Assignment 3

Zhengyang Fei

```
library(R2jags)
```

```
Warning: package 'R2jags' was built under R version 4.4.3
```

```
Loading required package: rjags
```

```
Warning: package 'rjags' was built under R version 4.4.3
```

```
Loading required package: coda
```

```
Warning: package 'coda' was built under R version 4.4.2
```

```
Linked to JAGS 4.3.1
```

```
Loaded modules: basemod,bugs
```

```

Attaching package: 'R2jags'
```

```
The following object is masked from 'package:coda':

    traceplot
```

```
library(coda)
```

Question 1

Part a

We run 20,000 iterations of the MCMC for the three parameters: “sd.b”, “b.smok”, and “b.ob”, using three independent chains. Each chain is displayed in a distinct color in the trace plots. Additionally, we provide the autocorrelation plots to evaluate the dependence structure within the samples and summarize key descriptive statistics for each parameter.

```
cat(
"smoke obese snore male hypoten n
0 0 0 1 5 60
0 0 0 0 10 149
1 0 0 1 2 17
1 0 0 0 6 16
0 1 0 1 1 12
0 1 0 0 2 9
0 0 1 1 36 187
0 0 1 0 28 138
1 0 1 1 13 85
1 0 1 0 4 39
0 1 1 1 15 51
0 1 1 0 11 28
1 1 1 1 8 23
1 1 1 0 4 12
", file= "SmokeHyperData.txt")


data <- read.table("SmokeHyperData.txt", header=TRUE, sep="")

model_string <- "
model {
  for( i in 1:14){
    hypoten[i] ~ dbin(mu[i], n[i])
    logit(mu[i]) <- b0 + b.smok * smoke[i] + b.ob * obese[i] + b.sn * snore[i] +
                    b.male * male[i] + b.smsn * smoke[i] * snore[i] + b[i]
    b[i] ~ dnorm(0, tau.b)
  }
  b0 ~ dnorm(0, 0.04)
  b.smok ~ dnorm(0, 0.04)
  b.ob ~ dnorm(0, 0.04)
  b.sn ~ dnorm(0, 0.04)
```

```
  b.male ~ dnorm(0, 0.04)
  b.smsn ~ dnorm(0, 0.04)
  sd.b ~ dunif(0, 5)
  tau.b <- 1 / (sd.b * sd.b)
}
"

writeLines(model_string, "SmokeHyperModel.txt")

data_list <- list(
  hypoten = data$hypoten,
  n = data$n,
  smoke = data$smoke,
  obese = data$obese,
  snore = data$snore,
  male = data$male
)

init_list <- function() {
  list(
    b = runif(14, -0.8, -0.2),    # Random initialization for b[i]
    b0 = runif(1, -0.8, -0.2),    # Intercept
    b.smok = runif(1, -0.8, -0.2),# Smoking coefficient
    b.ob = runif(1, -0.8, -0.2),  # Obesity coefficient
    b.sn = runif(1, -0.8, -0.2),  # Snoring coefficient
    b.male = runif(1, -0.8, -0.2),# Gender coefficient
    b.smsn = runif(1, -8, -0.2),  # Interaction term coefficient (smoking & snoring)
    sd.b = runif(1, 0.2, 0.8)      # Standard deviation of random effect
  )
}

params <- c("sd.b", "b.smok", "b.ob")

jags_fit <- jags(
  data = data_list,
  inits = init_list,
  parameters.to.save = params,
  model.file = "SmokeHyperModel.txt",
  n.chains = 3,
  n.iter = 20000, # 20,000 iterations
  n.burnin = 1,
  n.thin = 1,
```

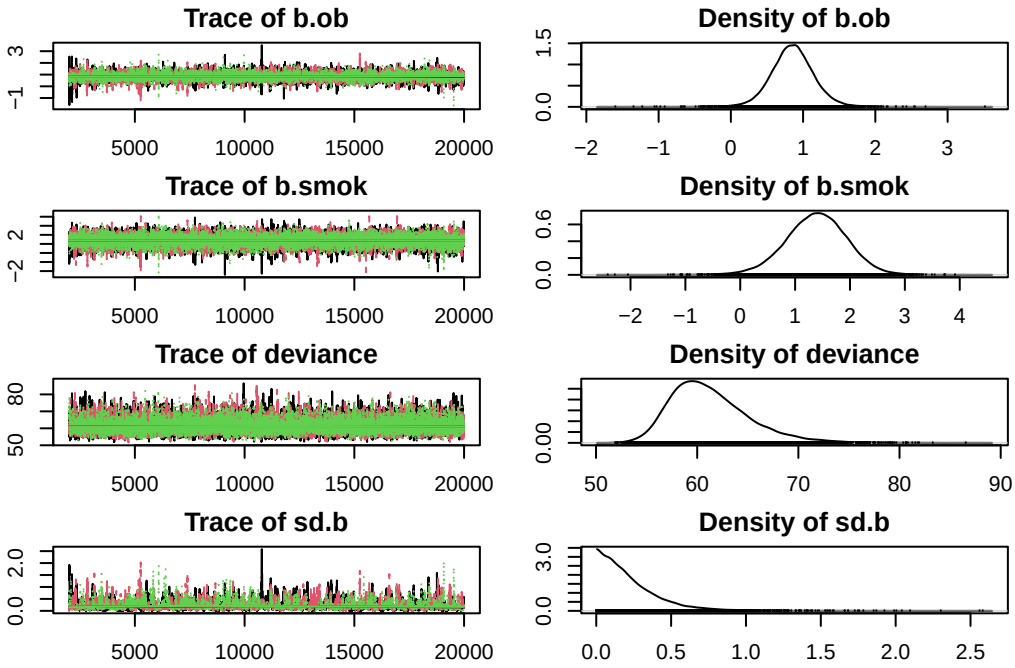
```
  DIC = TRUE
)
```

```
module glm loaded

Compiling model graph
   Resolving undeclared variables
   Allocating nodes
Graph information:
   Observed stochastic nodes: 14
   Unobserved stochastic nodes: 21
   Total graph size: 151

Initializing model
```

