Lab 5: MCMC

Team USA: Yuting Deng, Kendra Gilbertson, Lily Durkee, Bennett Hardy

2022-09-14

Contact: Hermione.Deng@colostate.edu, kendra01@colostate.edu, L.Durkee@colostate.edu,Bennett.Hardy@colostate.edu

Problem

You will write code using conjugate relationships, also known as Gibbs updates, to draw samples from marginal posterior distributions of a mean and variance.

1. Set the seed for random numbers = 10 in R with set.seed(10).
2. Load the actuar library, which contains functions for inverse gamma distributions.
3. Simulate 100 data points from a normal distribution with mean $\theta = 100$ and variance $\varsigma^2 = 25$. Call the data set y. Be careful here. R requires the standard deviation, not the variance, as a parameter. You will use these "fake" data to verify the Gibbs sampler you will write below. Simulating data is always a good way to test methods. Your method should be able to recover the generating parameters given a sufficiently large number of simulated observations.

I have saved you some time by writing a function called draw_mean that makes draws from the marginal posterior distributions for θ using a normal-normal conjugate relationship where the variance is assumed to be known. It is vital that you study the MCMCmath.pdf notes relative to this function.

4. I have saved you some time by writing a function called draw_mean that makes draws from the marginal posterior distributions for θ using a normal-normal conjugate relationship where the variance is assumed to be known. It is vital that you study the MCMCmath.pdf notes relative to this function.
5. I have also provided a function called draw_var that makes draws from the marginal posterior distribution for ς^2 using a inverse gamma-normal conjugate relationship where the mean is assumed to be known. Study this function relative to the MCMCmath.pdf handout.
6. Check the functions by simulating a large number of data points from a normal distribution using a mean and variance of your own choosing. Store the data points in a vector called y_check. Assume flat priors for the mean and the variance. A vague prior for the inverse gamma has parameters $\alpha_0 = .001$ and $\beta_0 = .001$.

```
## $z
## [1] 0.0791588
##
## $mu_1
## [1] 0.001564032
##
## $sigma.sq_1
## [1] 0.002499938
```

```
## $z
## [1] 9907.478
##
## $alpha_1
## [1] 5000.001
##
## $beta_1
## [1] 50003495
```

Write a sampler

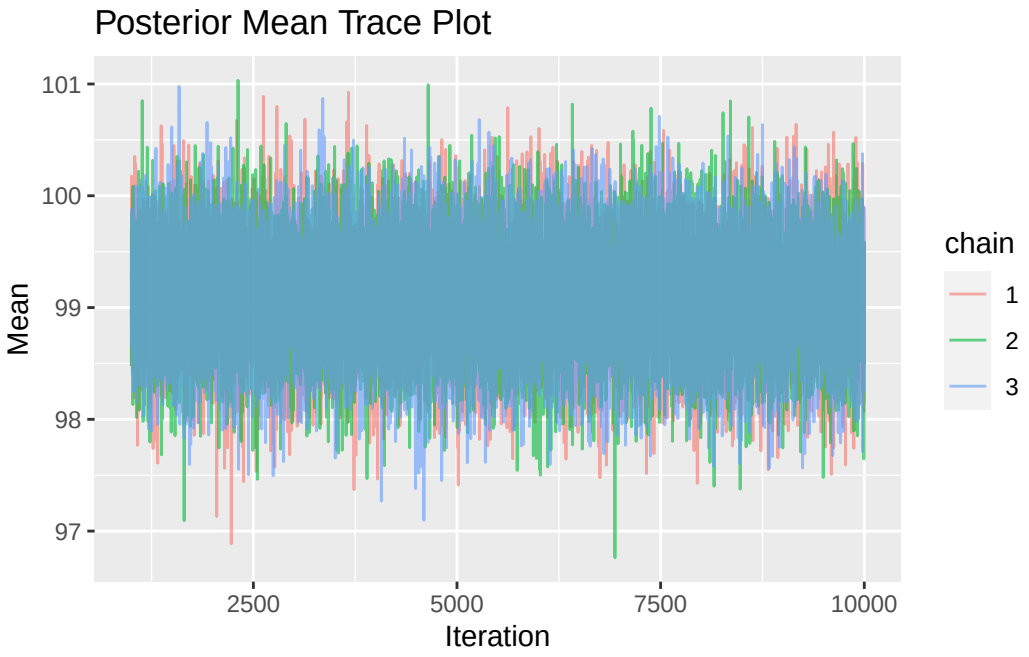
1. Set up a matrix for storing samples from the posterior distribution of the mean. The number of rows should equal the number of chains (3) and number of columns should equal the number of iterations (10,000). Do the same thing for storing samples from the posterior distribution of the variance.

