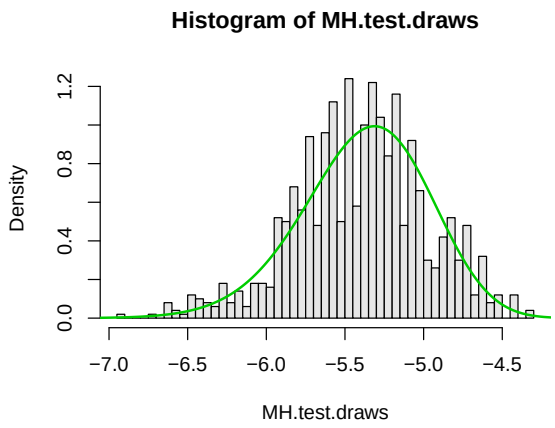
Bayes: Homework 8
Leslie Gains-Germain

1. (a) I modified the logit-normal code to draw candidate values for ϕ_j (rather than π_j) from a normal distribution centered at the current value of ϕ_j. I only printed the function that draws from the complete conditional distribution for ϕ_j because most of the rest of the code is the same as the tutorial (see appendix). Below, I show the histogram of draws for the complete conditional distribution for ϕ_1 to test the function. You can see the histogram of draws looks similar to the curve found by grid approximation.

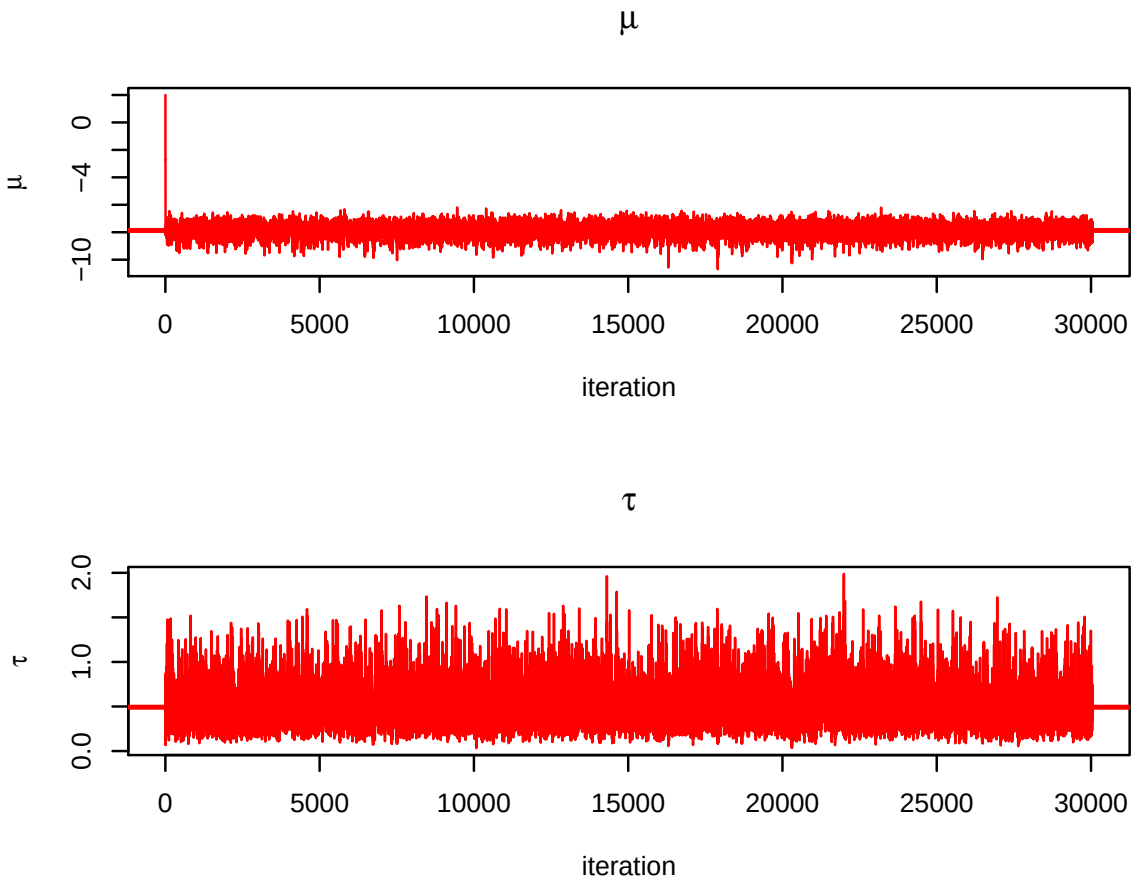
```
#### function that uses metropolis hastings to draw from complete conditional for phi_j
   MH.draw.cc.phi <- function(phi.cur, tuning, y.i, m, mu, tau){
    phi.cand <- rnorm(1, phi.cur, tuning)
    r.num.lcc <- y.i*log(expit(phi.cand)) + (m-y.i)*log(1-expit(phi.cand))+
                     dnorm(phi.cand, mu, sqrt(1/tau), log = TRUE)
    r.denom.lcc <- y.i*log(expit(phi.cur)) + (m-y.i)*log(1-expit(phi.cur))+
                     dnorm(phi.cur, mu, sqrt(1/tau), log = TRUE )
    r.num.ljump <- dnorm(phi.cur, phi.cand, tuning, log = TRUE)
    r.denom.ljump <- dnorm(phi.cand, phi.cur, tuning, log = TRUE)

    logr <- r.num.lcc + r.num.ljump - r.denom.lcc - r.denom.ljump
    ind.jump <- ifelse(runif(1) < exp(logr), 1, 0)
    phi.cur <- ifelse(ind.jump==1, phi.cand, phi.cur)
    return(c(phi.cur, ind.jump))
   }

#MH.draw.cc.phi(phi.cur=0, tuning=.8, y.i=0, m=917, mu=0, tau=1)

  #check M-H
    n.MH <- 1000
    MH.test.draws <- numeric(n.MH)
    MH.test.draws[1] <- -5
    for (k in 2:n.MH) {
        MH.test.draws[k] <- MH.draw.cc.phi(MH.test.draws[k-1], tuning=.8,
                                           y.i=y.data[1], m=m[1], mu=0, tau=1)[1]
      }
    hist(MH.test.draws, nclass=50, freq=FALSE, col=gray(0.9))
    lines(phi.grid, cc.phi.dens, col=3, lwd=2)
```