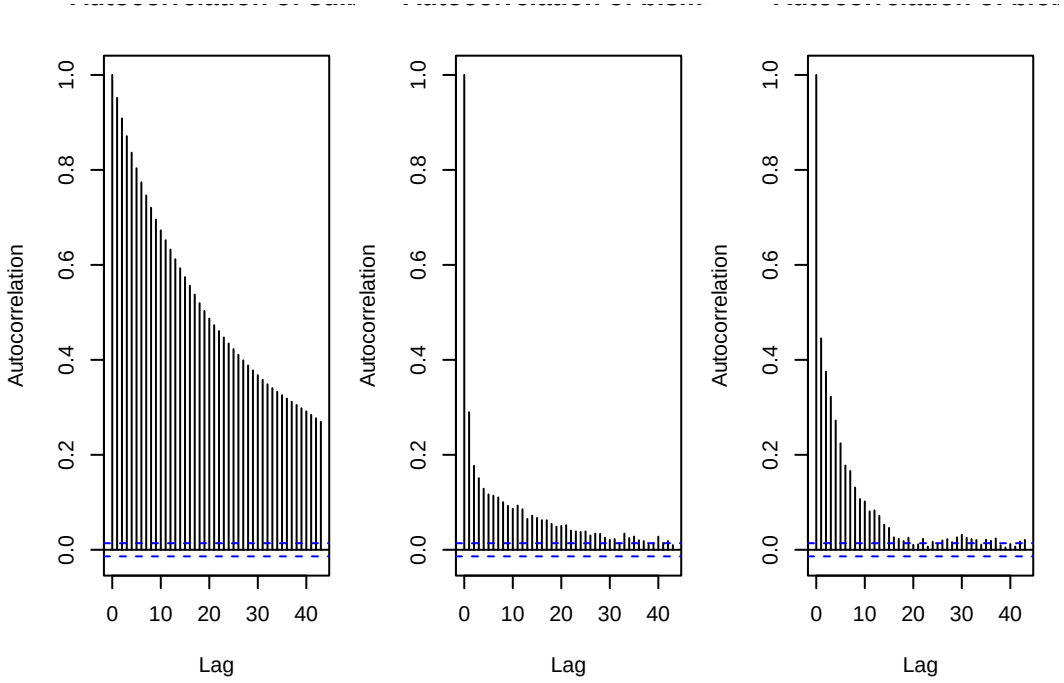
```
print(jags_fit)
```

```
Inference for Bugs model at "SmokeHyperModel.txt", fit using jags,
 3 chains, each with 20000 iterations (first 1 discarded)
 n.sims = 59997 iterations saved. Running time = 4.17 secs
          mu.vect sd.vect   2.5%    25%    50%    75%  97.5%  Rhat n.eff
b.ob        0.855   0.299  0.273  0.671  0.854  1.038  1.454 1.001 11000
b.smok      1.383   0.561  0.242  1.024  1.395  1.753  2.450 1.001  6300
sd.b        0.226   0.205  0.006  0.079  0.173  0.313  0.749 1.001  5100
deviance   61.269   3.909 55.242 58.479 60.697 63.452 70.437 1.001 32000

For each parameter, n.eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor (at convergence, Rhat=1).

DIC info (using the rule: pV = var(deviance)/2)
pV = 7.6 and DIC = 68.9
DIC is an estimate of expected predictive error (lower deviance is better).
```

```
# Convert JAGS output to MCMC object for diagnostics
mcmc_samples <- as.mcmc(jags_fit)

# Generate Trace Plots
par(mar=c(2,2,2,2))
plot(window(mcmc_samples, start=2000)) # question asks to show a portion of the iteration. H
```




```r
# Set up layout for 3 side-by-side autocorrelation plots
par(mfrow = c(1, 3), mar = c(4, 4, 2, 1))

# Generate Autocorrelation Plots with Labels
for (param in params) {
  acf(mcmc_samples[[1]][, param],
      main = paste("Autocorrelation of", param),
      xlab = "Lag",
      ylab = "Autocorrelation")
}
```




```
# Reset layout
par(mfrow = c(1, 1))
```

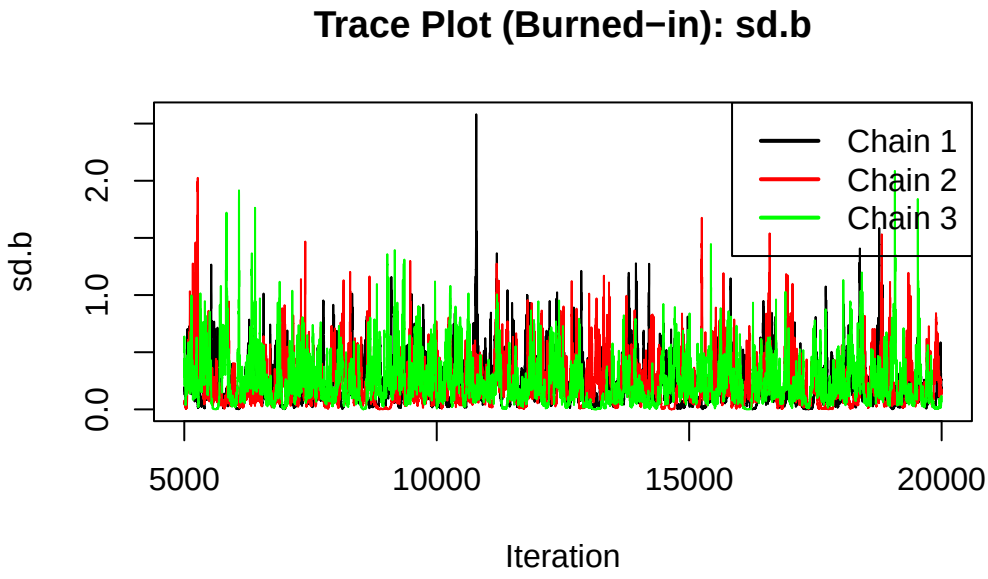
Part b

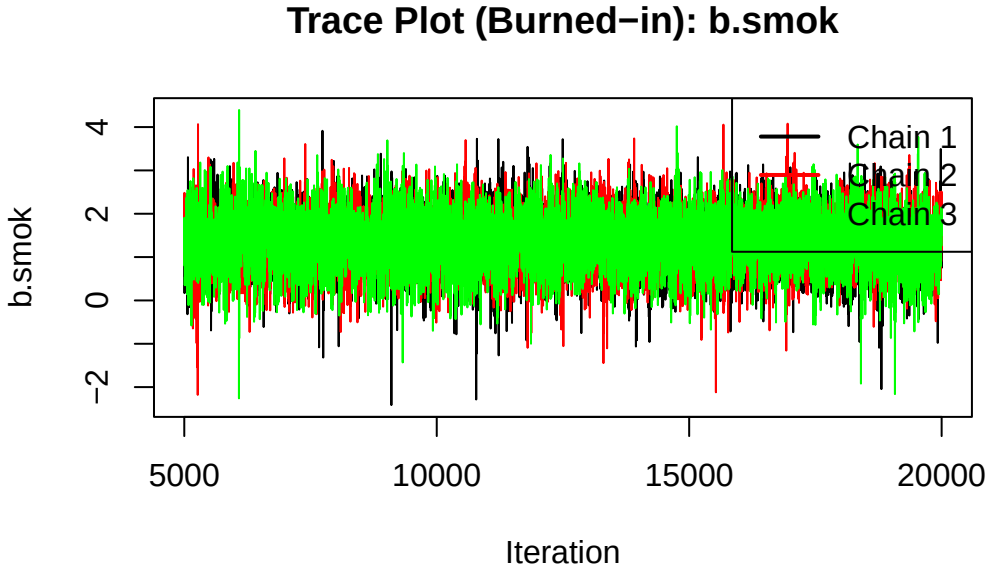
Burning-in refers to discarding the initial portion of the MCMC sample. Here, we use a burn-in of 5000 iterations. The primary reason for this is that the early iterations of an MCMC chain are often influenced by the initial values, which may be unstable and fluctuate significantly before reaching the stationary distribution. This instability can cause the chain to drift or jump toward high-probability regions before properly exploring the parameter space.

