ADVANCED BAYESIAN MODELING - ASSIGNMENT 3

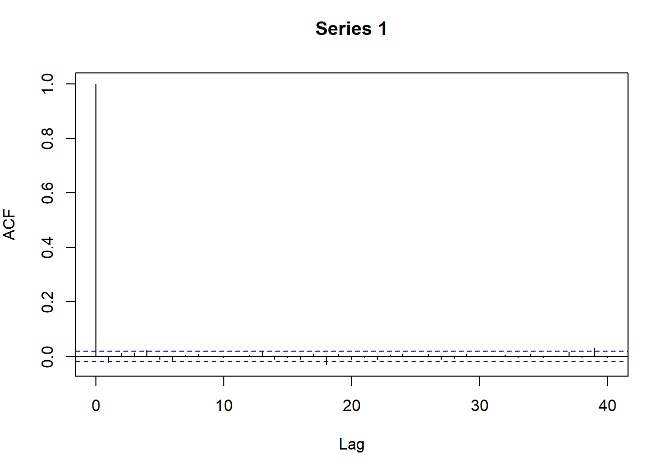
Shyam Shah

14/03/2021

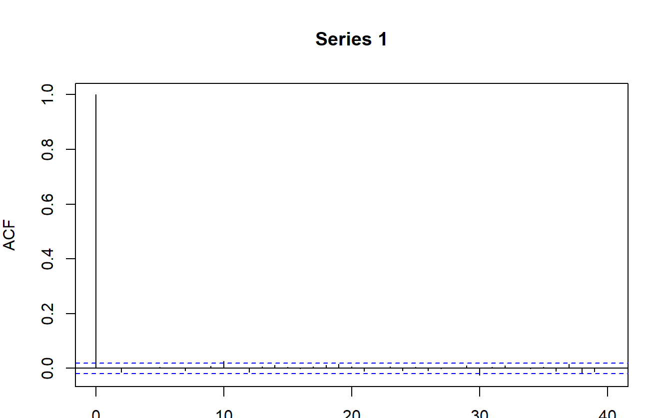
1 (a)

The code to simulate the posterior for μ and σ^2 and produce an autocorrelation plot is below.

```
### Gibbs sampler for (partially conjugate) analysis of Flint data

n <- 271
ybar <- 1.40
s.2 <- 1.684


mu0 <- 1.10
tau.2.0 <- sigma.2.0 <- 1.17
nu0 <- 1


mun <- function(sigma.2){
  (mu0/tau.2.0 + n*ybar/sigma.2) / (1/tau.2.0 + n/sigma.2)
}

tau.2.n <- function(sigma.2){
  1 / (1/tau.2.0 + n/sigma.2)
}

sigma.2.n <- function(mu){
  (nu0*sigma.2.0 + (n-1)*s.2 + n*(mu-ybar)^2) / (nu0 + n)
}


n.sim <- 10000
mu.sim <- numeric(n.sim)
sigma.2.sim <- numeric(n.sim)

mu.sim[1] <- 1.4         # starting value
sigma.2.sim[1] <- 1.7    # starting value

for(t in 2:n.sim){
  mu.sim[t] <- rnorm(1, mun(sigma.2.sim[t-1]),
                     sqrt(tau.2.n(sigma.2.sim[t-1])))

  sigma.2.sim[t] <- 1 / rgamma(1, (nu0+n)/2,
                              (nu0+n)*sigma.2.n(mu.sim[t])/2)
}

acf(matrix(mu.sim, ncol=1))
```




```
acf(matrix(sigma.2.sim, ncol=1))
```



These are the autocorrelation plots for the Gibbs sampler

1 (b) (i)

