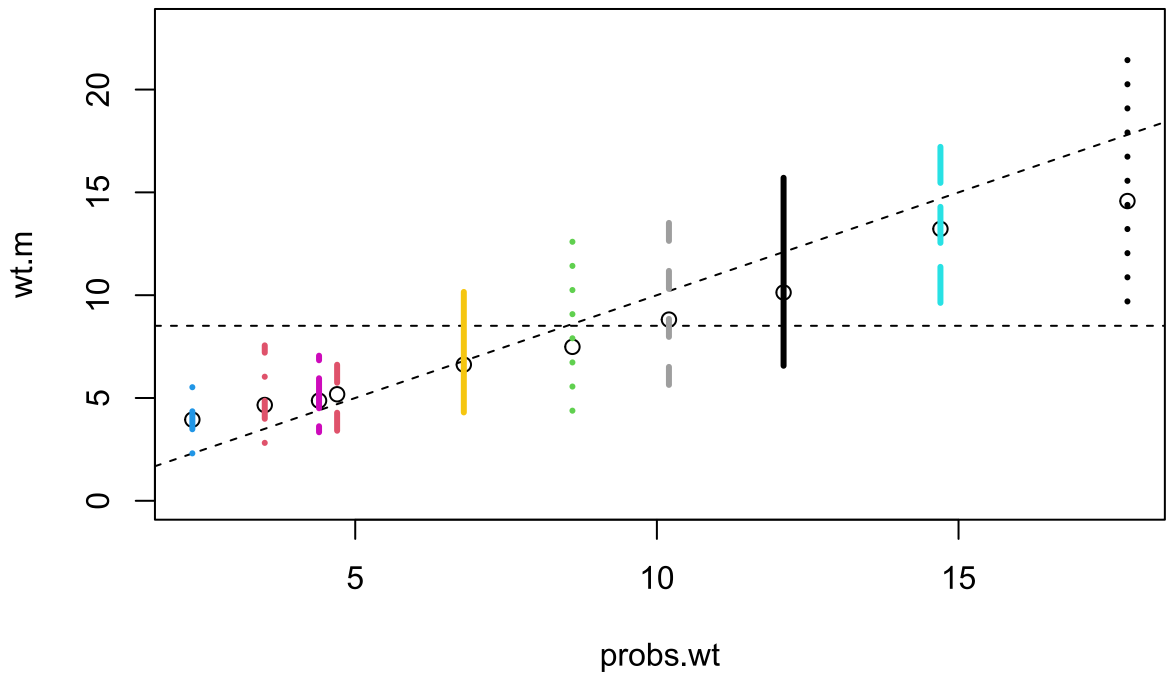
Summary Statistics for Waiting Times:

```
# Calculate Summary Statistics for Waiting Times
wt.m = apply(wt, 2, mean)
wt.l = apply(wt, 2, quantile, 0.025)
wt.h = apply(wt, 2, quantile, 0.975)
results.wt = matrix(NA, J + 2, 3)
results.wt[, 1] = c(wt.m, mean(alpha), mean(bbeta))
results.wt[, 2] = c(wt.l, quantile(alpha, 0.025), quantile(bbeta, 0.025))
results.wt[, 3] = c(wt.h, quantile(alpha, 0.975), quantile(bbeta, 0.975))
colnames(results.wt) = c("Mean", "Lower Bound", "Upper Bound")
rownames(results.wt) = c("NYC", "Boston", "Baltimore", "Charlotte", "Miami", "Denver",
    "Seattle", "San Diego", "Los Angeles", "Las Vegas", "Alpha", "Beta")
library(knitr)
kable(results.wt)
```

	Mean	Lower Bound	Upper Bound
NYC	10.130330	6.567344	15.708734
Boston	5.180440	3.406592	7.867796
Baltimore	7.486931	4.379622	12.928530

	Mean	Lower Bound	Upper Bound
Charlotte	3.946779	2.304059	6.422683
Miami	13.223403	9.625401	18.219235
Denver	4.870532	3.332885	7.056476
Seattle	6.623830	4.297828	10.157688
San Diego	8.812997	5.636255	13.878108
Los Angeles	14.578192	9.693336	22.045432
Las Vegas	4.657850	2.813513	7.558182
Alpha	8.177583	3.168423	12.074531
Beta	37.612192	4.823494	68.587636

```
# Summary Plot
probs.wt = y
plot(probs.wt, wt.m, ylim = c(0, 23))
abline(a = 0, b = 1, lty = 2)
abline(h = mean(probs.wt), lty = 2)
for (j in 1:J) {
    segments(probs.wt[j], wt.l[j], probs.wt[j], wt.h[j], col = j, lty = j, lwd = 3)
}
```



Assessing Convergence:

```r
# Acceptance Probabilities
resultall = rbind(result1, result2, result3, result4, result5)
mean(result1[, "accept"])
```

```
## [1] 0.4861
```

```r
mean(result2[, "accept"])
```

```
## [1] 0.4825
```

```r
mean(result3[, "accept"])
```

```
## [1] 0.47815
```

```r
mean(result4[, "accept"])
```

```
## [1] 0.45825
```