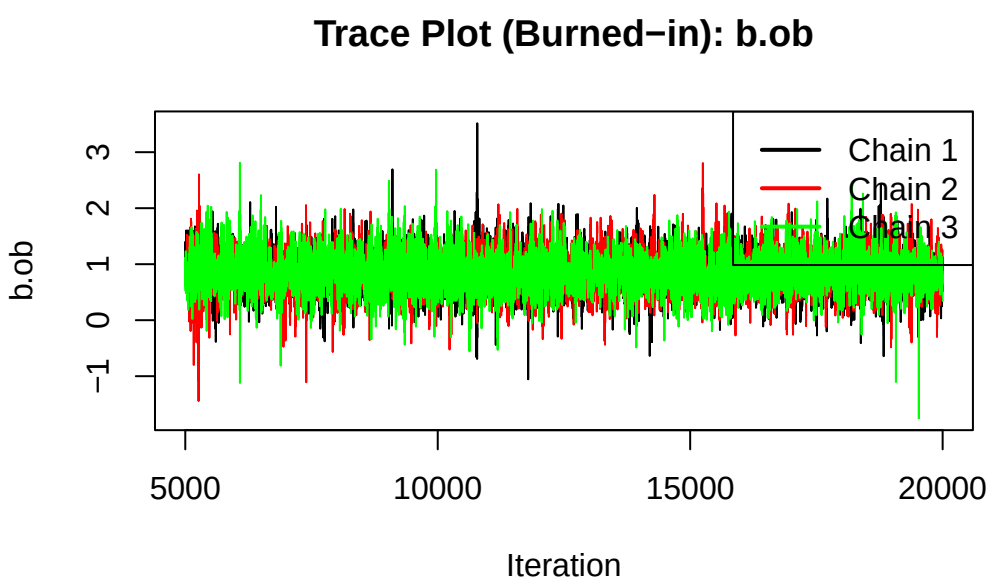
By discarding this initial period, we focus on the portion of the chain that has stabilized and adequately sampled from the posterior distribution. Additionally, early iterations often exhibit strong autocorrelation, meaning consecutive values are highly dependent on each other. Removing these iterations helps reduce this dependence and improves the quality of the remaining samples.

When plotting the density estimates, failing to discard the burn-in period can distort the posterior shape, as it may include samples that do not accurately represent the stationary distribution. Removing the burn-in ensures a more reliable estimate of the parameter distributions and enhances the validity of the inference.

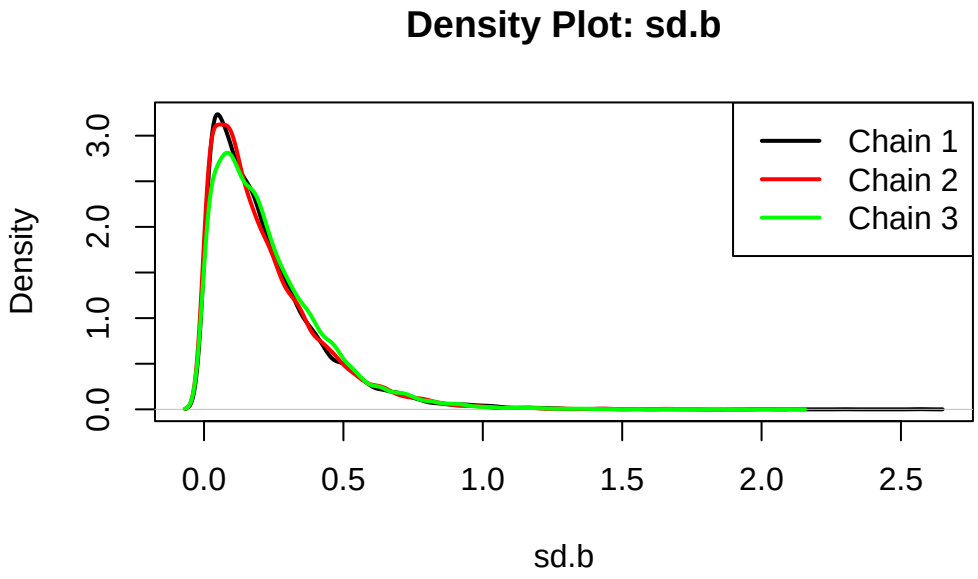
```r
# Extract posterior samples after burn-in (keeping only iterations after 5000)
burnin <- 5000
mcmc_samples_burned <- window(mcmc_samples, start = burnin)

# Define colors for the three chains
chain_colors <- c("black", "red", "green")

# Determine the correct iteration index
iter_index <- seq(burnin, 20000, by = 5)  # Adjusted for thinning (n.thin = 5)

# Trace plot of the MCMC samples after burn-in (each chain in a different color)
for (param in params) {
  matplot(
    burnin:20000,  # Display iterations from 5000 to 20000
    cbind(
      mcmc_samples_burned[[1]][, param],
      mcmc_samples_burned[[2]][, param],
      mcmc_samples_burned[[3]][, param]
    ),
    type = "l", col = chain_colors, lty = 1,
    main = paste("Trace Plot (Burned-in):", param),
    xlab = "Iteration", ylab = param
  )
  legend("topright", legend = c("Chain 1", "Chain 2", "Chain 3"), col = chain_colors, lwd = 1
}
```