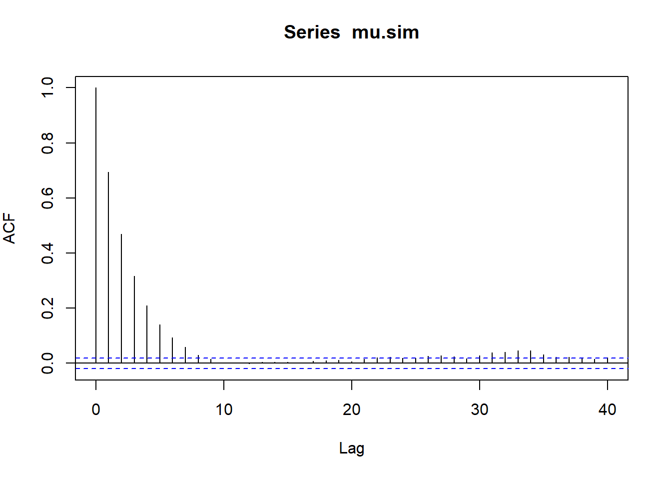
```
### Metropolis sampler for analysis of Flint data

n <- 271
ybar <- 1.40
s.2 <- 1.684


mu0 <- 1.10
tau.2.0 <- sigma.2.0 <- 1.17
nu0 <- 1


dinvchisq <- function(x, df, scalesq){  # inverse chi-square density
  if(x>0)
    ((df/2)^(df/2) / gamma(df/2)) * sqrt(scalesq)^df * x^(-(df/2+1)) *
      exp(-df*scalesq/(2*x))
  else 0
}

likelihood <- function(mu, sigma.2){
  sigma.2^(-n/2) * exp(-(n-1)*s.2/(2*sigma.2)) *
    exp(-n*(mu-ybar)^2/(2*sigma.2))
}

ratio <- function(mu.prop, sigma.2.prop, mu.old, sigma.2.old){
  dnorm(mu.prop,mu0,sqrt(tau.2.0)) * dinvchisq(sigma.2.prop,nu0,sigma.2.0) *
    likelihood(mu.prop,sigma.2.prop)  /
  (dnorm(mu.old,mu0,sqrt(tau.2.0)) * dinvchisq(sigma.2.old,nu0,sigma.2.0) *
    likelihood(mu.old,sigma.2.old))
}


n.sim <- 10000
mu.sim <- numeric(n.sim)
sigma.2.sim <- numeric(n.sim)
accept.prob <- numeric(n.sim-1)

rho <- 0.03

mu.sim[1] <- 1.4         # starting value
sigma.2.sim[1] <- 1.7    # starting value

for(t in 2:n.sim){
  mu.prop <- rnorm(1, mu.sim[t-1], sqrt(rho))
  sigma.2.prop <- rnorm(1, sigma.2.sim[t-1], sqrt(rho))

  accept.prob[t-1] <-
    min(ratio(mu.prop,sigma.2.prop,mu.sim[t-1],sigma.2.sim[t-1]), 1)
  if(runif(1) < accept.prob[t-1]){
    mu.sim[t] <- mu.prop
    sigma.2.sim[t] <- sigma.2.prop
  }else{
    mu.sim[t] <- mu.sim[t-1]
    sigma.2.sim[t] <- sigma.2.sim[t-1]
  }
}


mean(accept.prob)
```

```
## [1] 0.360908
```

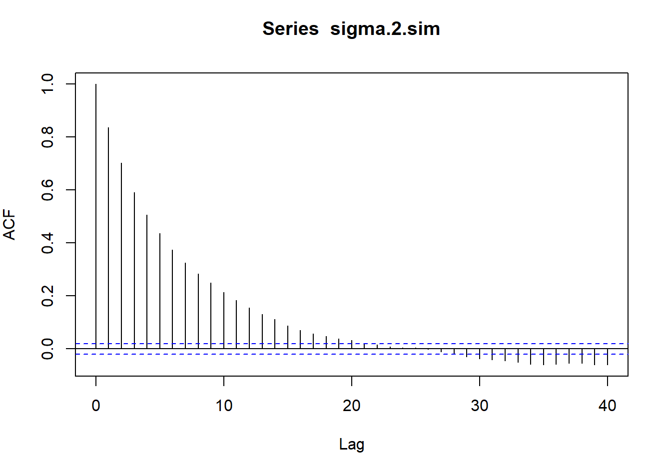
The value found for ρ is 0.03.

#1 (b) (ii)

```
acf(mu.sim)
```




```
acf(sigma.2.sim)
```



These are the autocorrelation plots for the Metropolis sampler

1 (c)

From the two plots, we can see that the Gibbs sampler mixes faster since it's autocorrelation decays much faster.