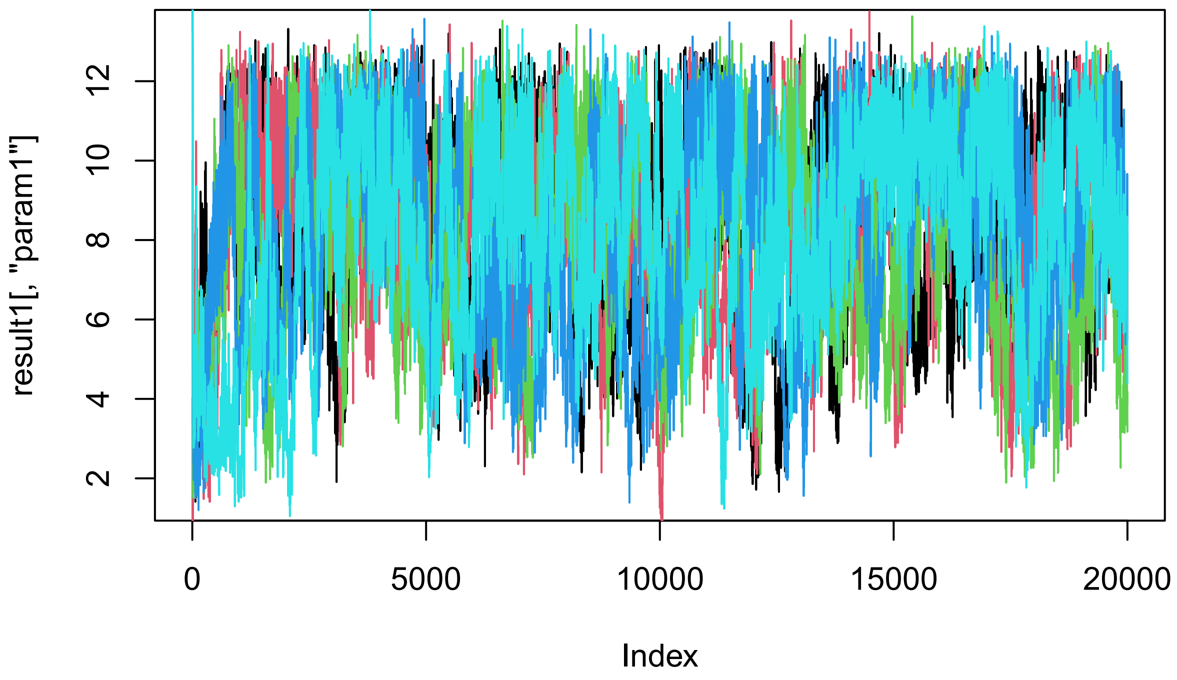
```r
mean(result5[, "accept"])
```

```
## [1] 0.47245
```

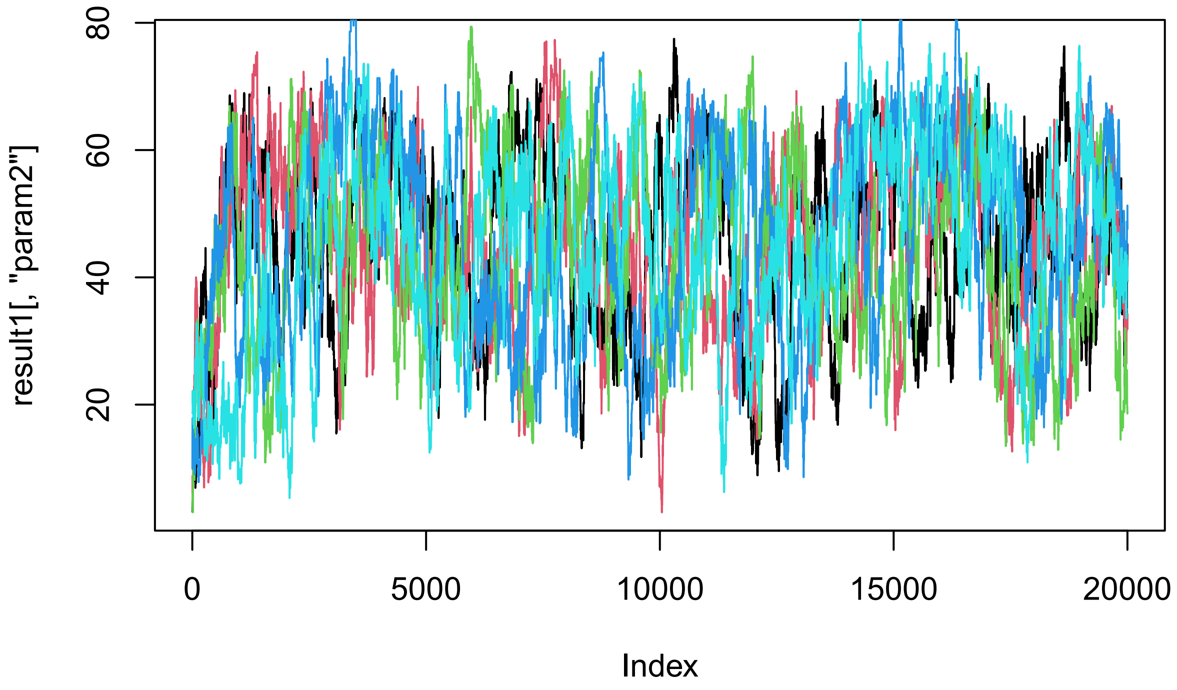
```r
mean(resultall[, "accept"])
```

```
## [1] 0.47549
```

```r
# Trace Plot for Alpha
plot(result1[, "param1"], type = "l")
lines(result2[, "param1"], col = 2)
lines(result3[, "param1"], col = 3)
lines(result4[, "param1"], col = 4)
lines(result5[, "param1"], col = 5)
```