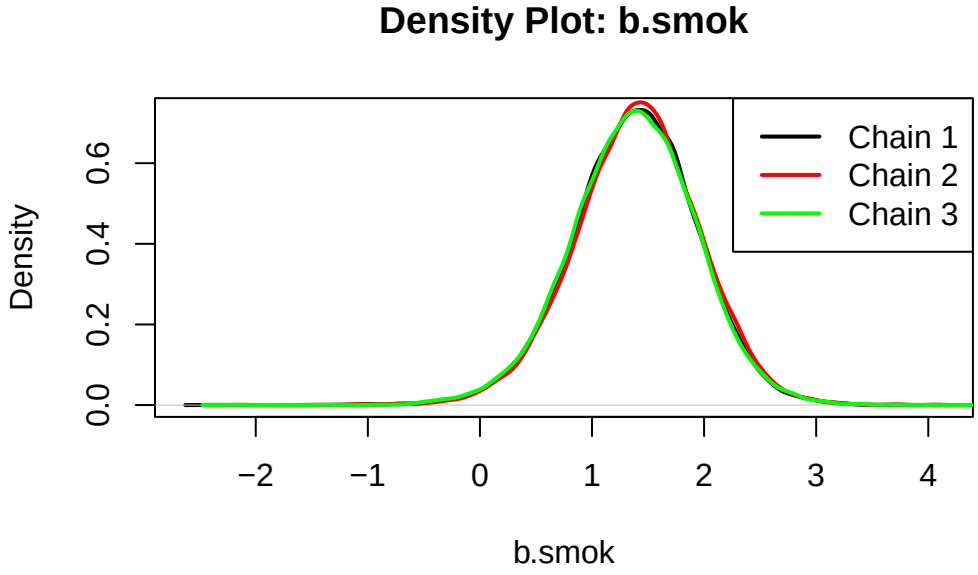

Trace Plot (Burned−in): sd.b
sd.b
0.0
1.0
2.0
5000
10000
15000
20000
Iteration
Chain 1
Chain 2
Chain 3

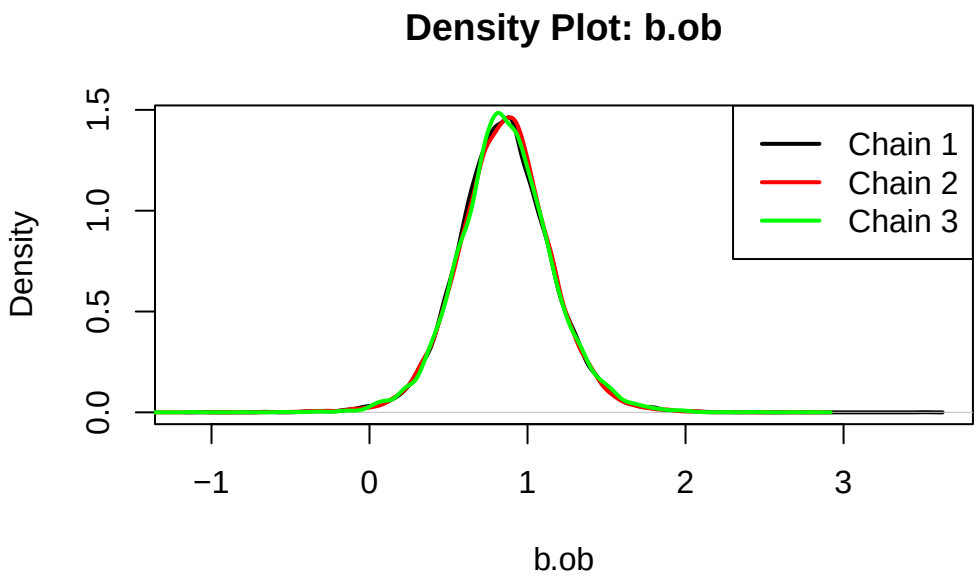

Trace Plot (Burned−in): b.smok
b.smok
−2
0
2
4
5000
10000
15000
20000
Iteration
Chain 1
Chain 2
Chain 3




```r
# Density plots for each parameter, comparing the three chains
for (param in params) {
  plot(
    density(mcmc_samples_burned[[1]][, param]),
    col = chain_colors[1], lwd = 2, main = paste("Density Plot:", param),
    xlab = param, ylab = "Density"
  )
  lines(density(mcmc_samples_burned[[2]][, param]), col = chain_colors[2], lwd = 2)
  lines(density(mcmc_samples_burned[[3]][, param]), col = chain_colors[3], lwd = 2)
  legend("topright", legend = c("Chain 1", "Chain 2", "Chain 3"), col = chain_colors, lwd = 2
}
```


Density Plot: sd.b
Density
0.0
1.0
2.0
3.0
0.0
0.5
1.0
1.5
2.0
2.5
sd.b
Chain 1
Chain 2
Chain 3


Density Plot: b.smok
Density
0.0
0.2
0.4
0.6
−2
−1
0
1
2
3
4
b.smok
Chain 1
Chain 2
Chain 3



Part c

Thinning refers to selecting every k-th sample from an MCMC chain to reduce autocorrelation and manage computational storage. If the chain exhibits strong autocorrelation, selecting every k-th iteration can help reduce dependency between consecutive samples, leading to a more independent subset. Additionally, MCMC simulations often generate a large number of samples, which can be memory-intensive. Thinning helps reduce storage requirements and computational overhead.

However, thinning is not always necessary and may even be inefficient if done improperly. If the chain mixes well and is not highly autocorrelated, thinning can unnecessarily discard useful information without improving inference. Instead of thinning, a better approach is often to run the chain for a longer duration and assess convergence diagnostics, ensuring that the collected samples adequately represent the posterior distribution.

Part d

We provide the estimate of the posterior mean and also give the MC accuracy using the method of batch means and using the autocorrelation function.

```r
# Extract posterior samples after burn-in
burnin <- 5000
mcmc_samples_burned <- window(mcmc_samples, start = burnin)

# Compute posterior means for each chain
posterior_means <- sapply(mcmc_samples_burned, colMeans)

# Define the batch means method function
CalcBatchMeans <- function(x, Batn=50) {
  BigN <- length(x)  # Total number of MCMC samples
  BatInc <- ceiling((1:BigN) / (BigN / Batn))  # Assign batch indices
  BM <- tapply(x, BatInc, mean)  # Compute batch means
  MCSE <- sd(BM) / sqrt(length(BM))  # Compute Monte Carlo SE
  return(list(MCSE = MCSE, MC_Accuracy = 1 / MCSE))  # Also compute accuracy
}

# Compute Monte Carlo standard error using batch means
mcse_batch_means <- sapply(mcmc_samples_burned, function(chain) {
  apply(chain, 2, function(param_samples) CalcBatchMeans(param_samples)$MCSE)
})

mc_accuracy_batch_means <- sapply(mcmc_samples_burned, function(chain) {
  apply(chain, 2, function(param_samples) CalcBatchMeans(param_samples)$MC_Accuracy)
})

# Define the autocorrelation function method for MCSE
CalcAcSe <- function(x, lag.max = 50) {
  autoc <- (acf(x, lag.max = lag.max, plot = FALSE))$acf  # Compute autocorrelations
  return(sd(x) / sqrt(length(x)) * sqrt(-1 + 2 * sum(autoc)))  # Compute MCSE
}

# Compute Monte Carlo standard error using the autocorrelation function method
mcse_acf <- sapply(mcmc_samples_burned, function(chain) {
  apply(chain, 2, CalcAcSe)
})

# Compute Monte Carlo accuracy using the autocorrelation function method
mc_accuracy_acf <- sapply(mcmc_samples_burned, function(chain) {
  apply(chain, 2, function(param_samples) 1 / CalcAcSe(param_samples))
})

# Print results
```

```
cat("Posterior Means for each Chain:\n")
```

```
Posterior Means for each Chain:
```

```
print(posterior_means)
```

```
               [,1]       [,2]       [,3]
b.ob      0.8528982  0.8558592  0.8591966
b.smok    1.3824578  1.4036644  1.3728442
deviance 61.2011647 61.3033714 61.3150374
sd.b      0.2212803  0.2188540  0.2321705
```

```
cat("\nMonte Carlo Accuracy (Batch Means Method):\n")
```

```
Monte Carlo Accuracy (Batch Means Method):
```

```
print(mc_accuracy_batch_means)
```

```
               [,1]      [,2]      [,3]
b.ob     160.88693 152.95189 141.57955
b.smok    90.46519  76.61937  80.28557
deviance  14.84402  14.27846  13.70330
sd.b      86.87790  88.95216  86.00577
```

```
cat("\nMonte Carlo Accuracy (Autocorrelation Function Method):\n")
```

```
Monte Carlo Accuracy (Autocorrelation Function Method):
```

```
print(mc_accuracy_acf)
```

```
               [,1]      [,2]      [,3]
b.ob     143.81876 151.15585 137.24783
b.smok    90.53284  80.76705  84.00118
deviance  13.65442  12.70666  12.79382
sd.b      88.85539  92.34191  95.97802
```