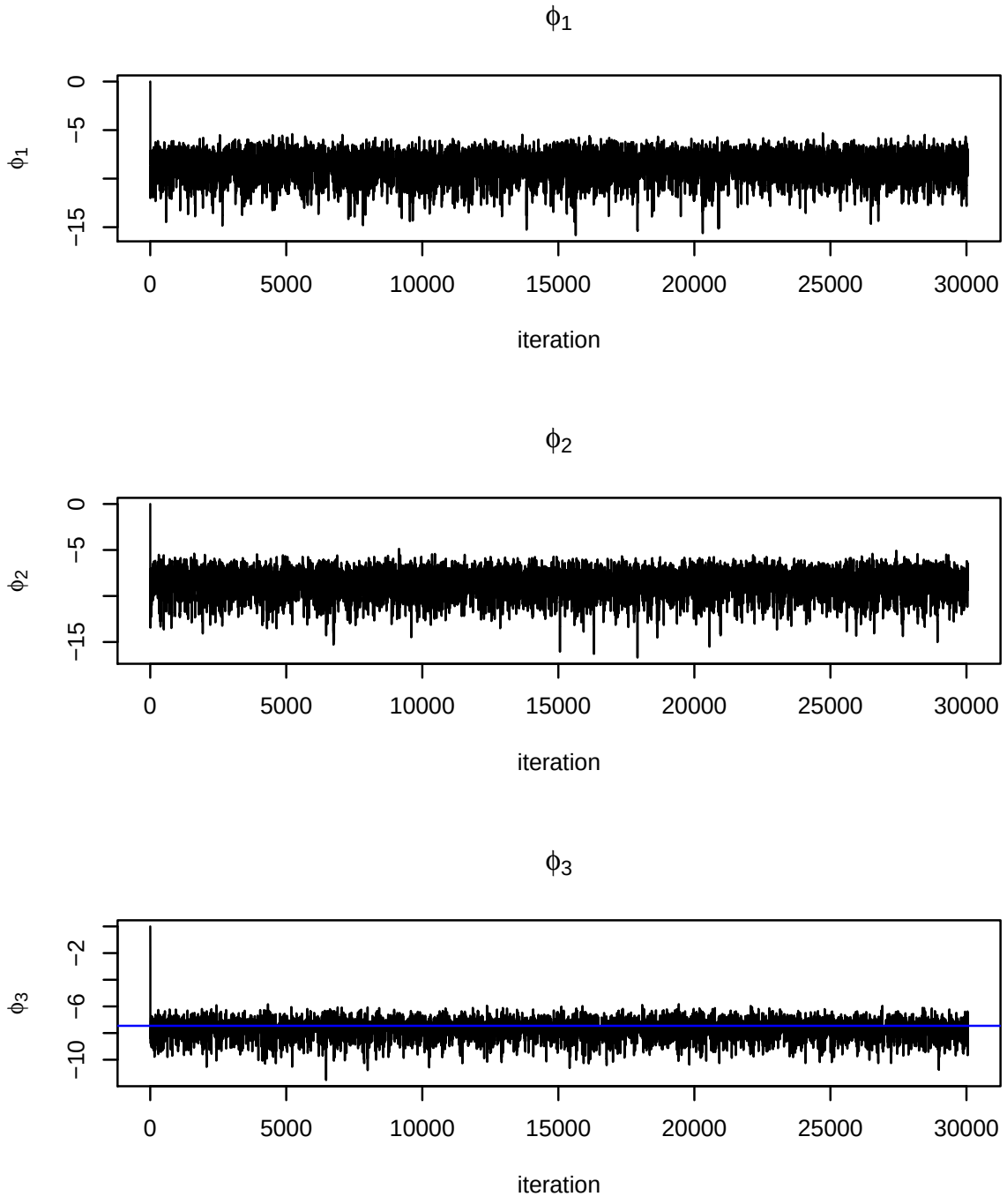


We are assuming that $\phi_i \sim N(\mu, \frac{1}{\tau})$. The prior on τ is $Gam(a_0, b_0)$. The prior on $\mu|\tau$ is $N(\mu_0, \frac{1}{\kappa_0 \tau})$. For the hyperparameter values, I chose $\mu_0 = -10$. I chose this value because Allison, Claire, and I looked online for the rate of death due to stomach cancer, and I found on worldlifexpectancy.com that the rate of death of stomach cancer in the US is 2.81/10000. This equals -10.45 on the logit scale, so it seemed reasonable to center the prior for μ at -10. I chose $\kappa_0 = 2$. I was thinking of κ_0 as the number of prior observations, so I figured I would choose a larger value of κ_0 if I wanted the prior to be more informative. I chose 2 because with a sample size of 20, the prior would be about 1/10th as informative as the data.

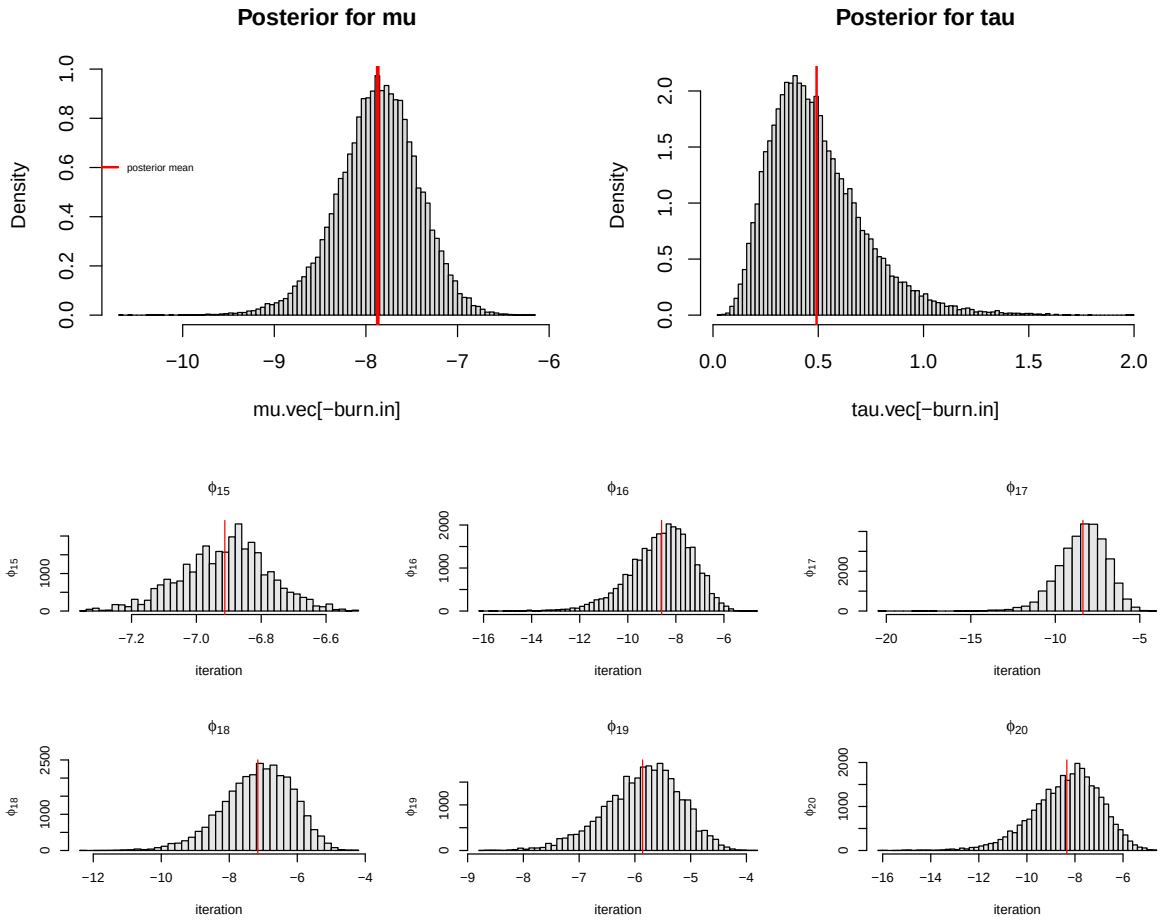
At first, I tried to specify a weakly informative prior for τ. I looked at the spread of stomach cancer death rates around the world, and I tried to use this information to think about what reasonable spreads would be for death rates among the counties in Missouri. But then, I started thinking about that maybe the spread of death rates due to stomach cancer among the world's countries aren't representative of the spread of death rates among Missouri's counties. In the end, I realized that I actually don't have much prior knowledge about what the spread of death rates should be among the counties in Missouri. Who knows? Maybe there is a stomach cancer epidemic in Missouri! As a result, I decided to use a non-informative $Gam(0.01, 0.01)$ prior for τ.

30050 iterations were run, and the traceplots are shown below for parameters $\mu, \tau, \phi_1, \phi_2$, and ϕ_3. Convergence looks good (I also checked the traceplots for all 20ϕ's). It looks like a small burn-in period of 50 iterations will be adequate for all parameters.

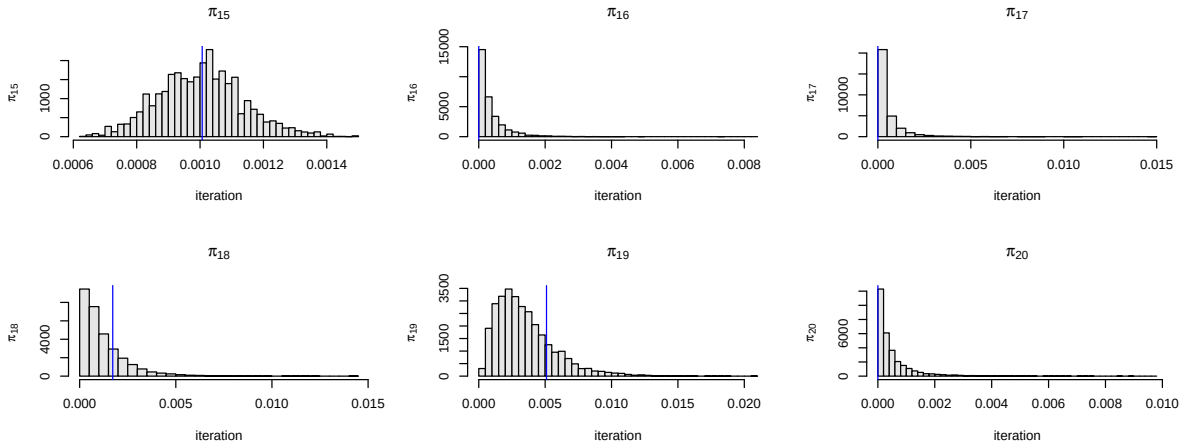




The histograms of posterior draws are shown below for μ, τ, $\phi_{15}, \phi_{16}, \phi_{17}, \phi_{18}, \phi_{19}$, and ϕ_{20} are shown below.