2 (a) (i)

```
library(rjags)
```

```
## Loading required package: coda
```

```
## Linked to JAGS 4.3.0
```

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

```
# prepare the data
d <- read.table("./polls2016.txt", header=TRUE)
d$sigma <- d$ME/2

# create initialization lists
initial.vals1 <- list(list(mu=-100, tau=0.01),
                      list(mu=100, tau=0.01),
                      list(mu=-100, tau=100),
                      list(mu=100, tau=100))

# create the model
m1 <- jags.model("./polls2016.bug", d, initial.vals1, n.chains=4)
```

```
## Warning in jags.model("./polls2016.bug", d, initial.vals1, n.chains = 4): Unused
## variable "poll" in data
```

```
## Warning in jags.model("./polls2016.bug", d, initial.vals1, n.chains = 4): Unused
## variable "ME" in data
```

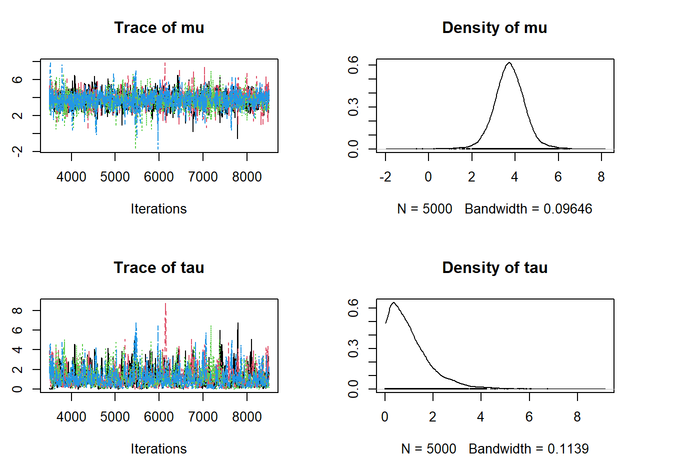
```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 7
##    Unobserved stochastic nodes: 9
##    Total graph size: 42
##
## Initializing model
```

2 (a) (ii)

```
# 2500 iterations of burn in
update (m1, 2500)

# 5000 iterations of monitoring
x1 <- coda.samples(m1, c("mu", "tau"), n.iter=5000)
```

#2 (a) (iii)

```
# display trace plots
plot (x1, smooth=FALSE)
```



Since all of the chains seem to be sampling from the same regions over all iterations, there does not seem to be any convergence problems.

2 (a) (iv)

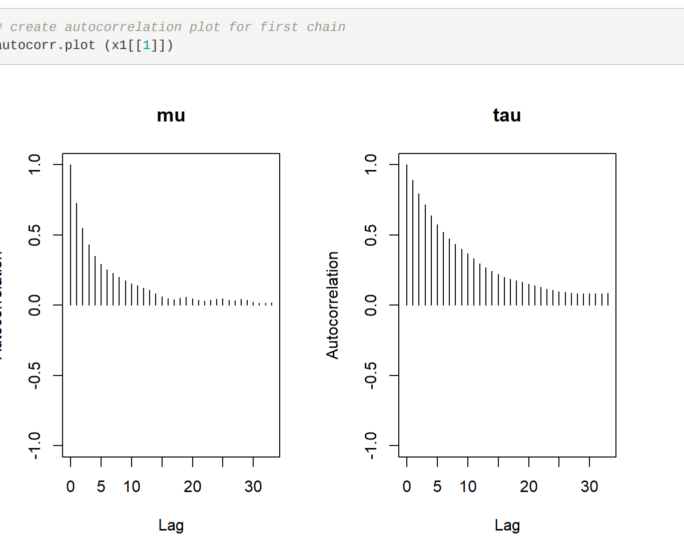
```
# get the Gelman-Rubin statistic
gelman.diag(x1, autoburnin=FALSE)
```

```
## Potential scale reduction factors:
##
##     Point est. Upper C.I.
## mu        1.01       1.01
## tau       1.00       1.01
##
## Multivariate psrf
##
## 1.01
```

The values under point est. represent the Gelman-Rubin statistic and since they are very close to 1 (and both less than 1.1), this shows that there were no convergence problems.

2 (a) (v)

```
# create autocorrelation plot for first chain
autocorr.plot (x1[[1]])
```



The autocorrelation plots for both mu and tau approach 0. So there don't appear to be any convergence problems.

2 (a) (vi)

```
# compute effective sample size
effectiveSize(x1)
```

```
##        mu       tau
## 2254.6800  877.8254
```

They both exceed the suggested minimum value of 400 so they would be considered adequate.