Part e

```
# Load necessary library
library(coda)

# Convert the MCMC samples into a format suitable for convergence diagnostics
mcmc_samples_coda <- as.mcmc.list(mcmc_samples_burned)

# Geweke Diagnostic (Z-scores for each parameter in each chain)
geweke_results <- lapply(mcmc_samples_coda, geweke.diag)

# Brooks-Gelman-Rubin Diagnostic (Split R-hat values)
bgr_results_split <- gelman.diag(mcmc_samples_coda, autoburnin = FALSE)

# Print Results
cat("Geweke Diagnostic Z-Scores:\n")
```

```
Geweke Diagnostic Z-Scores:
```

```
print(geweke_results)
```

```
[[1]]

Fraction in 1st window = 0.1
Fraction in 2nd window = 0.5

    b.ob   b.smok deviance     sd.b
  1.0721   1.1412  -0.4732   0.1153


[[2]]

Fraction in 1st window = 0.1
Fraction in 2nd window = 0.5

    b.ob   b.smok deviance     sd.b
 -0.9615  -2.4797   0.7047  -0.1115


[[3]]
```

```
Fraction in 1st window = 0.1
Fraction in 2nd window = 0.5

    b.ob   b.smok deviance     sd.b
  2.2810   0.5064  -0.6018   2.6502
```

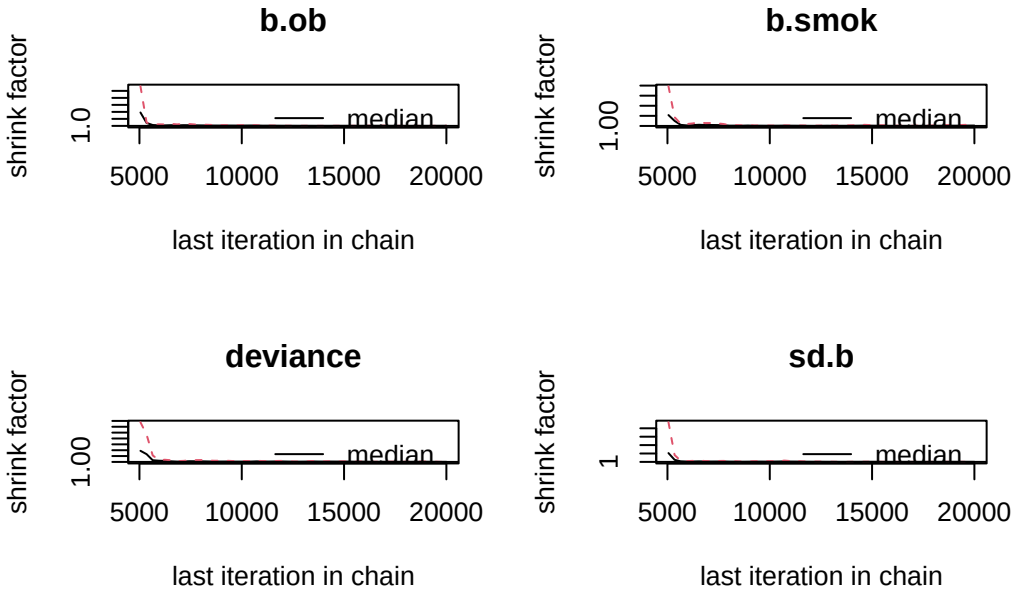
```
cat("\nBrooks-Gelman-Rubin Diagnostic (Split R-hat values):\n")
```

```
Brooks-Gelman-Rubin Diagnostic (Split R-hat values):
```

```
print(bgr_results_split)
```

```
Potential scale reduction factors:

         Point est. Upper C.I.
b.ob              1          1
b.smok            1          1
deviance          1          1
sd.b              1          1

Multivariate psrf

1
```

```
# Plot Split R-hat Diagnostic (for visual assessment)
gelman.plot(mcmc_samples_coda)
```



The Geweke diagnostic compares the means of early and late portions of the chain. A Z score of 0 indicates the chain is likely converged, while Z score away from 0 (2 or more) suggests non-convergence. We can see that the result from all three chains has abs(Z) < 2 (or standard deviations from 0) which suggests convergence.

The Brooks-Gelman-Rubin diagnostic compares the between-chain variance to the within-chain variance to determine if the chains have converged. Hence, if the output statistic R-hat is close to 1, this suggests a good convergence. If R-hat is over 1.2, we should be worried. We can see that all of our R-hat values are 1. This suggests convergence.

Part f

Based on the traceplot, we detect no trends of drifts for the three parameters b.ob, sd.b, and b.smok across the three chains. For the density plot, all three parameters show nearly identical shape, indicating it is from the same posterior distribution. This is a sign that the chain is mixed well.

Based on the Geweke Diagnostic, most Z-scores are close to 0, meaning the mean of the early and late parts of the chain are similar. Based on the BGR diagnostic, all R-hat values are close to one, suggesting good convergence.

Based on the evidence stated, I think the MCMC algorithm has converged.