2. Assign initial values to the first column of each matrix, a different value for each of the chains. These can be virtually any value within the support of the random variable, but it would be fine for this exercise to use values not terribly far away from to those you used to simulate the data, reflecting some prior knowledge. You might try varying these later to show that you will get the same results.

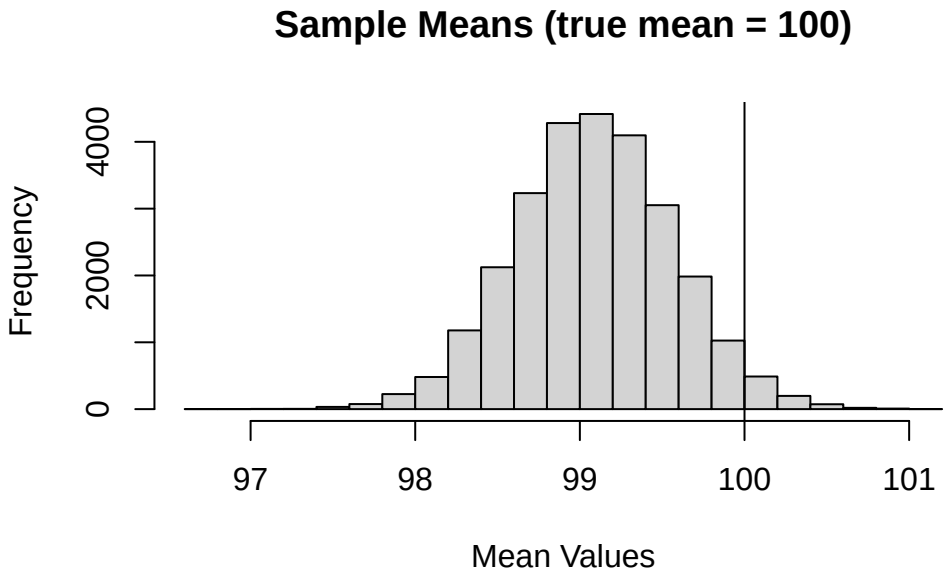
3. Set up nested for loops to iterate from one to the total number of iterations for each of the three chains for each parameter. Use the conjugate functions draw_mean and draw_var to draw a sample from the distribution of the mean using the value of the variance at the current iteration. Then make a draw from the variance using the current value of the mean. Repeat. Assume vague priors for the mean and variance:

Trace plots and plots of marginal posteriors

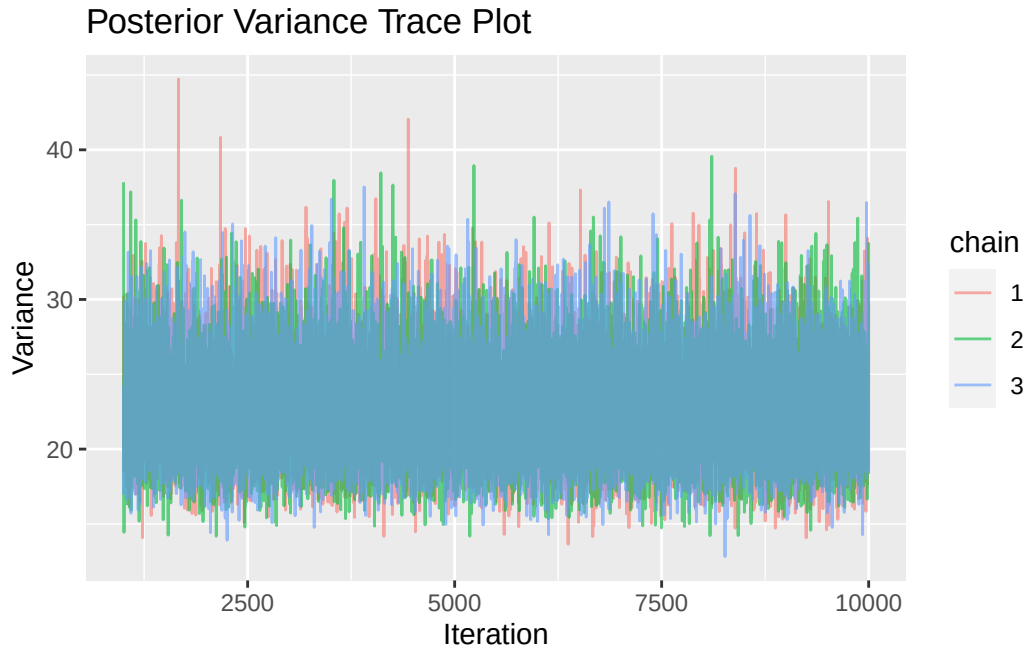
1. Discard the first 1000 iterations as burn-in. Plot the value of the mean as a function of iteration number for each chain. This is called a trace plot.

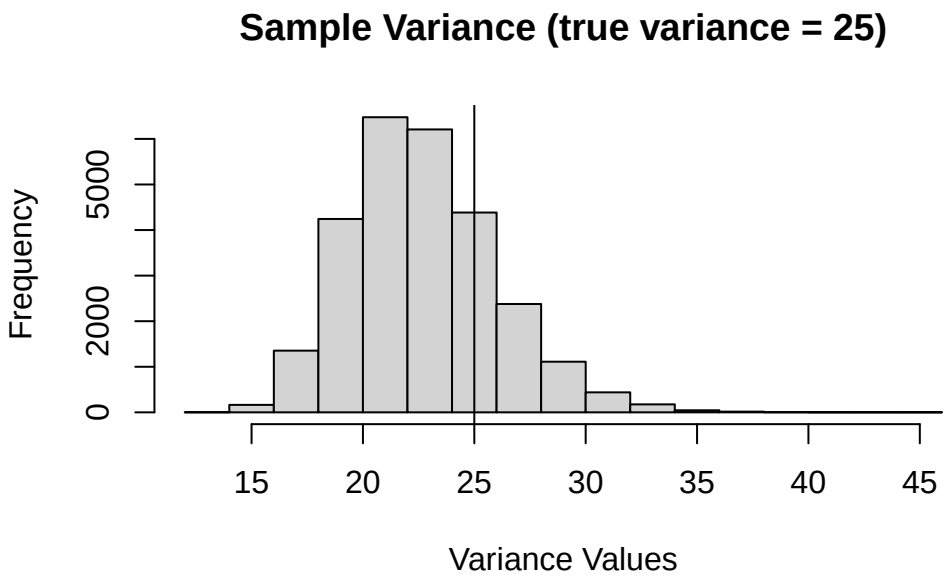


2. Make a histogram of the samples of the mean retained after burn-in including all chains. Put a vertical line on the plot showing the generating value.



3. Repeat steps 1-2 for the variance.





4. For both θ and ς^2, calculate the mean of all the chains combined and its standard deviation. Interpret these quantities.

Mean:

```
## [1] 99.08613
```

```
## [1] 0.4788827
```

The average mean of all our 9000 iterations of the mean is 99.1, and its standard deviation is 0.48. The estimate is very precise, and close to our true known mean of 100.