I transformed the posterior draws from the logit scale to the probability scale, and the histogram of posterior draws for $\pi_{15}, \pi_{16}, \pi_{17}, \pi_{18}, \pi_{19}$, and π_{20} are shown below.



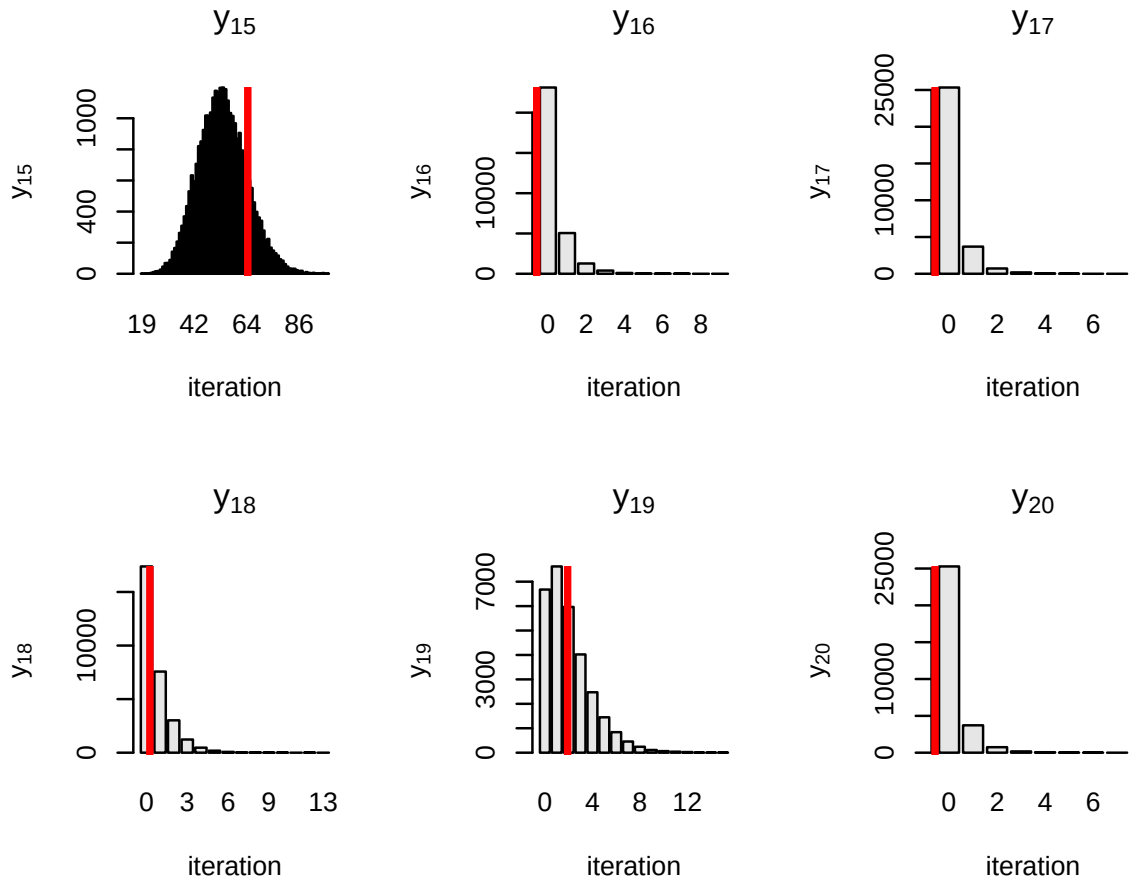
I noticed a few things in these plots. First, the posterior draws for the π_i's that had zero observed data values were bumped up against 0. The posterior draws for the π_i's that

had non-zero observed data values look more symmetric, but are more right skewed the closer the count is to 0. The Missouri county 15 had the largest posterior probabilities of death due to stomach cancer, and they were between 0.00067 and 0.00153.

(b) The code I used to generate posterior predictive counts from the model is shown below. First, I transformed the posterior draws of the ϕ_i's to posterior draws of the π_i's. Then, for each posterior draw of each π_i, I drew a random binomial count.

```
pi.mat <- matrix(NA, nrow=length(phi.mat[-burn.in, 1]), ncol=length(y.data))

for(j in 1:length(y.data)){
  pi.mat[,j] <- expit(phi.mat[-burn.in, j])
}

y.sim <- matrix(NA, nrow=length(pi.mat[,1]), ncol=length(y.data))

for(j in 1:length(y.data)) {
  for(i in 1:length(pi.mat[,1])) {
    y.sim[i,j] <- rbinom(1, m[j], pi.mat[i,j])
  }
}
```

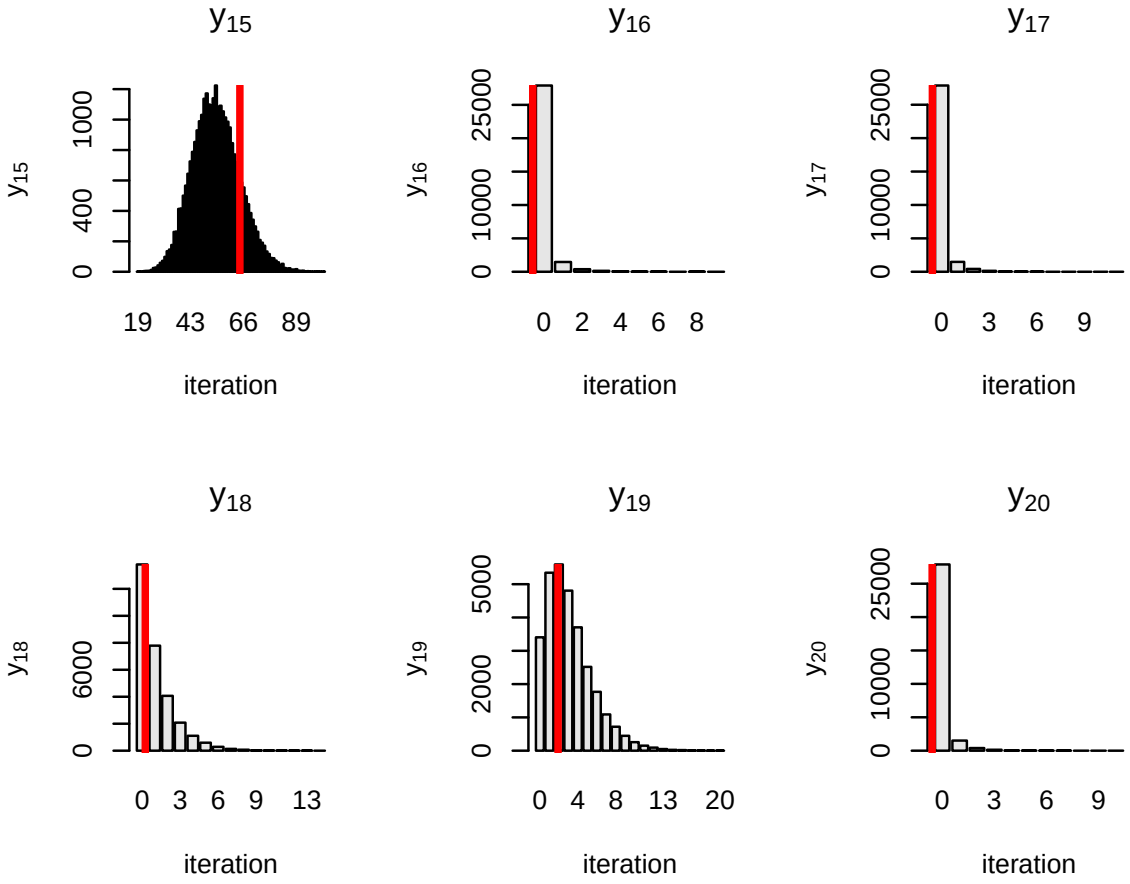
I show the barplots of posterior predictive draws below for the logit normal model. I only show the last six cities (cities $15 - 20$). These plots allow us to check whether the data observed could have been generated from the fitted model. The red lines on these plots are the observed counts.