Question 2

Part a

For linear model:

```
cat("
  model{
    for(i in 1:16){
    y[i]~dnorm(mu[i],tau)
    mu[i]<- b[1] + b[2]*(x[i]-31)

    #  likelihood for each observed and replicated data....
    #  note: need to know the density function of the probability model
    loglike[i]<-  (0.5)*log(tau/6.283) + (-0.5)*tau*pow((y[i]-mu[i]),2)

    }
  b[1]~dnorm(0,.000001)
  b[2]~dnorm(0,.000001)
  tau~dgamma(.0001,.0001)
  sd <- 1/sqrt(tau)

  #   Deviance statistic
  dev <- -2*sum(loglike[])

  }
  ", file="model1_q2.txt")


model1.Q2.dat=list("x", "y")  # what variable you need in the model

initM1.Q2.fun=function(){ list(b=c(runif(1,-.8,-.2), runif(1,-.8,-.2)),
                               tau=runif(1,.2,.8)) }

paramsM1.Q2=c("b[1]", "b[2]", "sd", "tau",
              "dev") # what variables you want to monitor

Q2.dat <- list( x=c(16,18,20,22,24,26,28,30,32,34,36,38,40,42,44,46),
y=c(2508,2518,3304,3423,3057,3190,3500,3883,3823,3646,3708,
3333,3517,3241,3103,2776))

#### Could change the code below...
```

```
model1.Q2 <-jags(Q2.dat, initM1.Q2.fun, paramsM1.Q2,
                 model.file="model1_q2.txt",
                 n.chains=3, n.iter=10000, n.burnin=2000, n.thin=10)
```

```
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
Graph information:
   Observed stochastic nodes: 16
   Unobserved stochastic nodes: 3
   Total graph size: 165

Initializing model
```

```
############################
##  Calculating the DIC
############################

# comparing intrinsic and self calculated value for Deviance:
xxx <- model1.Q2$BUGSoutput
x1<-xxx$sims.list$dev
x2<-xxx$sims.list$deviance
cat("Intrinsic deviance:", xxx$mean$deviance,
    "\nSelf calculated deviance:", xxx$mean$dev)
```

```
Intrinsic deviance: 239.6024
Self calculated deviance: 239.602
```

```
Dbar<-mean(x2) #Dbar
pd2<-0.5*var(x2) #pd2

devNormFunc <- function(beta0, beta1, beta2, tau, x, y){
     mu<- beta0+beta1*(x-31) + beta2*(x-31)^2
     return(-2*sum(log(dnorm(y,mu,1/sqrt(tau)))))}

beta0Bar<- xxx$mean$b[1]
beta1Bar<- xxx$mean$b[2]
beta2Bar<- 0
tauBar <- xxx$mean$tau
Dhat <- devNormFunc(beta0Bar, beta1Bar, beta2Bar, tauBar,
                    Q2.dat$x, Q2.dat$y) #Dhat
```

```
pd1<-Dbar-Dhat #pd1

DIC1<-Dbar+pd1
cat("DIC1 linear model:", DIC1)
```

```
DIC1 linear model: 242.6524
```

```
DIC2<-Dbar+pd2
cat("DIC2 linear model:", DIC2)
```

```
DIC2 linear model: 243.5579
```

```
cat("Intrinsic DIC linear model:", xxx$DIC)
```

```
Intrinsic DIC linear model: 243.5579
```

For quadratic model:

```
cat("
  model{
  for(i in 1:16){
  y[i]~dnorm(mu[i],tau)
  mu[i]<- b[1] + b[2]*(x[i]-31)+ b[3]*pow((x[i]-31),2)

  #  likelihood for each observed and replicated data....
  #  note: need to know the density function of the probability model
  loglike[i] <- (0.5)*log(tau/6.283) + (-0.5)*tau*pow((y[i]-mu[i]),2)
  }

  b[1]~dnorm(0,.000001)
  b[2]~dnorm(0,.000001)
  b[3]~dnorm(0,.01)
  tau~dgamma(.0001,.0001)
  sd <- 1/sqrt(tau)

  #   Deviance statistic
  dev <- -2*sum(loglike[])

  }
  ", file="model2_q2.txt")
```

```
initM2.Q2.fun=function(){ list(b=c(runif(1,-.8,-.2), runif(1,-.8,-.2),
                                   runif(1,-.8,-.2)),
                               tau=runif(1,.2,.8)) }

paramsM2.Q2=c("b[1]", "b[2]", "b[3]", "sd", "tau",
              "dev") # what variables you want to monitor

#### Could change the code below...
model2.Q2 <-jags(Q2.dat, initM2.Q2.fun, paramsM2.Q2,
                 model.file="model2_q2.txt",
                 n.chains=3, n.iter=10000, n.burnin=2000, n.thin=10)
```

```
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
Graph information:
   Observed stochastic nodes: 16
   Unobserved stochastic nodes: 4
   Total graph size: 191

Initializing model
```

```
############################
##  Calculating the DIC
############################

# comparing intrinsic and self calculated value for Deviance:
xxx <- model2.Q2$BUGSoutput
x1<-xxx$sims.list$dev
x2<-xxx$sims.list$deviance
cat("Intrinsic deviance:", xxx$mean$deviance,
    "\nSelf calculated deviance:", xxx$mean$dev)
```

```
Intrinsic deviance: 216.9505
Self calculated deviance: 216.95
```

```
Dbar<-mean(x2)
pd2<-0.5*var(x2)

devNormFunc <- function(beta0, beta1, beta2, tau, x, y){
```

```
     mu<- beta0+beta1*(x-31) + beta2*(x-31)^2
     return(-2*sum(log(dnorm(y,mu,1/sqrt(tau)))))}
beta0Bar<- xxx$mean$b[1]
beta1Bar<- xxx$mean$b[2]
beta2Bar<- xxx$mean$b[3]
tauBar <- xxx$mean$tau
Dhat <- devNormFunc(beta0Bar, beta1Bar, beta2Bar, tauBar, Q2.dat$x, Q2.dat$y)
pd1<-Dbar-Dhat

DIC1<-Dbar+pd1
cat("DIC1 quadratic model:", DIC1)
```

```
DIC1 quadratic model: 221.1721
```

```
DIC2<-Dbar+pd2
cat("DIC2 quadratic model:", DIC2)
```

```
DIC2 quadratic model: 222.3583
```

```
cat("Intrinsic DIC quadratic model:", xxx$DIC)
```

```
Intrinsic DIC quadratic model: 222.3582
```