Variance

```
## [1] 22.68649
```

```
## [1] 3.304163
```

The average mean of all our 900 iterations of the variance is 22.69, and its standard deviation is 3.3. The estimate is very precise, and close to our true known variance of 25.

5. Compare the standard deviation of the posterior distribution of θ with an approximation using the standard deviation of the data divided by the square root of the sample size. What is this approximation called in the frequentist world?

```
## [1] 0.4788827
```

```
## [1] 0.4706179
```

The approximation in the frequentist world is called "standard error".

6. Vary the number of values in the simulated data set, e.g., $n = 10, 100, 1,000, 10,000$. We do not exactly recover the generating values of θ and ς^2 when n is small. Why? The mean of the marginal posterior distribution of the variance is further away from its generating value than the mean is. Why? Try different values for set seed with $n = 100$ and interpret the effect of changing the random number sequence.

A. n=10

Average mean and standard deviation of mean

```
## [1] 97.35436
```

```
## [1] 2.403836
```

Average variance and standard deviation of variance

```
## [1] 45.82257
```

```
## [1] 29.29341
```

B. n=100

Average mean and standard deviation of mean

```
## [1] 100.6783
```

```
## [1] 0.5256242
```

Average variance and standard deviation of variance

```
## [1] 27.09358
```

```
## [1] 4.351669
```

C. n=1000

Average mean and standard deviation of mean

```
## [1] 99.85155
```

```
## [1] 0.1584277
```

Average variance and standard deviation of variance

```
## [1] 25.47553
```

```
## [1] 1.111865
```

C. n=10000

Average mean and standard deviation of mean

```
## [1] 100.0269
```

```
## [1] 0.05050042
```

Average variance and standard deviation of variance

```
## [1] 25.28128
```

```
## [1] 0.3518398
```

E. Use a different value for set seed with n=100

Average mean and standard deviation of mean

```
## [1] 99.59813
```

```
## [1] 0.4632297
```

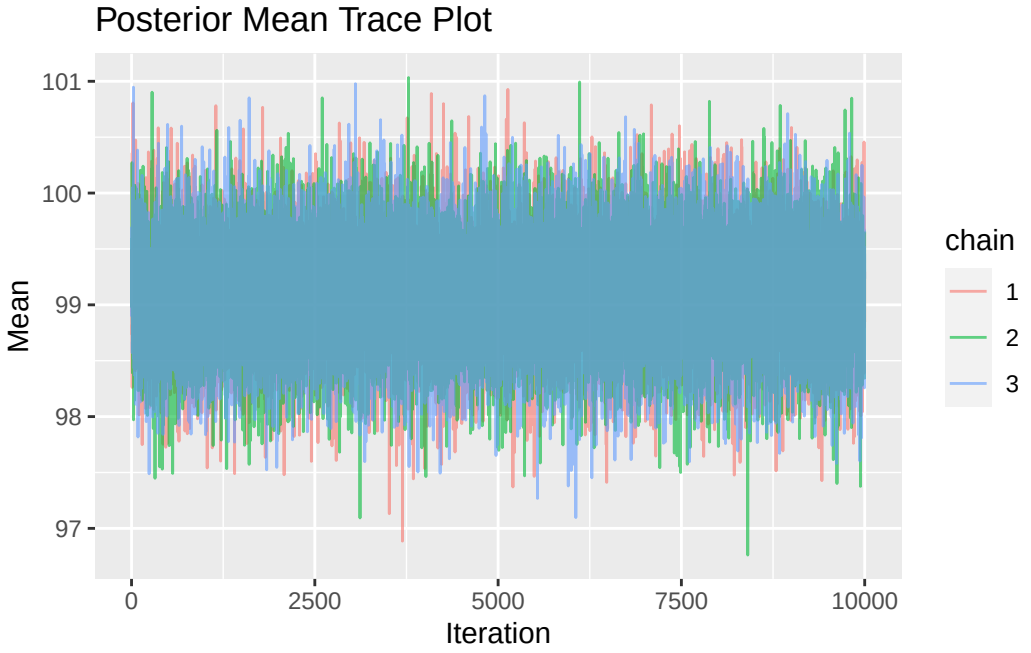
Average variance and standard deviation of variance