The code I used to get the posterior predictive draws from the Beta-Binomial model is shown below. I used the π_i's generated from the JAGS beta binomial model (run in the extra credit part (f)) to get posterior predictive draws for these 20 counties.

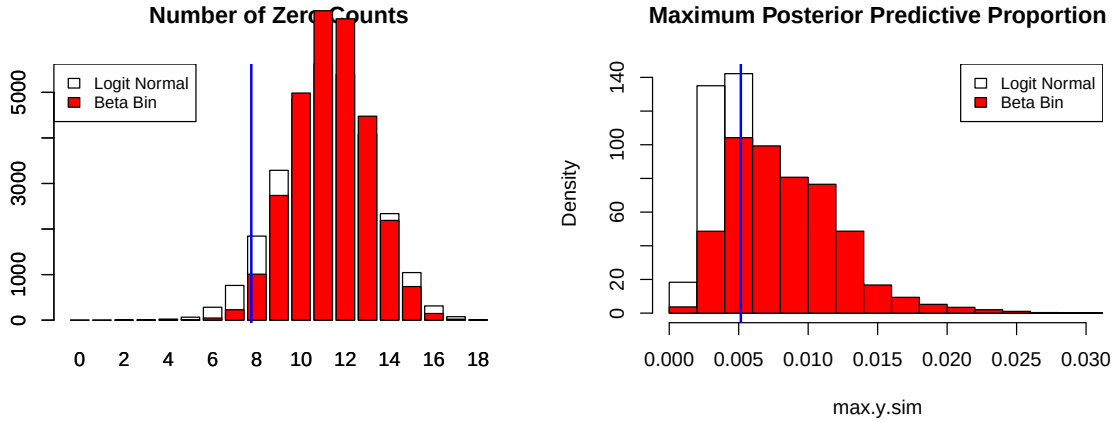
```
pi.bb.mat <- as.matrix(betabinom)[,3:22]

ybb.sim <- matrix(NA, nrow=length(pi.bb.mat[,1]), ncol=length(y.data))

for(j in 1:length(y.data)) {
  for(i in 1:length(pi.bb.mat[,1])) {
    ybb.sim[i,j] <- rbinom(1, m[j], pi.bb.mat[i,j])
  }
}
```

The barplots of posterior predictive draws are shown below for the Beta-Binomial model. Cities 15 through 20 are shown.



For cities with observed death counts of 0, the Beta Binomial model had more posterior predictive draws of 0 (and fewer posterior predictive draws of 1 and 2) than the logit normal model. For cities with non-zero observed death counts, the posterior predictive counts appeared to be larger, on average, than the predicted counts from the logit normal model.

For further comparison, I compared the maximum posterior predictive proportions in the Beta Binomial model to the logit normal model. I also compared the number of 0 counts in each posterior predictive draw across models.



Over all 30000 samples, it looks like the number of posterior draws of 0 in each sample was about the same in the logit normal model as the Beta Binomial model, although the Beta Binomial model had slightly more posterior predictive samples with 11 or 12 zeros. Also, it looks like the maximum posterior predictive proportion was higher, on average, for the Beta Binomial model.

(c) *Note: I would prefer if you grade this question after grading the whole problem (including the extra credit).*

I really wanted to use the logit normal model for inference, because for some reason I feel more comfortable with a model that assumes that the $logit(\pi_i)$'s are normally distribution rather than a model that assumes that the π_i's follow a Beta distribution. I think I feel this way just because I'm more familiar with the Normal distribution.

But, in the end, I felt like the Beta-Binomial model dealt with the large number of 0's in the dataset better than the logit normal model. If you look at my plots in the extra credit, you can see that more of the posterior draws of π_i's were closer to the observed proportions in the dataset. For counties with zero observed counts, the Beta-Binomial model had more small posterior draws of π_i's, and as a result the Beta Binomial model had more posterior predictive draws of 0 than the logit normal model.

I like comparing the posterior predictive draws of π_i's separately for each county. I don't like calculating a summary statistic for each of the groups of 20 counties as much, because I feel like this deprives the reader of the ability to compare what differs between the models for individual counties. For example, even though the number of zeros in the posterior predictive draws for the Beta Binomial model were about the same, overall, as the number of zeros in the posterior predictive draws from the logit normal model, this only occurred because the logit normal model predicted fewer zeros in cities with zero observed counts and more zeros in cities with non-zero observed counts. I never would have known this if I hadn't compared posterior predictive draws for individual counties.

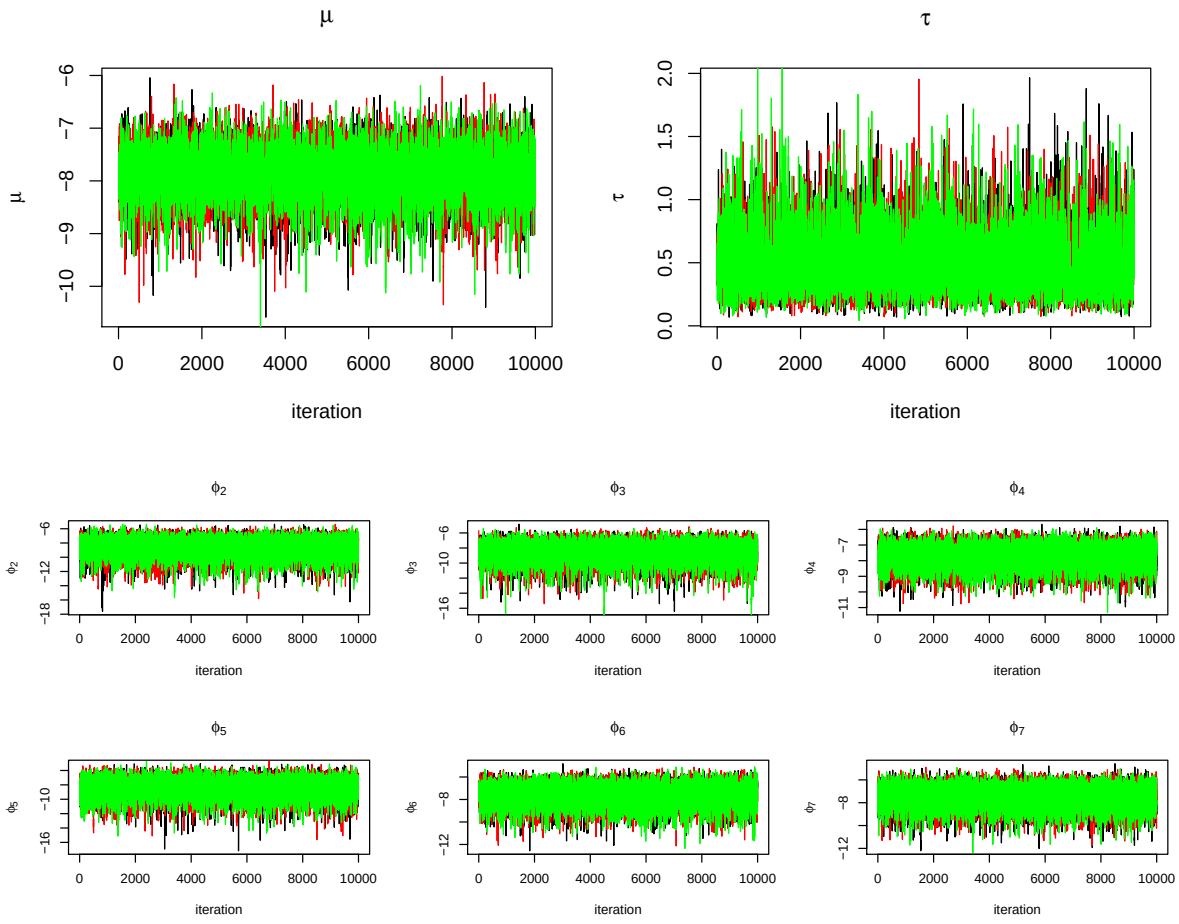
(d) The logit normal model, fit in JAGs, is shown below. I also showed the model call. Let me know if you have any suggestions about better (or different) ways to call the model.

```
##write model file first
cat("
model
{
for(i in 1:N)
{
logit(pi[i]) <- phi[i]
y[i] ~ dbin(pi[i], n[i])
phi[i] ~ dnorm(mu, tau)
}

mu ~ dnorm(mu.0, K.0*tau)
tau ~ dgamma(a.0, b.0)
sigmasq <- pow(tau, -1)

a.0 <- .01
b.0 <- .01
K.0 <- 2
mu.0 <- -10
}",
file="jags-stomachcancer.jags")
```

```
##jags call
library(R2jags)
set.seed(52)

#change to recache

stomach.data <- list(N=length(y.data), y=y.data, n=m)
```