Now we calculate Baye's factor.

```
set.seed(123)
model1.Q2.dat <- list(x=c(16,18,20,22,24,26,28,30,32,34,36,38,40,42,44,46),
y=c(2508,2518,3304,3423,3057,3190,3500,3883,3823,3646,3708,
3333,3517,3241,3103,2776))


SBayes.dat <- model1.Q2.dat
nvar <- 2
for(i in 1:nvar){
  xx<- model1.Q2.dat[[i]]
  mu<- mean(xx)
  sig<- sd(xx)
  SBayes.dat[[i]] <- (xx-mu)/sig
}
```

```r
names(SBayes.dat) <- c(paste("sx", 1:(nvar-1), sep=""),"sy")

cat("
model {

  for(i in 1:16){
    sy[i] ~ dnorm(mu[i], tau)
    mu[i] <- b[1] + b[2] * sx1[i] + del * b[3] * pow(sx1[i], 2)
  }

  # Updated priors: normal(0,1) for each b[j] (precision=1, which is between 1/16 and 4)
  b[1] ~ dnorm(0, 16)
  b[2] ~ dnorm(0, 16)
  b[3] ~ dnorm(0, 16)

  del ~ dbern(0.5)

  tau ~ dgamma(0.0001, 0.0001)
}
", file = "modelB_q2.txt")

initB.Q2.fun=function(){ list(b=c(runif(1,-.8,-.2), runif(1,-.8,-.2),
                                  runif(1,-.8,-.2)),
                              tau=runif(1,.2,.8),
                              del=rbinom(1,1,0.5)) }

paramsB.Q2=c("del") # what variables you want to monitor

#### Could change the code below...
modelB.Q2 <-jags(SBayes.dat, initB.Q2.fun, paramsB.Q2, model.file="modelB_q2.txt",
                 n.chains=3, n.iter=10000, n.burnin=2000, n.thin=10)
```

```
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
Graph information:
   Observed stochastic nodes: 16
   Unobserved stochastic nodes: 5
   Total graph size: 98

Initializing model
```

```r
PM2 <- mean(modelB.Q2$BUGSoutput$sims.list$del[,1]==1)
PM1 <- mean(modelB.Q2$BUGSoutput$sims.list$del[,1]==0)

cat("\nBayes factor:", PM2/PM1)
```

```
Bayes factor: 19.51282
```

From Bayes factor, it seems the quadratic model is favoured over the linear model decisively (>10) using the Kass and Raftery scale. The DIC values also supports this, as the quadratic model is lower for quadratic model(217.1158) compared to linear model (239.7103).

Part b

For linear model:

```r
cat("
  model {
    for(i in 1:16){
      y[i] ~ dnorm(mu[i], tau)
      mu[i] <- b[1] + b[2] * (x[i] - 31)

      # Log-likelihood for deviance
      loglike[i] <- 0.5 * log(tau / 6.283) - 0.5 * tau * pow((y[i] - mu[i]), 2)

      # Residuals
      res[i] <- y[i] - mu[i]
      stdres[i] <- res[i] * sqrt(tau)

      # Posterior predictive checks
      y.rep[i] ~ dnorm(mu[i], tau)
      res.rep[i] <- y.rep[i] - mu[i]
      stdres.rep[i] <- res.rep[i] * sqrt(tau)

      # Posterior predictive p-value (more extreme check)
      p.smaller[i] <- step(y[i] - y.rep[i])
    }

    # Priors
    b[1] ~ dnorm(0, 1.0E-6)
```

```
    b[2] ~ dnorm(0, 1.0E-6)
    tau ~ dgamma(0.0001, 0.0001)
    sd <- 1 / sqrt(tau)

    # Deviance
    dev <- -2 * sum(loglike[])
  }
", file = "model1_q2_updated.txt")

paramsM1.Q2 <- c("b[1]", "b[2]", "sd", "tau", "dev",
                 "res", "stdres", "res.rep", "stdres.rep", "p.smaller")

model1.Q2 <- jags(data = Q2.dat,
                  inits = initM1.Q2.fun,
                  parameters.to.save = paramsM1.Q2,
                  model.file = "model1_q2_updated.txt",
                  n.chains = 3, n.iter = 10000, n.burnin = 2000, n.thin = 10)
```

```
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
Graph information:
   Observed stochastic nodes: 16
   Unobserved stochastic nodes: 19
   Total graph size: 260

Initializing model
```

```
# extract posterior samples
res_post      <- model1.Q2$BUGSoutput$sims.list$res
stdres_post   <- model1.Q2$BUGSoutput$sims.list$stdres
psmaller_post <- model1.Q2$BUGSoutput$sims.list$p.smaller

# posterior means
res_mean      <- colMeans(res_post)
stdres_mean   <- colMeans(stdres_post)
pval_mean     <- colMeans(psmaller_post)

summary_b <- data.frame(
  Observation = 1:16,
  Residual = round(res_mean, 2),
```

```
  StdResidual = round(stdres_mean, 2),
  PredictivePval = round(pval_mean, 3)
)

print(summary_b)
```

```
   Observation Residual StdResidual PredictivePval
1            1  -556.19       -1.31          0.115
2            2  -570.22       -1.35          0.124
3            3   191.74        0.44          0.657
4            4   286.71        0.67          0.720
5            5  -103.33       -0.25          0.406
6            6     5.64        0.01          0.508
7            7   291.60        0.68          0.733
8            8   650.57        1.52          0.926
9            9   566.53        1.32          0.894
10          10   365.50        0.85          0.792
11          11   403.46        0.94          0.820
12          12     4.43        0.00          0.502
13          13   164.39        0.38          0.630
14          14  -135.64       -0.33          0.391
15          15  -297.68       -0.71          0.259
16          16  -648.71       -1.53          0.088
```

We observe that most observations have absolute valued standardized residuals less than abs(1.5), which is acceptable. Posterior predictive p-values generally remain near 0.5, indicating the model predicts observed outcome reasonably well. Across multiple runs, some pattern I noticed are observations 1, 2, 8, 9, and 16 stands out with high/low p-values (not near 0.5) which is also reflected through the slighly larger StdResidual values in absolute values. This may suggest non linearity in the data which motivates us to fit a quadratic term in model 2.