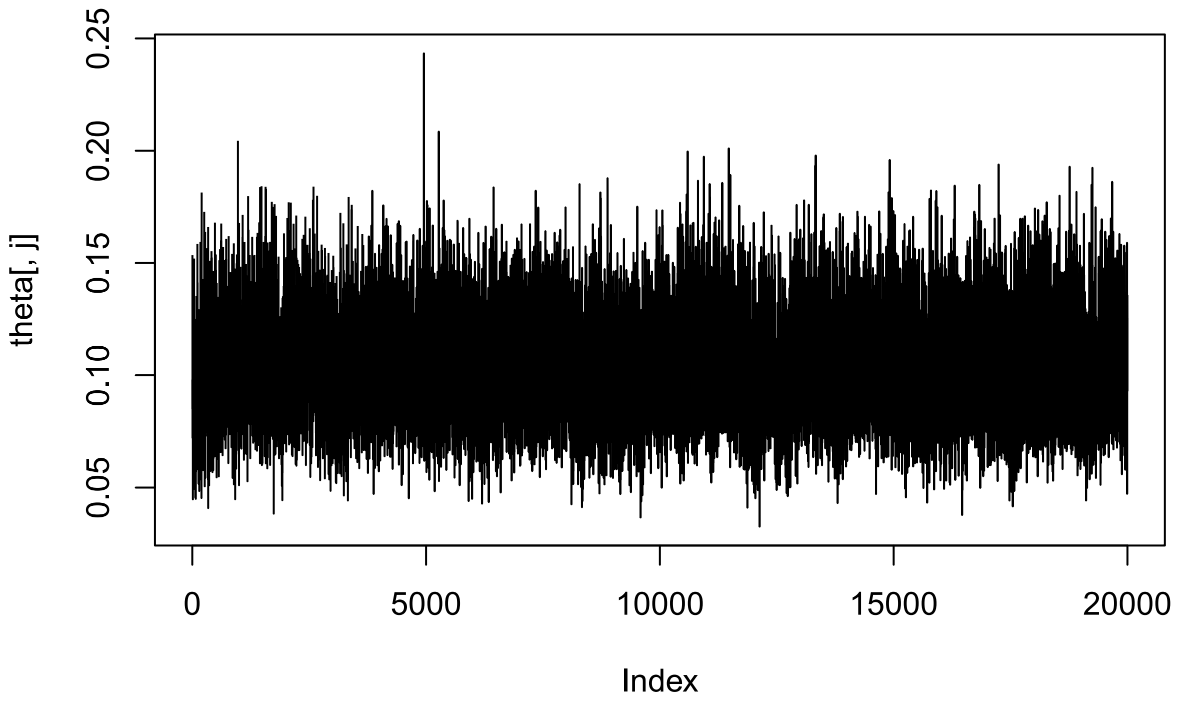


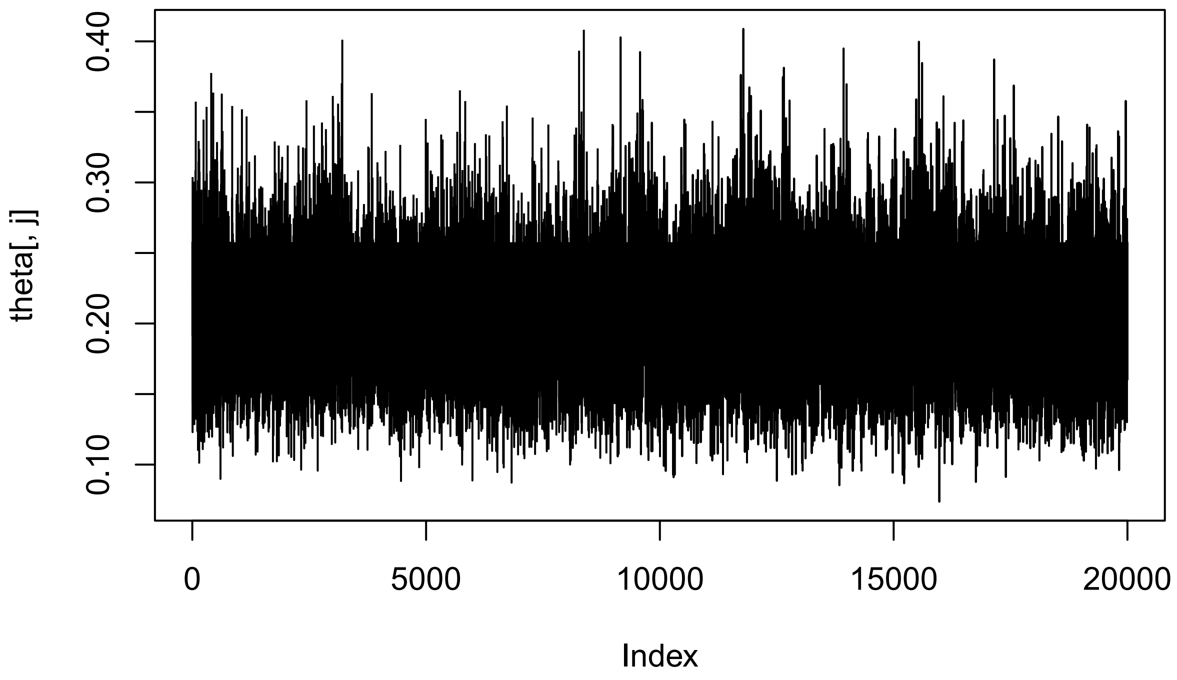
```r
# Trace Plot for Beta
plot(result1[, "param2"], type = "l")
lines(result2[, "param2"], col = 2)
lines(result3[, "param2"], col = 3)
lines(result4[, "param2"], col = 4)
lines(result5[, "param2"], col = 5)
```