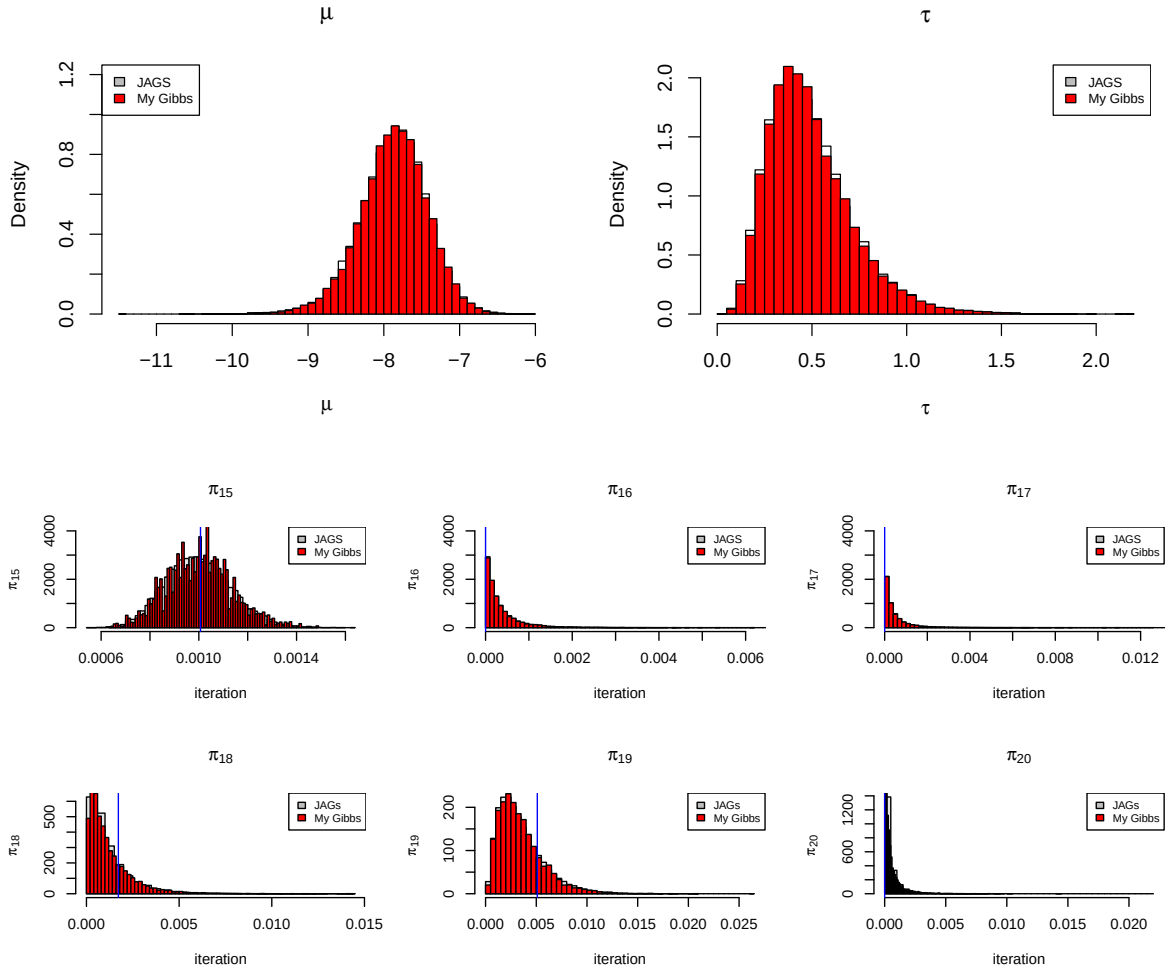
```
inits <- list(list(mu = -4, phi=c(-20:-1), tau = 0.6),
              list(mu = -6, phi=c(-1:-20), tau = 0.5),
              list(mu = -6, phi =c(-5:-10, -11:-20, -1:-4), tau = 0.3))
n.chain <- 3

#warmup
warmup.stomach <- jags.model("jags-stomachcancer.jags", data=stomach.data, n.chains=n.chain, inits =

#parameters to save
params <- c("phi", "pi", "mu", "tau", "sigmasq")

n.iter=10000
#running the model for real
samples <- coda.samples(warmup.stomach, params, n.iter=n.iter)
```

Below is a sample of the traceplots. Convergence looks good.

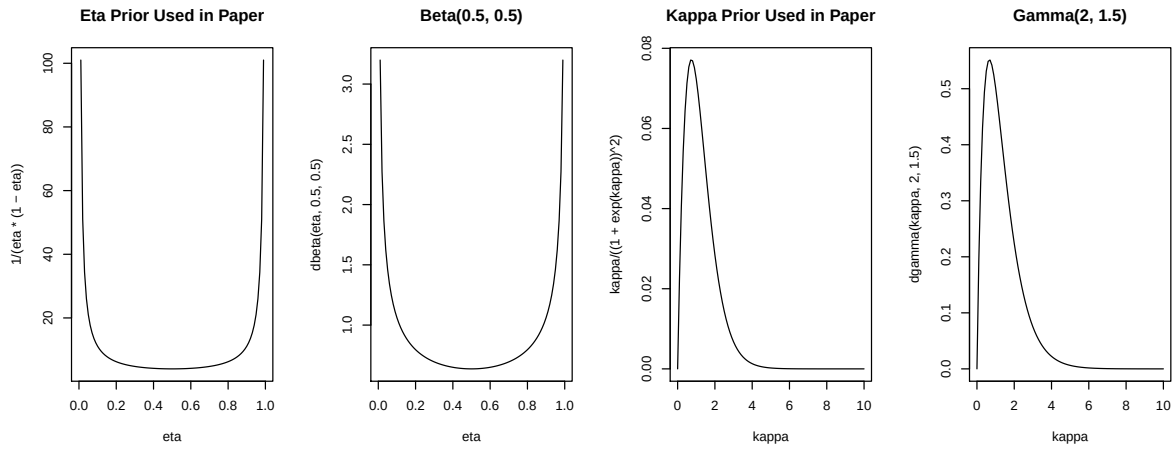


(e) Below, I overlaid the histogram of posterior draws for μ and the last six π_i's from the Gibbs sampler we programmed over the JAGs output. The JAGs draws looked almost identical to the draws from the gibbs sampler we programmed. With 30000 iterations

each, they should look almost exactly the same (at first they did not, and I realized I had a bug in my gibbs sampler).



(f) Below the JAGs code is shown that I used to fit the Beta-Binomial model. I found beta and gamma priors that looked similar to the priors used in the paper.

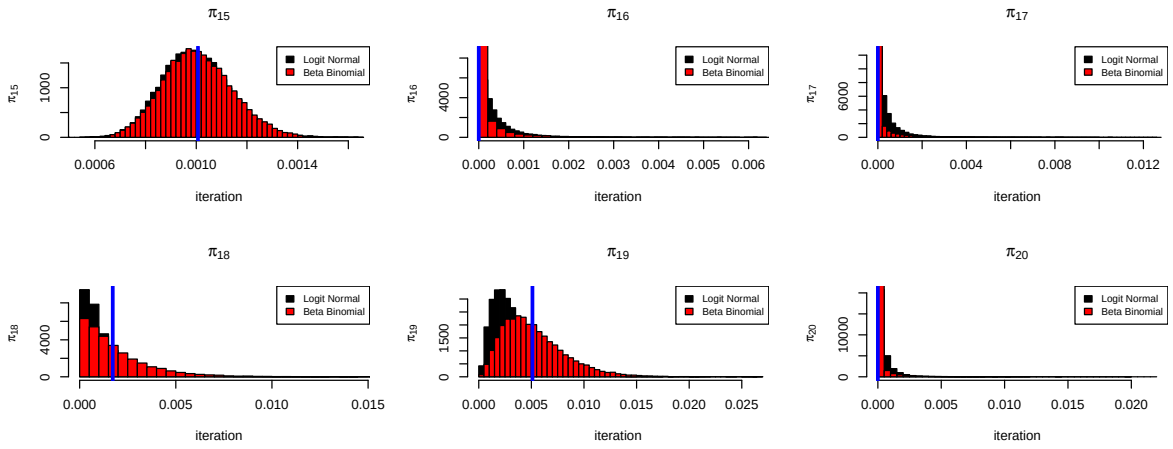



```
##write model file first
cat("
model
{
for(i in 1:N)
{
y[i] ~ dbin(pi[i], n[i])
pi[i] ~ dbeta(a.0, b.0)
}
a.0 <- kappa*eta
b.0 <- kappa*(1-eta)

eta ~ dbeta(.5, .5)
kappa ~ dgamma(2, 1.5)
}",
file="jags-betabinomial.jags")
```