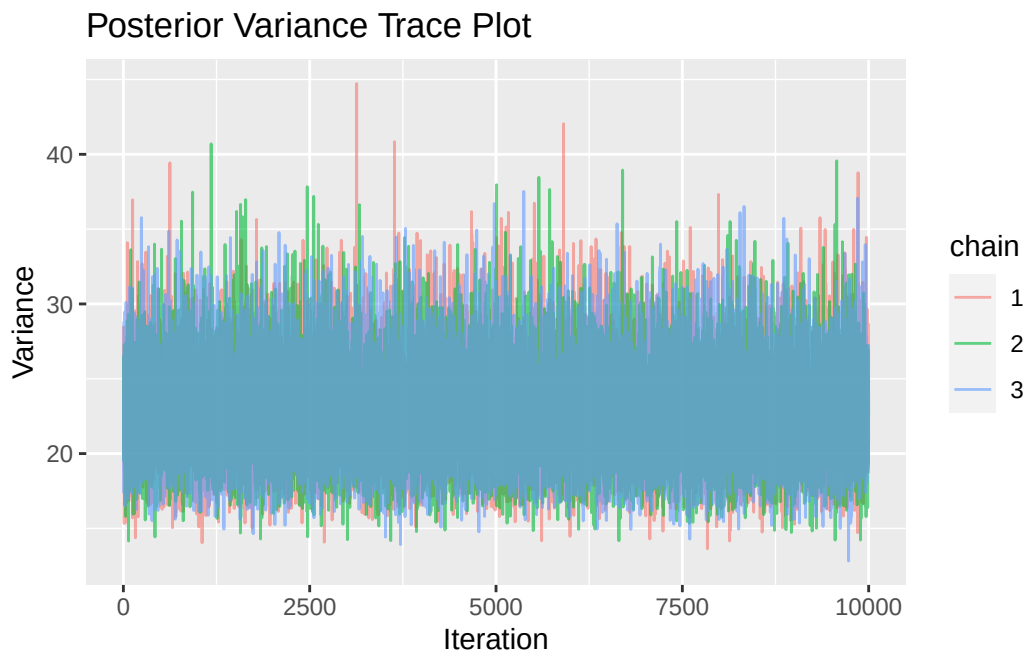
```
## [1] 22.45903
```

```
## [1] 3.423258
```

When setting a different seed value and sample size is relatively small (n=100), we saw a different mean and variance of the posterior distribution from the ones with seed.value = 10, both different from the generating values. It is because each sample is achieving a different result in stochastic process, thus the mean and variance of the posterior distribution would be different each time. When sample size is small, it's harder to get to the true value. When sample size is big enough (n=10000), the posterior distribution has the mean and variance closer to the generating value. The mean of the marginal posterior distribution of the variance is further away from its generating value than the mean because variance is more sensitive to the dispersed data.

7. Make the burnin=1 instead of 1000. Does this change your results? Why or why not?






```
## [1] 99.08613
```

```
## [1] 0.4788827
```

```
## [1] 22.68649
```

```
## [1] 3.304163
```

Despite reducing the burnin to one, both our posterior mean and variance quickly converged. This means that Gibbs update is very effective. Their averages were very close to the true values, and the standard deviation for both estimates was small. This is because we are still using a large sample size of 10000; it is a large enough sample to make up for a short burnin period.

8. Reverse the order of the conjugate functions in step 3 of the Writing a Sampler section so that the variance is drawn first followed by the mean. Does this reordering have an effect on the posteriors? Why or why not?

```
## [1] 99.08613
```

```
## [1] 0.4788827
```

```
## [1] 22.68649
```

```
## [1] 3.304163
```

Changing the order within the sampler does not effect the posteriors because samples are drawn from the full-conditionals. The only significant change is that we're using the posterior mean initial value instead of the variance initial value, and from the second iteration on they're each samplin from each other. This is too small to make a difference, but we're also dropping the first iteration during burning, so it isn't included in our results.