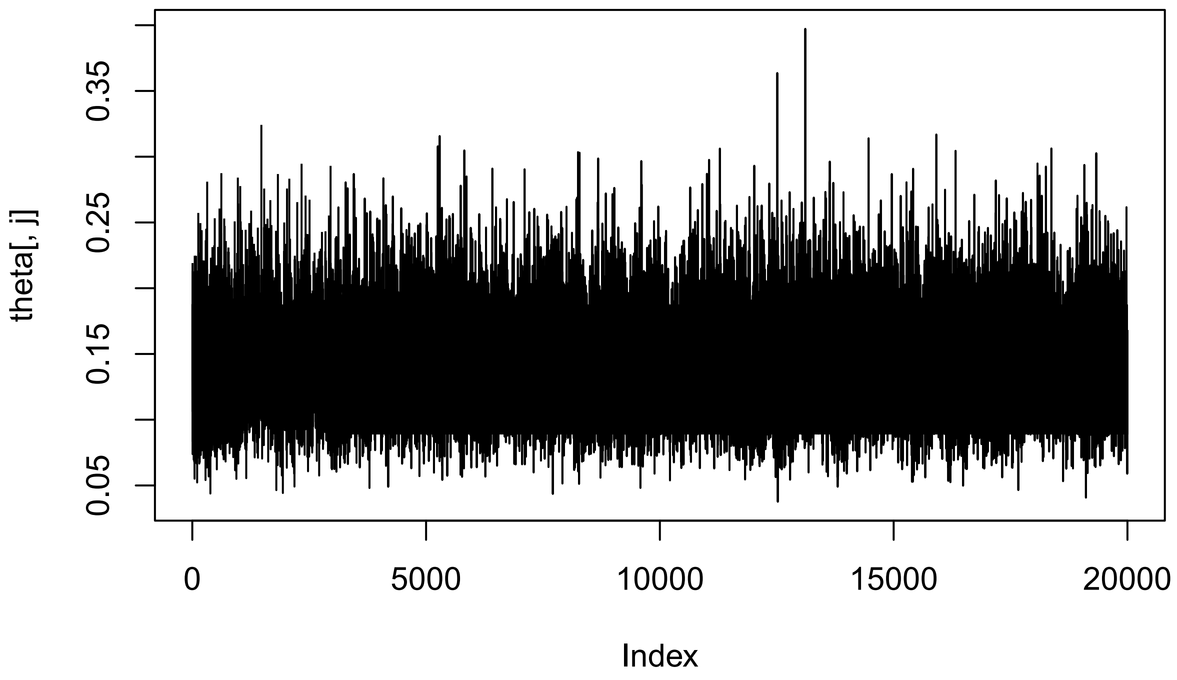


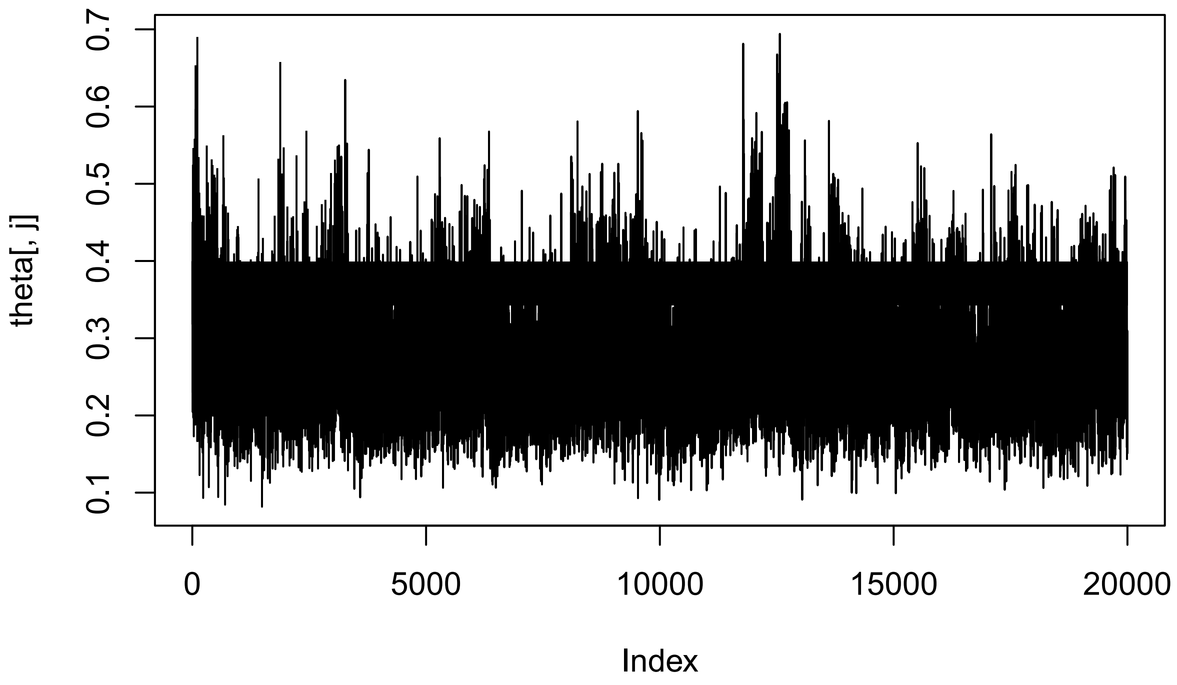
```r
# Chain 1:
a = result1[, "param1"]
b = result1[, "param2"]
theta = matrix(NA, n.iter, J)
for (j in 1:J) {
    theta[, j] = rgamma(n.iter, n[j] + alpha, n[j] * sum(y[j]) + bbeta)
}
chain1 = cbind(theta, a, b)

# Chain 2:
a = result2[, "param1"]
b = result2[, "param2"]
theta = matrix(NA, n.iter, J)
for (j in 1:J) {
    theta[, j] = rgamma(n.iter, n[j] + alpha, n[j] * sum(y[j]) + bbeta)
}
chain2 = cbind(theta, a, b)

# Chain 3:
a = result3[, "param1"]
b = result3[, "param2"]
```