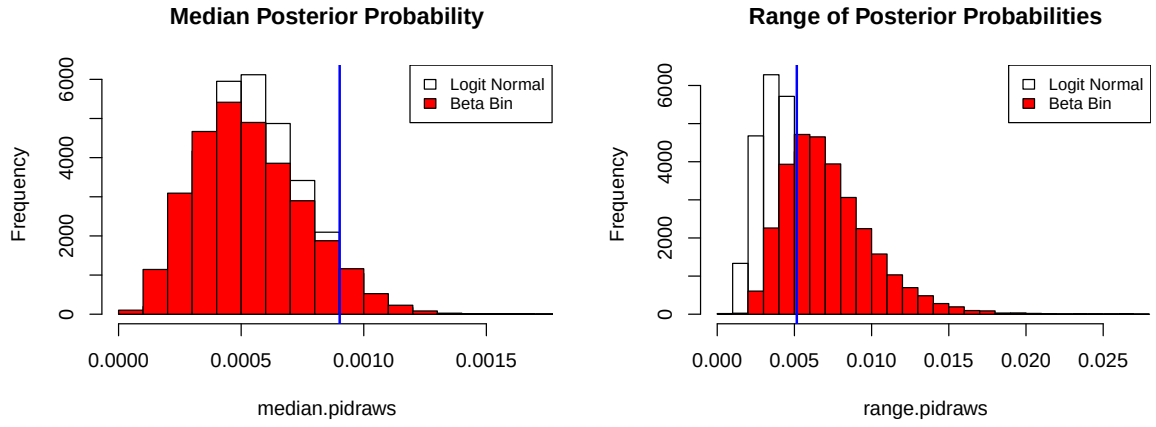
```
##jags call
library(R2jags)
set.seed(52)

stomach.data <- list(N=length(y.data), y=y.data, n=m)

inits <- list(list(pi=seq(0.0001, 0.2, length=20), eta = .00003, kappa = .7),
              list(pi=seq(0.2, 0.0001, length=20), eta = .2, kappa = 3),
              list(pi=c(seq(0.1, 0.2, length=10), seq(0.0001, 0.1, length=10)),
                        eta = .03, kappa = .1))
n.chain <- 3

#warmup
warmup.betabinom <- jags.model("jags-betabinomial.jags", data=stomach.data, n.chains=n.chain, inits=

#parameters to save
params <- c("pi", "eta", "kappa")

n.iter=10000
```

```
#running the model for real
betabinom <- coda.samples(warmup.betabinom, params, n.iter=n.iter)
```

The histogram of draws for parameters π_{15} through π_{20} are shown below. I overlaid the histogram of draws from the logit normal model fit using JAGs. This is similar to what I saw when doing the posterior predictive checks. For counties with observed counts of zero, the posterior probabilities were generally lower for the Beta Binomial model, but for counties with non-zero observed counts, the posterior probabilities were generally higher for the Beta Binomial model.



I also compared the median and range of the π_i's in this model to the logit normal model. It looks like the Beta Binomial model had lower median posterior probabilities on average, but the range of posterior probabilities was generally wider than the logit normal model.



2. (a)
   - How is a Bayesian hierarchical model different from a random effects model?
   - I know that you (Megan) have talked about how a random effect shouldn't be included if the variable was not random in the design (for example, in my and Kenny's experiment, row and block were not random in the design). Is this still something you think about when using a Bayesian hierarchical model?
   - I understand the "superpopulation variance" that Gelman refers to (σ_m^2) because in random effects models we think about and estimate this quantity in random effects models. But, I'm less familiar with the "finite population variances", s_m^2. I'm used to thinking about the sums of squares in the ANOVA table as the amount of variability that is explained by each term in the ANOVA table. I'm trying to relate this idea to s_m, the standard deviation of regression coefficients in a batch. (larger SD of regression coefficients in a batch means that more variability is explained by that factor?)
   - The main point I got out of this is that ANOVA is more important than ever because it helps us identify the batches of coefficients. But then he goes on to say, "The ideas of the analysis of variance also help us to include finite population and superpopulation inferences in a single fitted model, hence unifying fixed and random effects". I am struggling to understand this statement.
   - Do you think it's important to learn how to use ANOVA for complicated designs? (I am still unclear how to use ANOVA for complicated designs). Or do you think it's better to always turn to a random effects or hierarchical model for complicated designs?
   - He talks about ANOVA assuming exchangeable coefficients within each batch. I am confused about how this can be possible. Isn't the point to assess the effects at different levels of a factor?

   (b) The R code used to simulate data is shown below. I used the original way we used to simulate data to fit a model, and then I grabbed the model matrix. I then simulated

coefficients for each effect, and multiplied the simulated effects by the model matrix. Here's the code we used to simulate the data at first, and the model we fit. I looked at the model matrix to make sure it correctly displayed the nesting structure.

```
# Create some fake data with no differences anywhere
set.seed(983724)
fert.sim1 <- data.frame(expand.grid(
  "status2" = c("CNTL", "INOC"),
  "n.trt" = c("FALL", "EARLY SPRING", "LATE SPRING"),
  "variety" = c("Var1", "Var2", "Var3", "Var4", "Var5"),
  "row" = factor(1:30),
  "block" = LETTERS[1:6],
  "total" = 30),
  "infected" = rnbinom(180, 4, 7/9))
#  "infected" = sample(fert2$infected, 180, replace = TRUE))

glm3way <- glm(cbind(infected, total) ~ block + row +
                 variety*n.trt*status2,
               family = quasibinomial, data = fert.sim1)
```

```
#this grabs the model matrix from the three way interaction binomial model Kenny and I fit (so I dor
#the code for fitting the glm3way model is in the appendix
X <- model.matrix(glm3way)[1:180, 1:64]

set.seed(1132)

m <- 30

#Simulate all main effects and interactions
baseline <- runif(1, -7, -4)
block.adj <- rnorm(5, 0, 1)
variety.adj <- rnorm(4, 0, 1)
inoc.effect <- rnorm(1, 0, 1)
nit.effect <- rnorm(2, 0, 1)
rowinblock <- rnorm(29, 0, .5)
variety.nit <- rnorm(8, 0, 0.5)
variety.inoc <- rnorm(4, 0, 0.5)
nit.inoc <- rnorm(2, 0, 0.5)
variety.nit.inoc <- rnorm(8, 0, 0.5)

coefs <- c(baseline, block.adj, variety.adj, inoc.effect, nit.effect, rowinblock,
           variety.nit, variety.inoc, nit.inoc, variety.nit.inoc)

logitp <- X%*%coefs
p <- exp(logitp)/(1+exp(logitp))

plotcount.fun <- function(p, m=30){
  rbinom(1, m, p)
}
```

```
y.data <- apply(p, 1, plotcount.fun)
```

(c) The model is

$$y_i \sim \text{Binomial}(30, p_i),$$

$$\begin{aligned}\text{logit}(p_i) = \mu + block_{b[i]} + row_{b[i],r[i]} + \beta_{v[i]} + \gamma_{t[i]} + \delta_{j[i]} \\ + (\beta\gamma)_{v[i],t[i]} + (\beta\delta)_{v[i],j[i]} + (\gamma\delta)_{t[i],j[i]} + (\beta\gamma\delta)_{v[i],t[i],j[i]}\end{aligned}$$

where

- p_i is the probability of infection in the ith plot,
- $\text{logit}(p_i) = \log\left(\frac{p_i}{1-p_i}\right)$ is the natural logarithm of the odds of infection, and
- $b[i]$, $r[i]$, $v[i]$, $t[i]$, and $j[i]$ index the block, row, variety, nitrogen application timing, and inoculation status of plot i,
- $a_b \sim \text{N}(0, \sigma_a^2)$ and $(ab)_{b,r} \sim \text{N}(0, \sigma_{ab}^2)$ are random adjustments for each block and each row within each block.
- $i \in (1, .., 180)$, $b \in (1, ..., 6)$, $r \in (1, .., 5)$, $v \in (1, ..., 5)$, $t \in (1, 2, 3)$, $j \in (1, 2)$