Code

```
knitr::opts_chunk$set(
    echo = FALSE,
    fig.height = 3.5,
    fig.width = 5.5,
    message = FALSE,
    warning = FALSE,
    attr.source = ".numberLines"
)
library(ggplot2)
library(gridExtra)
library(dplyr)
library(ggpubr)
library(mathjaxr)
library(actuar)
set.seed(10)
library(actuar)
y<-rnorm(100,mean=100,sd=5)
# normal likelihood with normal prior conjugate for mean, assuming variance is known
# mu_0 is prior mean
# sigma.sq_0 is prior variance of mean
# varsigma.sq is known variance of data

draw_mean = function(mu_0, sigma.sq_0, varsigma.sq, y){
    mu_1 =((mu_0 / sigma.sq_0 + sum(y)/varsigma.sq)) / (1/sigma.sq_0 + length(y) / varsigma.sq)
    sigma.sq_1 = 1/(1 / sigma.sq_0 + length(y) / varsigma.sq)
    z = rnorm(1, mu_1, sqrt(sigma.sq_1))
    param = list(z = z, mu_1 = mu_1, sigma.sq_1 = sigma.sq_1)
    return(param)
}
# normal likelihood with gamma prior conjugate relationship for variance, assuming mean is known
# alpha_0 is parameter of prior for variance
# beta_0 is parameter of prior for variance
# Note that this uses scale parameterization for inverse gamma

draw_var = function(alpha_0, beta_0, theta, y){
    alpha_1 = alpha_0 + length(y) / 2
    beta_1 = beta_0 + sum((y - theta)^2) / 2
    z = rinvgamma(1, alpha_1, scale = beta_1)
    param = list(z = z, alpha_1 = alpha_1, beta_1 = beta_1)
    return(param)
}
y_check<-rnorm(10000,0,1)

draw_mean(mu_0=0, sigma.sq_0=100, varsigma.sq=25, y=y_check)

draw_var(alpha_0=.001, beta_0=.001, theta=100, y=y_check)
post_mean<-matrix(NA,nrow=3,ncol=10000)
post_var<-matrix(NA,nrow=3,ncol=10000)
post_mean[,1]<-c(3,4,5)
post_var[,1]<-c(3,4,5)
a=.001
```

```r
b=.001
mu<-0
sigma<-100

for(i in 1:9999){
    for(x in 1:3){
      post_mean[x,i+1]<-draw_mean(mu_0=mu, sigma.sq_0=sigma, varsigma.sq=post_var[x,i], y=y)$z
      post_var[x,i+1]<-draw_var(alpha_0=a, beta_0=b, theta=post_mean[x,i+1], y=y)$z
    }
  }
post_mean2<-post_mean[,c(1001:10000)]
post_var2<-post_var[,c(1001:10000)]

df_mean = data.frame(iteration=rep(1:10000, times=3),
                         chain=as.character(rep(1:3, each=10000)),
                         mean=c(post_mean[1,], post_mean[2,], post_mean[3,]))

# mean traceplot
ggplot(df_mean %>% filter(iteration>1000))+
  geom_line(aes(x=iteration, y=mean, group=chain, color=chain), alpha=0.6)+
  labs(title="Posterior Mean Trace Plot", x="Iteration", y="Mean")
# mean histogram
allmean<-c(post_mean2)
hist(allmean,main="Sample Means (true mean = 100)",
     xlab="Mean Values")
abline(v=100)
df_var = data.frame(iteration=rep(1:10000, times=3),
                         chain=as.character(rep(1:3, each=10000)),
                         var=c(post_var[1,], post_var[2,], post_var[3,]))

# variance traceplot
ggplot(df_var %>% filter(iteration>1000))+
  geom_line(aes(x=iteration, y=var, group=chain, color=chain), alpha=0.6)+
  labs(title="Posterior Variance Trace Plot", x="Iteration", y="Variance")
# variance histogram
allvar<-c(post_var2)
hist(allvar,main="Sample Variance (true variance = 25)",
     xlab="Variance Values")
abline(v=25)
mean(allmean)
sd(allmean)
mean(allvar)
sd(allvar)
sd(allmean)
sd(y)/sqrt(length(y))
y<-rnorm(10,mean=100,sd=5)

post_mean<-matrix(NA,nrow=3,ncol=10)
post_var<-matrix(NA,nrow=3,ncol=10)

post_mean[,1]<-c(3,4,5)
post_var[,1]<-c(3,4,5)