2 (b) (i)

This is what the new model looks like:

```
model {

  for (j in 1:length(y)) {
    y[j] ~ dnorm(theta[j], 1/sigma[j]^2)
    theta[j] ~ dnorm(mu, 1/tau^2)
  }

  mu ~ dunif(-1000,1000)
  tau <- exp(logtau)
  logtau ~ dunif(-100,100)

}
```

2 (b) (ii)

```
# create initialization lists with seeds because I saw lots of variaton
initial.vals2 <- list(list(mu=-100, logtau=log(0.01), .RNG.name="base::Wichmann-Hill", .RNG.seed=1),
                      list(mu=100, logtau=log(0.01), .RNG.name="base::Wichmann-Hill", .RNG.seed=2),
                      list(mu=-100, logtau=log(100), .RNG.name="base::Wichmann-Hill", .RNG.seed=3),
                      list(mu=100, logtau=log(100), .RNG.name="base::Wichmann-Hill", .RNG.seed=4))

# create the model
m2 <- jags.model("./polls20162.bug", d, initial.vals2, n.chains=4)
```

```
## Warning in jags.model("./polls20162.bug", d, initial.vals2, n.chains = 4):
## Unused variable "poll" in data
```

```
## Warning in jags.model("./polls20162.bug", d, initial.vals2, n.chains = 4):
## Unused variable "ME" in data
```

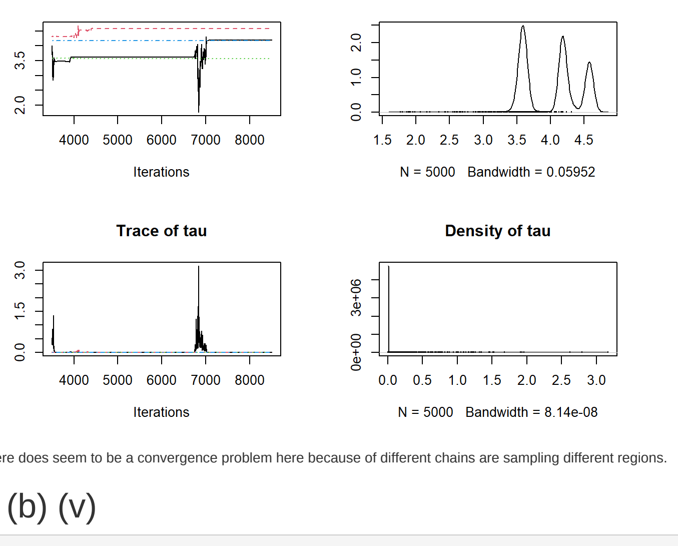
```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 7
##    Unobserved stochastic nodes: 9
##    Total graph size: 44
##
## Initializing model
```

2 (b) (iii)

```
# 2500 iterations of burn in
update (m2, 2500)

# 5000 iterations of monitoring
x2 <- coda.samples(m2, c("mu", "tau"), n.iter=5000)
```

2 (b) (iv)

```
# display trace plots
plot (x2, smooth=FALSE)
```



There does seem to be a convergence problem here because of different chains are sampling different regions.

2 (b) (v)

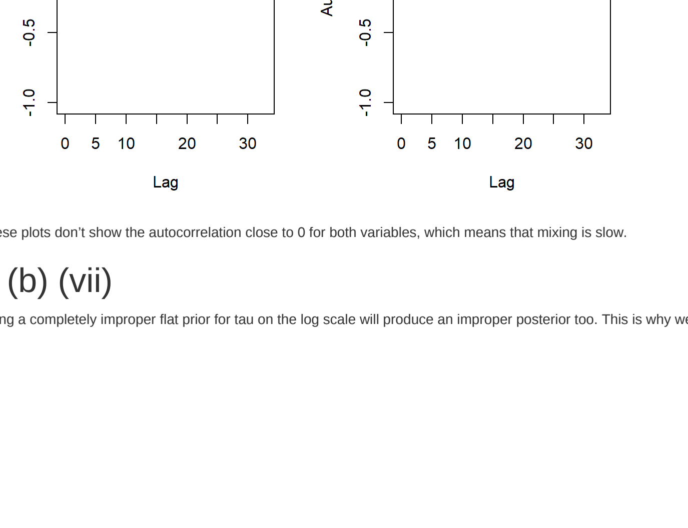
```
# get the Gelman-Rubin statistic
gelman.diag(x2, autoburnin=FALSE)
```

```
## Potential scale reduction factors:
##
##     Point est. Upper C.I.
## mu        3.88      17.87
## tau       1.31       2.05
##
## Multivariate psrf
##
## 3.74
```

This also indicates an issue with convergence since the Gelman-Rubin statistic is over the suggested 1.1 for both variables.

2 (b) (vi)

```
# create autocorrelation plot for first chain
autocorr.plot (x2[[1]])
```



These plots don't show the autocorrelation close to 0 for both variables, which means that mixing is slow.

2 (b) (vii)

Using a completely improper flat prior for tau on the log scale will produce an improper posterior too. This is why we don't see convergence.