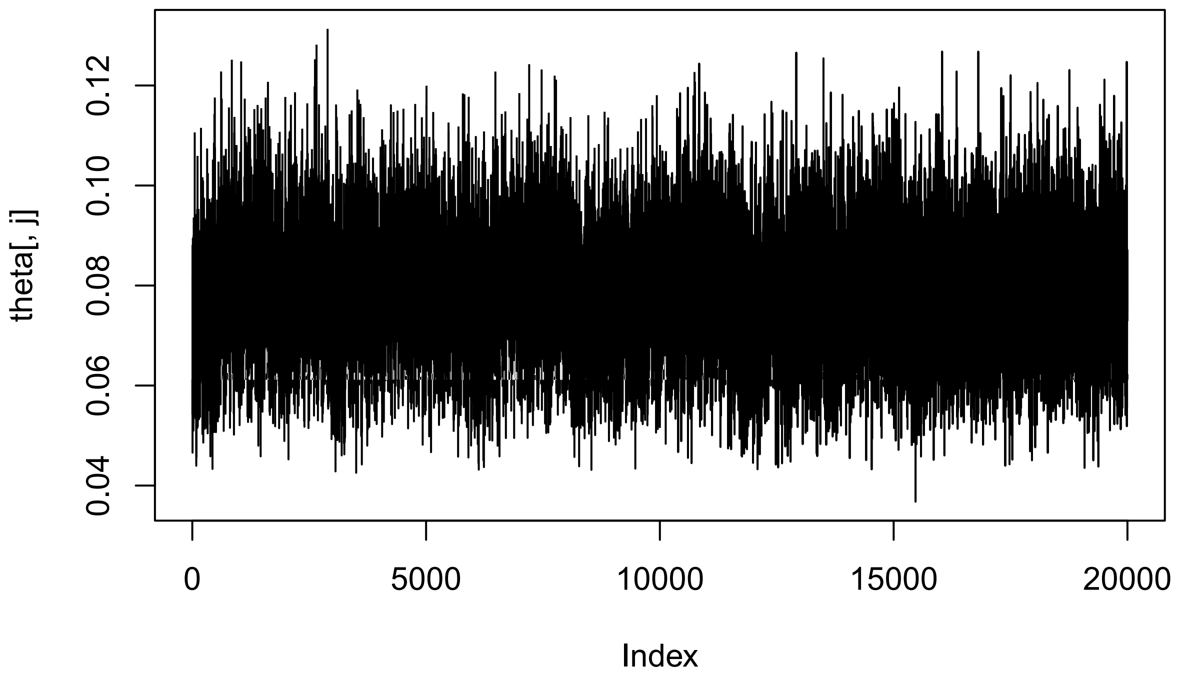
```r
theta = matrix(NA, n.iter, J)
for (j in 1:J) {
    theta[, j] = rgamma(n.iter, n[j] + alpha, n[j] * sum(y[j]) + bbeta)
}
chain3 = cbind(theta, a, b)

# Chain 4:
a = result4[, "param1"]
b = result4[, "param2"]
theta = matrix(NA, n.iter, J)
for (j in 1:J) {
    theta[, j] = rgamma(n.iter, n[j] + alpha, n[j] * sum(y[j]) + bbeta)
}
chain4 = cbind(theta, a, b)

# Chain 5
a = result5[, "param1"]
b = result5[, "param2"]
theta = matrix(NA, n.iter, J)
for (j in 1:J) {
    theta[, j] = rgamma(n.iter, n[j] + alpha, n[j] * sum(y[j]) + bbeta)
}
chain5 = cbind(theta, a, b)

for (j in 1:J) {
    plot(theta[, j], type = "l")
}
```