```

```
a=.001
b=.001
mu<-0
sigma<-100

for(i in 1:9){
    for(x in 1:3){
      post_mean[x,i+1]<-draw_mean(mu_0=mu, sigma.sq_0=sigma, varsigma.sq=post_var[x,i], y=y)$z
      post_var[x,i+1]<-draw_var(alpha_0=a, beta_0=b, theta=post_mean[x,i+1], y=y)$z
    }
  }

post_mean2<-post_mean[,c(2:10)]
post_var2<-post_var[,c(2:10)]
mean(post_mean2)
sd(post_mean2)
mean(post_var2)
sd(post_var2)
y<-rnorm(100,mean=100,sd=5)

post_mean<-matrix(NA,nrow=3,ncol=100)
post_var<-matrix(NA,nrow=3,ncol=100)

post_mean[,1]<-c(3,4,5)
post_var[,1]<-c(3,4,5)

a=.001
b=.001
mu<-0
sigma<-100

for(i in 1:99){
    for(x in 1:3){
      post_mean[x,i+1]<-draw_mean(mu_0=mu, sigma.sq_0=sigma, varsigma.sq=post_var[x,i], y=y)$z
      post_var[x,i+1]<-draw_var(alpha_0=a, beta_0=b, theta=post_mean[x,i+1], y=y)$z
    }
  }

post_mean2<-post_mean[,c(11:100)]
post_var2<-post_var[,c(11:100)]
mean(post_mean2)
sd(post_mean2)
mean(post_var2)
sd(post_var2)
y<-rnorm(1000,mean=100,sd=5)

post_mean<-matrix(NA,nrow=3,ncol=1000)
post_var<-matrix(NA,nrow=3,ncol=1000)

post_mean[,1]<-c(3,4,5)
post_var[,1]<-c(3,4,5)

a=.001
```

```r
b=.001
mu<-0
sigma<-100

for(i in 1:999){
    for(x in 1:3){
      post_mean[x,i+1]<-draw_mean(mu_0=mu, sigma.sq_0=sigma, varsigma.sq=post_var[x,i], y=y)$z
      post_var[x,i+1]<-draw_var(alpha_0=a, beta_0=b, theta=post_mean[x,i+1], y=y)$z
    }
  }

post_mean2<-post_mean[,c(101:1000)]
post_var2<-post_var[,c(101:1000)]
mean(post_mean2)
sd(post_mean2)
mean(post_var2)
sd(post_var2)
y<-rnorm(10000,mean=100,sd=5)

post_mean<-matrix(NA,nrow=3,ncol=1000)
post_var<-matrix(NA,nrow=3,ncol=1000)

post_mean[,1]<-c(3,4,5)
post_var[,1]<-c(3,4,5)

a=.001
b=.001
mu<-0
sigma<-100

for(i in 1:999){
    for(x in 1:3){
      post_mean[x,i+1]<-draw_mean(mu_0=mu, sigma.sq_0=sigma, varsigma.sq=post_var[x,i], y=y)$z
      post_var[x,i+1]<-draw_var(alpha_0=a, beta_0=b, theta=post_mean[x,i+1], y=y)$z
    }
  }

post_mean2<-post_mean[,c(101:1000)]
post_var2<-post_var[,c(101:1000)]
mean(post_mean2)
sd(post_mean2)
mean(post_var2)
sd(post_var2)
set.seed(101)
y<-rnorm(100,mean=100,sd=5)

post_mean<-matrix(NA,nrow=3,ncol=100)
post_var<-matrix(NA,nrow=3,ncol=100)

post_mean[,1]<-c(3,4,5)
post_var[,1]<-c(3,4,5)

a=.001
```

```r
b=.001
mu<-0
sigma<-100

for(i in 1:99){
    for(x in 1:3){