3. (a) We are trying to make inference about μ, σ^2 where $y_i \sim t_\nu(\mu, \sigma^2)$ and ν is known. We introduce the auxiliary variables V_i and use the following model:

$$\begin{aligned} y_i \sim & N(\mu, V_i) \\ p(y_i|\mu, V_i) = & \frac{1}{\sqrt{2\pi V_i}} e^{\frac{y_i-\mu^2}{2V_i}} \\ V_i \sim & Inv-\chi^2(\nu, \sigma^2) \\ p(V_i|\nu, \sigma^2) = & \frac{(\nu/2)^{\nu/2}}{\Gamma(\nu/2)} \sigma^\nu V_i^{-(\nu/2+1)} e^{-\frac{\nu\sigma^2}{2V_i}} \end{aligned}$$

He says that he puts uniform priors on μ and σ^2, and going through his calculations for

the complete conditional distributions, I think he used the following reference priors:

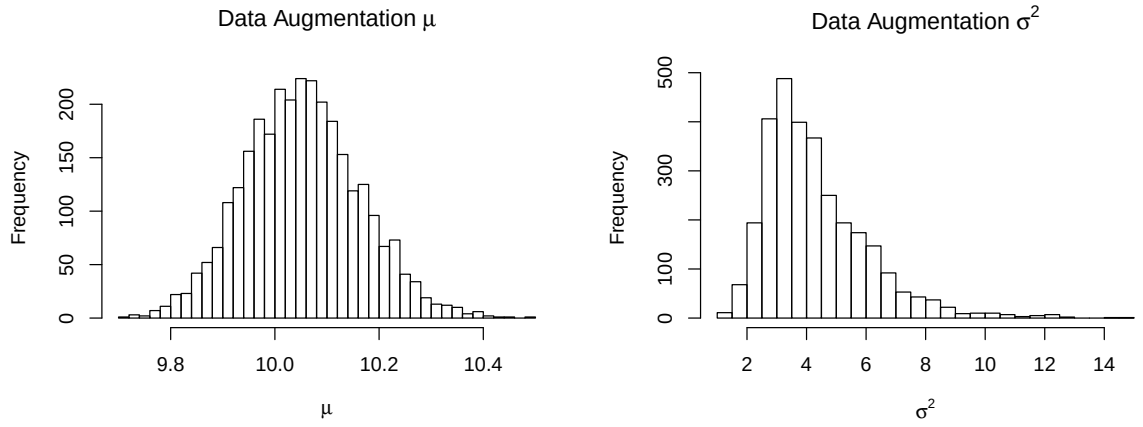
$$p(\mu) \propto 1$$
$$p(\sigma^2) \propto \frac{1}{\sigma^2}$$

(b) The code for the Gibbs sampler from the data augmentation approach is shown below.

```
expit <- function(x) {exp(x)/(1+exp(x))}
logit <- function(x) { log(x/(1-x))}

# Generate data under the model  ##
set.seed(459)
n <- 20
nu <- 30
y.data <- rt(n, df=nu, ncp=10)

##2. Set up vectors and matrices to store results in
n.gibbs <- 1000
nchain <- 3
mu.vec <- matrix(NA, nrow=n.gibbs, ncol=nchain)
sigmasq.vec <- matrix(NA, nrow=n.gibbs, ncol=nchain)
V.mat <- array(NA, dim=c(n.gibbs, length(y.data), nchain))

##3. Set initial values
mu.vec[1,] <- c(10.2, 9.8, 10.5)  #initial values for mu
sigmasq.vec[1,] <- c(1, 3, 15)  #initial values for sigmasq
V.mat[1,,1] <- rep(10, length(y.data)) #initial values for V_i's in chain 1
V.mat[1,,2] <- rep(5, length(y.data)) #initial values for V_i's in chain 2
V.mat[1,,3] <- rep(15, length(y.data)) #initial values for V_i's in chain 3

##4. NOW Set up the d steps for each iteration ##

for (k in 1:nchain){
for (t in 2:n.gibbs) {
 ## First update the vector of V_j's
 require(MCMCpack)
  for (j in 1:length(y.data)) {
    V.mat[t,j,k] <- rinvgamma(1, (nu+1)/2,
                              (nu*sigmasq.vec[t-1,k]+(y.data[j]-mu.vec[t-1,k])^2)/2)
   }

 #(b) Update mu
 mu.vec[t,k] <- rnorm(1, sum(y.data/V.mat[t,,k])/sum(1/V.mat[t,,k]),
                      1/sum(V.mat[t,,k]))


 #(a) Update sigmasq
 sigmasq.vec[t,k] <- rgamma(1, length(y.data)*nu/2, nu/2*sum(1/V.mat[t,,k]))
```

```
}
}
```

3 chains of 1000 posterior draws of μ and σ^2 from the Gibbs sampler coded using the data augmentation approach are shown below.

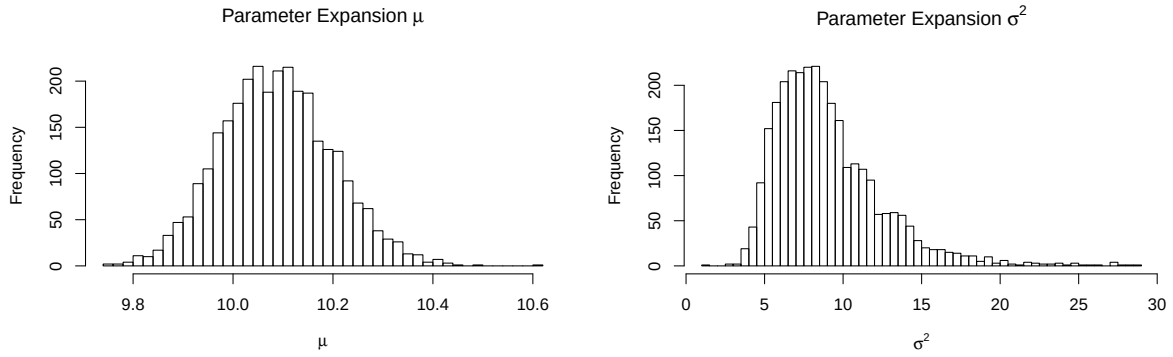


(c) The code for the Gibbs sampler from the parameter expansion approach is shown below.

```
##2. Set up matrices and arrays to store results in
n.gibbs <- 1000
nchain <- 3
mu.vec2 <- matrix(NA, nrow=n.gibbs, ncol=nchain)
sigmasq.vec2 <- matrix(NA, nrow=n.gibbs, ncol=nchain)
tausq.vec <- matrix(NA, nrow=n.gibbs, ncol=nchain)
alphasq.vec <- matrix(NA, nrow=n.gibbs, ncol=nchain)
U.mat <- array(NA, dim=c(n.gibbs, length(y.data), nchain))
V.mat2 <- array(NA, dim=c(n.gibbs, length(y.data), nchain))

##3. Set initial values
mu.vec2[1,] <-  c(10.2, 9.8, 10.5)  #initial value for mu
tausq.vec[1,] <- c(.1, .3, 1)
alphasq.vec[1,] <- c(4, 2, 3)
sigmasq.vec2[1,] <- c(1, 3, 15)   #initial value for sigmasq
U.mat[1,,1] <- rep(10, length(y.data)) #initial values for U_i's in chain 1
U.mat[1,,2] <- rep(20, length(y.data)) #initial values for U_i's in chain 2
U.mat[1,,3] <- rep(15, length(y.data)) #initial values for U_i's in chain 3

##4. NOW Set up the d steps for each iteration ##

for (k in 1:nchain) {
for (t in 2:n.gibbs) {
 ## First update the vector of U_j's
  for (j in 1:length(y.data)) {
    U.mat[t,j,k] <- rinvgamma(1, (nu+1)/2,
                              (nu*tausq.vec[t-1,k] +
```

```
                                (y.data[j]-mu.vec2[t-1,k])^2 /
                                alphasq.vec[t-1,k])/2)
    V.mat2[t,j,k] <- alphasq.vec[t-1,k]*U.mat[t,j,k]
   }

 # Update mu
 mu.vec2[t,k] <- rnorm(1, sum(y.data/V.mat2[t,,k])/sum(1/V.mat2[t,,k]),
                       1/sum(V.mat2[t,,k]))


 # Update tausq
 tausq.vec[t,k] <- rgamma(1, n*nu/2, nu/2*sum(1/U.mat[t,,k]))

 #update alphasq
 alphasq.vec[t,k] <- rinvgamma(1, n/2, sum((y.data-mu.vec2[t,k])^2/U.mat[t,,k]))

 #track sigmasq
 sigmasq.vec2[t,k] <- alphasq.vec[t,k]*tausq.vec[t,k]
}
}
```