theta[, j]
0.05
0.10
0.15
0.20
0.25
0
5000
10000
15000
20000
Index

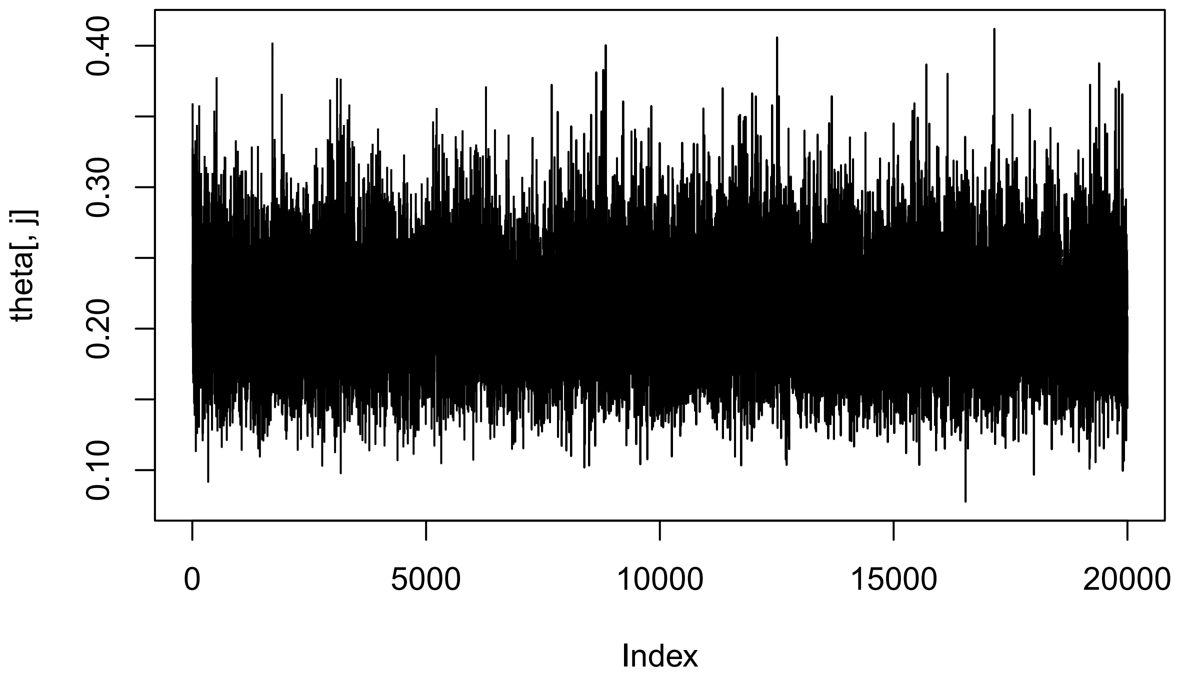

theta[, j]
0.10
0.20
0.30
0.40
0
5000
10000
15000
20000
Index

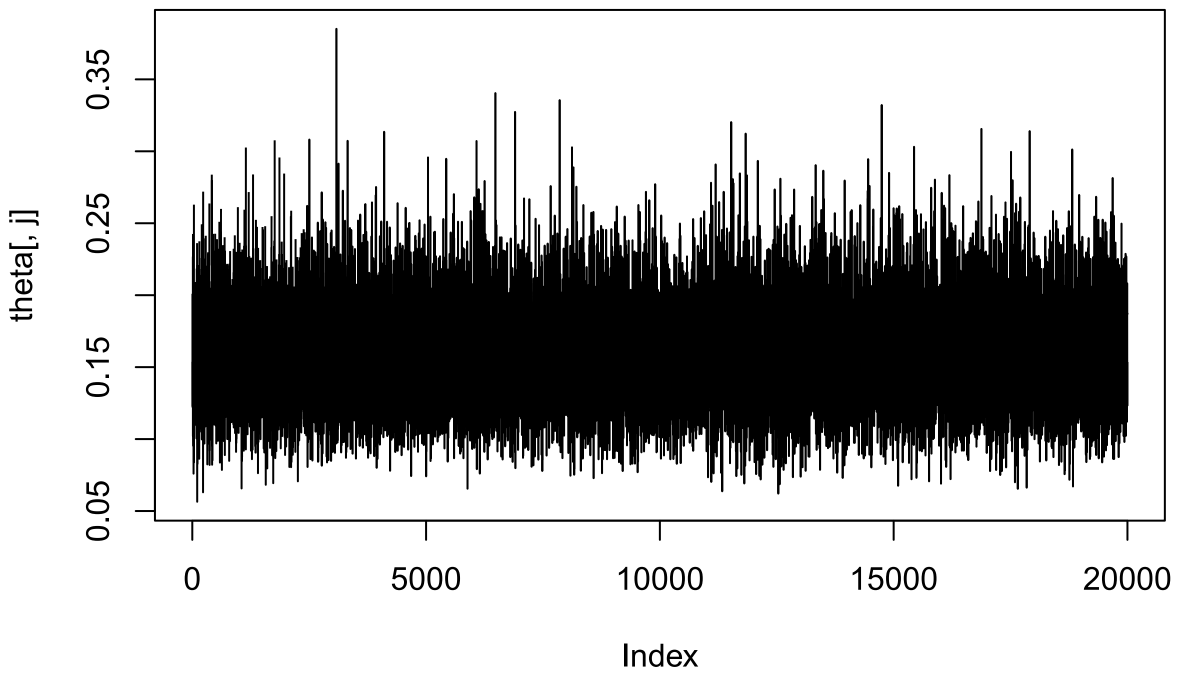

theta[, j]
0.05
0.15
0.25
0.35
0
5000
10000
15000
20000
Index

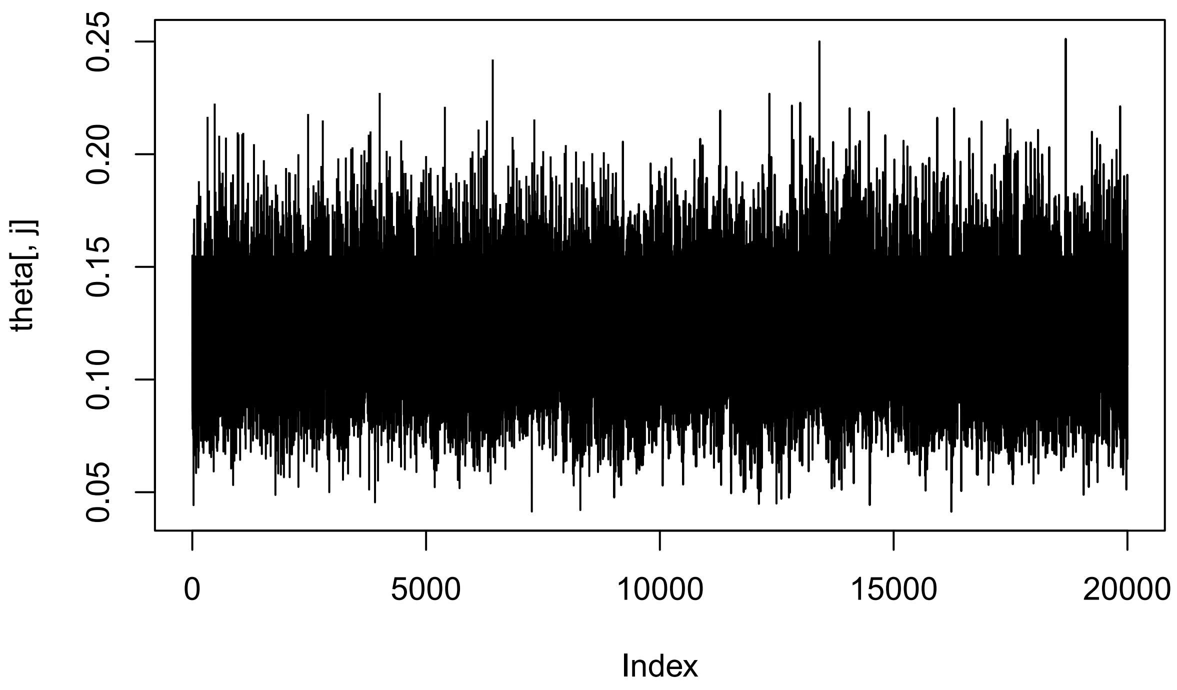

theta[, j]
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
5000
10000
15000
20000
Index