      post_mean[x,i+1]<-draw_mean(mu_0=mu, sigma.sq_0=sigma, varsigma.sq=post_var[x,i], y=y)$z
      post_var[x,i+1]<-draw_var(alpha_0=a, beta_0=b, theta=post_mean[x,i+1], y=y)$z
    }
  }

post_mean2<-post_mean[,c(11:100)]
post_var2<-post_var[,c(11:100)]
mean(post_mean2)
sd(post_mean2)
mean(post_var2)
sd(post_var2)
set.seed(10)
y<-rnorm(100,mean=100,sd=5)

post_mean<-matrix(NA,nrow=3,ncol=10000)
post_var<-matrix(NA,nrow=3,ncol=10000)

post_mean[,1]<-c(3,4,5)
post_var[,1]<-c(3,4,5)

a=.001
b=.001
mu<-0
sigma<-100

for(i in 1:9999){
    for(x in 1:3){
      post_mean[x,i+1]<-draw_mean(mu_0=mu, sigma.sq_0=sigma, varsigma.sq=post_var[x,i], y=y)$z
      post_var[x,i+1]<-draw_var(alpha_0=a, beta_0=b, theta=post_mean[x,i+1], y=y)$z
    }
  }

post_mean2<-post_mean[,c(2:10000)]
post_var2<-post_var[,c(2:10000)]

# mean traceplot
df_mean = data.frame(iteration=rep(1:10000, times=3),
                         chain=as.character(rep(1:3, each=10000)),
                         mean=c(post_mean[1,], post_mean[2,], post_mean[3,]))

# mean traceplot
ggplot(df_mean %>% filter(iteration>1))+
  geom_line(aes(x=iteration, y=mean, group=chain, color=chain), alpha=0.6)+
  labs(title="Posterior Mean Trace Plot", x="Iteration", y="Mean")
#variance traceplot
df_var = data.frame(iteration=rep(1:10000, times=3),
                         chain=as.character(rep(1:3, each=10000)),
```

```
                          var=c(post_var[1,], post_var[2,], post_var[3,]))

# variance traceplot
ggplot(df_var %>% filter(iteration>1))+
  geom_line(aes(x=iteration, y=var, group=chain, color=chain), alpha=0.6)+
  labs(title="Posterior Variance Trace Plot", x="Iteration", y="Variance")

mean(allmean)
sd(allmean)

mean(allvar)
sd(allvar)
set.seed(10)
y<-rnorm(100,mean=100,sd=5)
post_mean<-matrix(NA,nrow=3,ncol=10000)
post_var<-matrix(NA,nrow=3,ncol=10000)

post_mean[,1]<-c(3,4,5)
post_var[,1]<-c(3,4,5)

a=.001
b=.001
mu<-0
sigma<-100

for(i in 1:9999){
    for(x in 1:3){

      post_var[x,i+1]<-draw_var(alpha_0=a, beta_0=b, theta=post_mean[x,i], y=y)$z
      post_mean[x,i+1]<-draw_mean(mu_0=mu, sigma.sq_0=sigma, varsigma.sq=post_var[x,i+1], y=y)$z
    }
  }

post_mean2<-post_mean[,c(2:10000)]
post_var2<-post_var[,c(2:10000)]

mean(allmean)
sd(allmean)

mean(allvar)
sd(allvar)
# this R markdown chunk generates a code appendix
```