For quadratic model:

```
cat("
  model {
    for(i in 1:16){
      y[i] ~ dnorm(mu[i], tau)
      mu[i] <- b[1] + b[2] * (x[i] - 31) + b[3] * pow((x[i] - 31), 2)

      # Log-likelihood for deviance
      loglike[i] <- 0.5 * log(tau / 6.283) - 0.5 * tau * pow((y[i] - mu[i]), 2)
```

```
      # Residuals
      res[i] <- y[i] - mu[i]
      stdres[i] <- res[i] * sqrt(tau)

      # Posterior predictive checks
      y.rep[i] ~ dnorm(mu[i], tau)
      res.rep[i] <- y.rep[i] - mu[i]
      stdres.rep[i] <- res.rep[i] * sqrt(tau)

      # Posterior predictive p-value (more extreme check)
      p.smaller[i] <- step(y[i] - y.rep[i])
    }

    # Priors
    b[1] ~ dnorm(0, 1.0E-6)
    b[2] ~ dnorm(0, 1.0E-6)
    b[3] ~ dnorm(0, 1.0E-6)
    tau ~ dgamma(0.0001, 0.0001)
    sd <- 1 / sqrt(tau)

    # Deviance
    dev <- -2 * sum(loglike[])
  }
", file = "model2_q2_updated.txt")

paramsM2.Q2 <- c("b[1]", "b[2]", "b[3]", "sd", "tau", "dev",
                 "res", "stdres", "res.rep", "stdres.rep", "p.smaller")

model2.Q2 <- jags(data = Q2.dat,
                  inits = initM2.Q2.fun,
                  parameters.to.save = paramsM2.Q2,
                  model.file = "model2_q2_updated.txt",
                  n.chains = 3, n.iter = 10000, n.burnin = 2000, n.thin = 10)
```

```
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
Graph information:
   Observed stochastic nodes: 16
   Unobserved stochastic nodes: 20
   Total graph size: 285
```

```
Initializing model
```

```r
res_post2      <- model2.Q2$BUGSoutput$sims.list$res
stdres_post2   <- model2.Q2$BUGSoutput$sims.list$stdres
psmaller_post2 <- model2.Q2$BUGSoutput$sims.list$p.smaller

res_mean2      <- colMeans(res_post2)
stdres_mean2   <- colMeans(stdres_post2)
pval_mean2     <- colMeans(psmaller_post2)

summary_b2 <- data.frame(
  Observation = 1:16,
  Residual = round(res_mean2, 2),
  StdResidual = round(stdres_mean2, 2),
  PredictivePval = round(pval_mean2, 3)
)

print(summary_b2)
```

```
   Observation Residual StdResidual PredictivePval
1            1    25.53        0.13          0.552
2            2  -232.13       -1.10          0.171
3            3   321.02        1.53          0.907
4            4   242.00        1.15          0.856
5            5  -287.20       -1.38          0.109
6            6  -282.58       -1.36          0.107
7            7   -66.15       -0.33          0.388
8            8   258.11        1.22          0.871
9            9   174.19        0.82          0.789
10          10     8.09        0.03          0.510
11          11   115.81        0.54          0.695
12          12  -178.65       -0.86          0.206
13          13   120.71        0.57          0.693
14          14    -5.11       -0.03          0.502
15          15    41.89        0.20          0.581
16          16   -65.29       -0.31          0.392
```

Similarly, we observe that more observations have absolute valued standardized residuals less than abs(1.5), which indicates strong fit. maybe even a little better than the linear model. Predictive p-values are well behaved with only a few outliers across each run. A few points like observation 3, 5, 6 still show moderate deviations, but are less extreme than the ones

we saw in the linear model. The added quadratic term seems to improved overall model adequacy.

Question 3

```
g = function(x){(x>0)*(x<1)*((x<=0.5)*4*x+ (x>0.5)*(4-4*x))}
n = 1000
```

Part a

```
set.seed(2025)

u1 <- runif(n)
u2 <- runif(n)
z <- (u1 + u2)/2

zbar <- mean(z)
zvar <- var(z)

cat("Estimated E(X):", zbar, "\n")
```

```
Estimated E(X): 0.5038997
```

```
cat("Estimated V(X):", zvar, "\n")
```

```
Estimated V(X): 0.0419637
```

Part b

(i)

The weight function is defined as $w(x_i) = \frac{f(x_i)}{g(x_i)}$. In our case, $w(x) = \begin{cases} 4x & \text{for } 0 \leq x \leq 0.5 \\ 4(1-x) & \text{for } 0.5 < x \leq 1 \\ 0 & \text{otherwise} \end{cases}$

(ii)

```
set.seed(3)

x <- runif(n)
w <- g(x)

exp_est <- sum(x * w)/ sum(w) # E(Z)
exp_est2 <- sum(x^2 * w) / sum(w) # E(Z^2)
var_est <- exp_est2 - exp_est^2 # V(Z) = E(Z^2) - (E(Z)^2

cat("Estimated E(X):", exp_est, "\n")
```

```
Estimated E(X): 0.5016185
```

```
cat("Estimated V(X):", var_est, "\n")
```

```
Estimated V(X): 0.04239576
```

Part c

(i)

```
c <- 2 # max(g(x)) = 2 so choose constant = 2

accepted <- numeric(0)
n_trials <- 0

while (length(accepted) < n){
  x <- runif(1) # proposal from f(x) = uniform(0,1)
  u <- runif(1) # uniform for acceptance test
  if (u <= g(x) / c){
    accepted <- c(accepted, x)
  }
  n_trials <- n_trials + 1
}
```

The goal is to sample from the target distribution $g(x)$, the triangle distribution on [0, 1]. We use the $f \sim \text{Uniform}(0, 1)$ as our proposal distribution. We choose the constant $M = 2$, as $g(x) \leq 2 \cdot g(x)$ over the interval $[0, 1]$. Note that $g \leq Mf$

Now, we begin by sampling $X \sim f = \text{Uniform}(0, 1)$, and also $U \sim \text{Uniform}(0, 1)$. We accept $Y = X$ if

$$U \leq \frac{g(X)}{Mf(X)} = \frac{g(X)}{2} \qquad \text{(acceptance test function)}$$

. If not accepted, we return to the beginning of the paragraph and restart this process.

(ii)

```
mean_ar <- mean(accepted)
var_ar <- var(accepted)
acceptance_rate <- n / n_trials
cat("Acceptance rate:", acceptance_rate * 100, "%\n")
```

```
Acceptance rate: 50.15045 %
```

(iii)

```
cat("Acceptance Rejection Estimate of E(X):",mean_ar,"\n")
```

```
Acceptance Rejection Estimate of E(X): 0.5000323
```

```
cat("Acceptance Rejection Estimate of Var(X):",var_ar,"\n")
```

```
Acceptance Rejection Estimate of Var(X): 0.0410941
```

Part d

(i)

The test function

$$\alpha(x, y) = \min\left(1, \frac{u(y)q(y, x)}{u(x)q(x, y)}\right)$$

In this case, $q(x, y) = q(y, x) = 1$ and $u(x) = g(x)$. Hence the above simplifies

$$\alpha(x, y) = \min\left(1, \frac{g(y)}{g(x)}\right)$$

where g is given.

(ii)

```
set.seed(23)
x <- numeric(n)
x[1] <- runif(1) # start at random value
accepted <- 0

for (i in 2:n){
  y <- runif(1) # Proposed y ~ Uniform(0,1)

  alpha <- min(1, g(y) / g(x[i-1])) # test function

    # Accept or reject
  if (runif(1) <= alpha) {
    x[i] <- y
    accepted <- accepted + 1
  } else {
    x[i] <- x[i - 1]
  }
}
```

(iii)

```
accept_rate <- accepted / (n-1) # subtract 1 to account for the first value x[1]
cat("The acceptance rate for the proposed moves in this chain is", accept_rate, "\n")
```

```
The acceptance rate for the proposed moves in this chain is 0.6716717
```

(iv)

```r
mean_mh <- mean(x)
var_mh <- var(x)

cat("Metropolis-Hastings Estimate of E(X):", mean_mh, "\n")
```

```
Metropolis-Hastings Estimate of E(X): 0.5058745
```

```r
cat("Metropolis-Hastings Estimate of Var(X):", var_mh, "\n")
```

```
Metropolis-Hastings Estimate of Var(X): 0.04117888
```