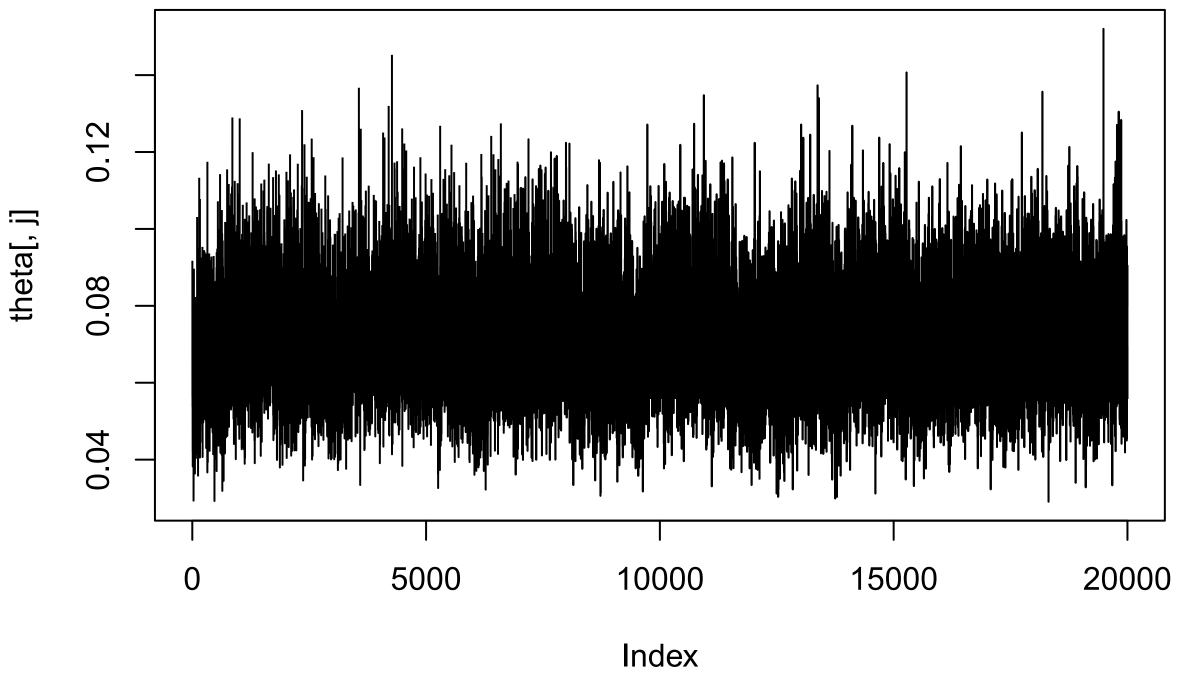
theta[, j]
0.04
0.06
0.08
0.10
0.12
0
5000
10000
15000
20000
Index

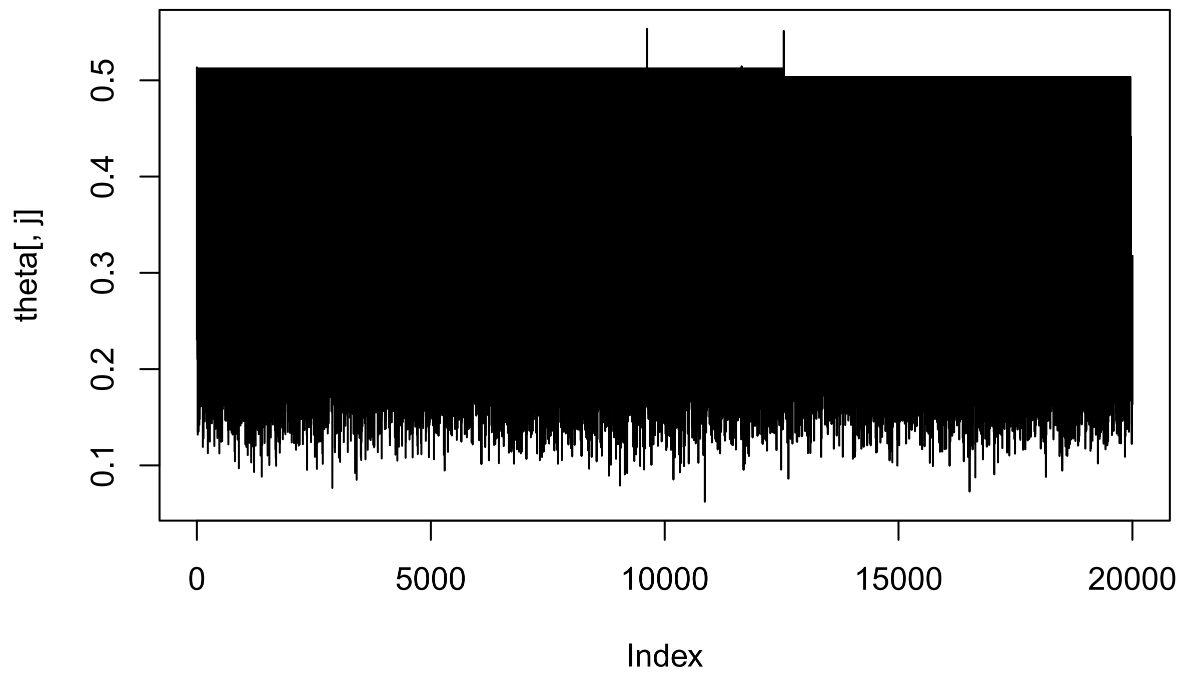

theta[, j]
0.10
0.20
0.30
0.40
0
5000
10000
15000
20000
Index


theta[, j]
0.05
0.15
0.25
0.35
0
5000
10000
15000
20000
Index



theta[, j]
0.05
0.10
0.15
0.20
0.25
0
5000
10000
15000
20000
Index


theta[, j]
0.04
0.08
0.12
0
5000
10000
15000
20000
Index



```r
# Gelman-Rubin Diagnostic
mcmc.all = mcmc.list(as.mcmc(chain1), as.mcmc(chain2), as.mcmc(chain3), as.mcmc(chain4)
    as.mcmc(chain5))
gelman.diag(mcmc.all)
```

```
## Potential scale reduction factors:
##
##   Point est. Upper C.I.
##         1.00       1.00
##         1.00       1.00
##         1.00       1.00
##         1.00       1.00
##         1.00       1.00
##         1.00       1.00
##         1.00       1.00
##         1.00       1.00
##         1.00       1.00
##         1.00       1.00
## a       1.02       1.04
## b       1.02       1.06
##
```

```
## Multivariate psrf
##
## 1.02
```

```r
# Raftery-Lewis Diagnostic
raftery.diag(resultall[, c("param1", "param2")], r = 0.0125)
```

```
##
## Quantile (q) = 0.025
## Accuracy (r) = +/- 0.0125
## Probability (s) = 0.95
##
##         Burn-in  Total Lower bound  Dependence
##         (M)      (N)   (Nmin)       factor (I)
##  param1 252      42168 600          70.3
##  param2 325      53375 600          89.0
```

```r
# Effective Sample Size
effectiveSize(mcmc.all)
```

```
##
## 20959.3513 49847.1726 56549.2657  3780.7174 19072.4910 40389.6667 80561.4211
##                                           a           b
## 34542.9618 10689.9815 13281.5950   310.3171   228.7063
```

Technical Write-Up:

To draw inference on each θ_j in this hierarchical model, we must first draw samples of our hyperparameters α and β. For this, we must use an MCMC sampling method, since there is no recognizable distribution that aligns with the joint posterior $P(\alpha, \beta|y)$. To implement this method, we first put in our known data of waiting times and sample sizes for each city. Then we create a function to calculate our unnormalized posterior density from the joint posterior $P(\alpha, \beta|y)$ from our exam. For the proposal distribution to propose a new α and β, I used the 2-dimensional normal distribution, with bounds $(0, \infty)$, means centered at the means of α and β and the variance equal to diag(4,2). This distribution was from the notes and seemed like a reasonable proposal for an MCMC sampler with 2 parameters. Next, I created a qdens function to calculate the height of the proposal point for each α and β and to help calculate our acceptance probability. For this, I also used the bivariate normal distribution from our notes, with the same center and variance as before. The next function is the actual MCMC sampler, which takes the proposal point, calculates an acceptance probability that is compared to a random uniform distribution value, decides whether to accept or reject this point and stores this all in a results matrix. Once the functions were complete, I ran the sampler for 5 different chains of 20000 iterations each, for a total of 100000 iterations of α and β. The mean for α was 8.1775831 and the mean for β was 37.612192. I then combined the 5 results into α and β values. Next, I sampled each θ_j using the α and β we drew from our MCMC sampler, and combined the data we had observed, such as the mean wait times and sample sizes for each group. For this, I used a gamma distribution that was also found in our exam, with parameters n[j] + α and n[j] * sum(y[j]) + beta, which is equivalent to the parameters found on our exam, allowing us to both share information between groups and take each group mean into account. Since θ_j are our parameters of interest, I don't have to sample anymore. Instead, I started to summarize the α, β, and θ_j. First, I put α, β, and each θ_j into a nice table showing their means and 95% posterior intervals. Next, I plotted each θ_j with their mean and posterior interval on a plot, showing how they relate the both the overall grand mean (8.51) and data means. This is a great way of visually showing how information is both shared between the groups and pulled from our data means in calculating each θ_j. Since θ_j aren't the actual waiting times, I decided to do summaries on waiting times ($1/\theta_j$) as well. I did similar summaries for waiting times, with a table showing the mean waiting time and 95% posterior interval for each city based on our samples. I also made a plot of waiting times, exactly like the plot for each θ_j. This is another good representation of how the waiting times are based on both the overall mean, by sharing information between groups, and the individual means. After summarizing, I started to assess the convergence of our samples to make sure our results are reliable. First, I looked at the acceptance probabilities for each chain and all chains together. Each chain had a similar acceptance probability, with the total acceptance probability being about 0.475 For a 2-dimensional sampler, we would like this to be closer to 0.25 and may want to increase our proposal variance, but this is not too important. Next, I looked at the trace plots for α, β, and each θ_j. The trace plot for all chains of α looked extremely good for a hyperparameter and had a bushy hedge appearance. The trace plot for β showed a slight pattern but still looked great for a hyperparameter. The trace plots for each θ_j also looked extremely good with a bushy hedge appearance, showing evidence of convergence and full space exploration for α, β, and each θ_j. I then looked at the Gelman-Rubin diagnostic for this MCMC model. α, β, and each θ_j all had psrf below well below 1.1 and the multivariate psrf was 1.02, which is well below 1.1, showing evidence of convergence. Next, I looked at the Raftery-Lewis diagnostic with all my chains combined into one. The recommended iterations from this diagnostic were 42168 for α and 53375 for β. Since I had a total of 100000 iterations for both α and β this shows evidence of convergence. The last thing I checked was the effective sample sizes we had for α, β, and each θ_j. Our effective sample sizes were relatively high for each θ_j. For α, β they were lower, but still enough for convergence. Overall, there is strong evidence of convergence for α, β, and each θ_j, showing our results are reliable.

Part 2: Non-Technical Findings

In my analysis, I found that the overall average waiting time across all cities is expected to be about 8 minutes from our samples. However, there is quite a large range across the different cities. The lowest expected waiting time was around 4 minutes in Charlotte and the highest expected waiting time was around 14.5 minutes in Los Angeles. The other expected waiting times were about 10 minutes for New York City, about 5 minutes for Boston, about 7.5 minutes for Baltimore, about 13 minutes for Miami, about 5 minutes for Denver, about 6.5 minutes for Seattle, about 9 minutes for San Diego, and about 4.5 minutes for Las Vegas. These exact waiting times can be found in the waiting time table above, but depend slightly on the sample draws. The vast majority of wait times should not exceed the upper bound in the respective city, meaning almost no expected wait times should exceed 22 minutes. One trend seen is that higher expected waiting times are correlated with less waiting time certainty. This could mean that the areas with the highest waiting times have less consistent bus schedules and could use reform from the Department of Public Transportation. The lower expected waiting times are more certain, and any city with an expected waiting time of under 10 minutes seems to have reasonable transportation infrastructure. The main cities that are areas of concern for reforms are New York City, Miami, and Los Angeles.