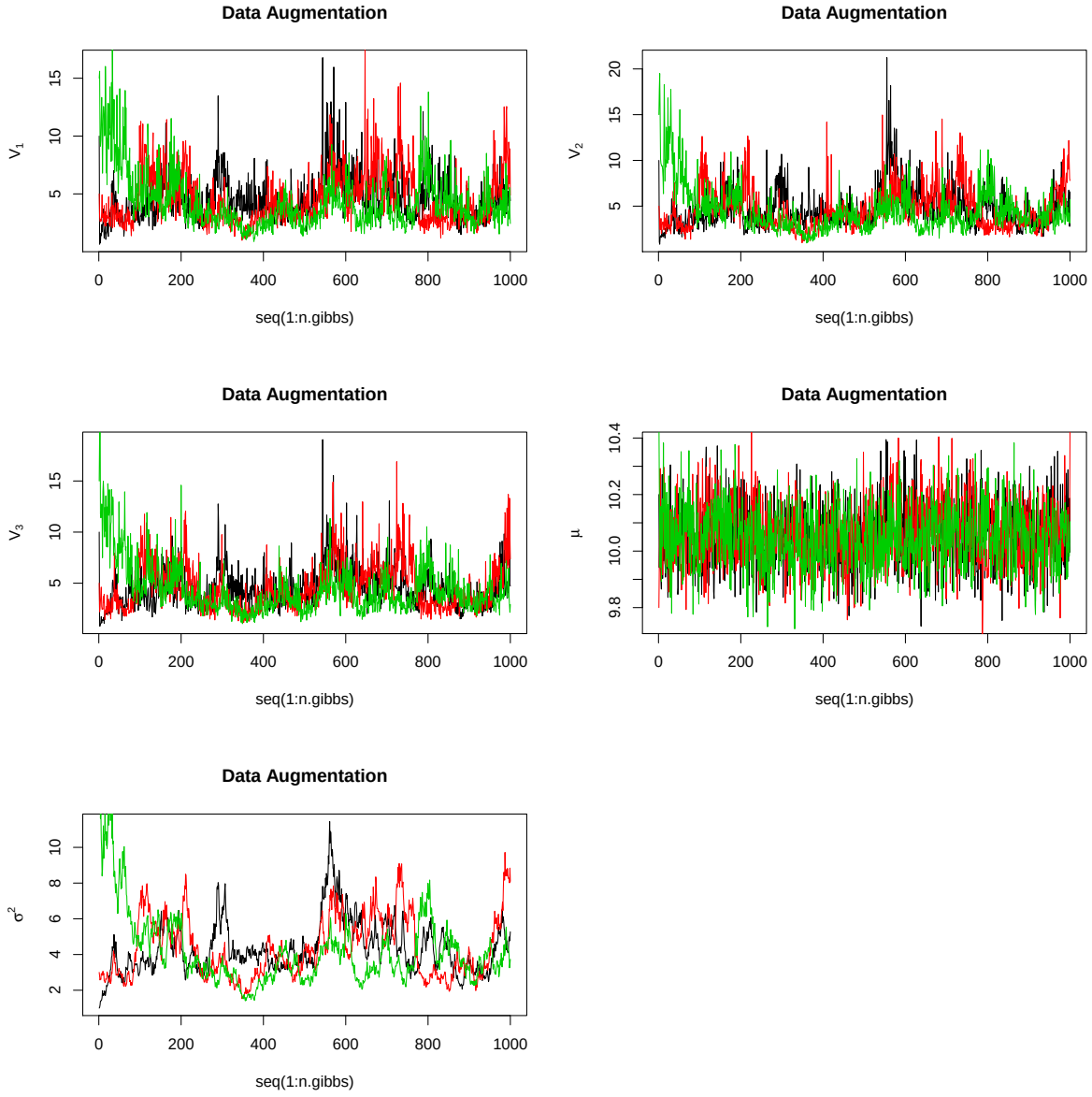
3 chains of 1000 posterior draws of μ and σ^2 from the Gibbs sampler coded using the parameter expansion approach are shown below.

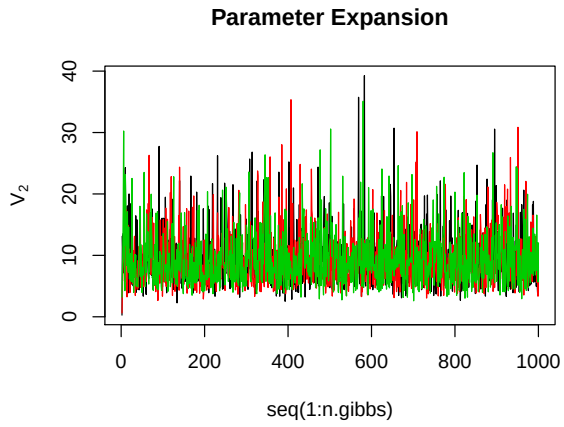


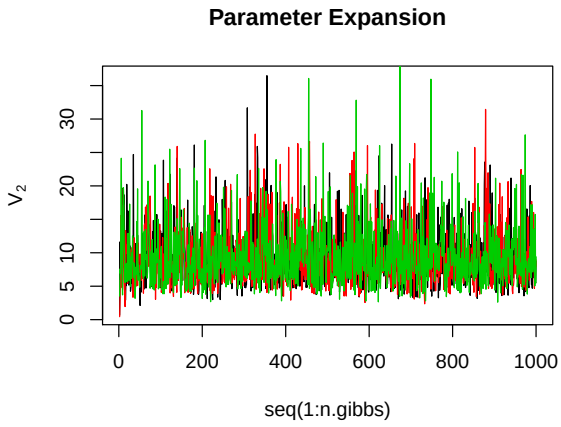
(d) It says in the book that we should monitor convergence of μ, σ^2, and V_i. The parameters α^2, U, and τ in the parameter expansion approach are not identifiable and we cannot estimate each of them, but convergence does appear to occur more quickly with the parameter expansion approach.

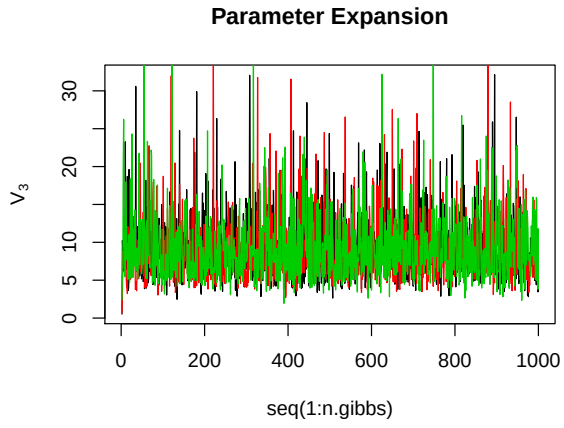
I only ran 1000 iterations in order to compare the two methods. For the data augmentation approach, the algorithm has not converged and 1000 iterations is clearly not

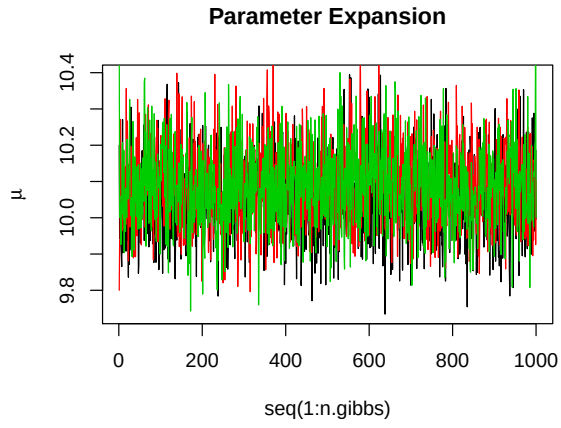
enough. The traceplot for μ looks good, but the traceplot for σ^2 looks like the chains have not mixed and more iterations are needed to obtain draws from the entire parameter space. In the traceplots for the first three V_i's, we also see that the chains have not mixed completely and more draws are needed to reach convergence. The traceplots from the parameter expansion method, however, indicate that convergence has been reached in only 1000 iterations, with good mixing among the chains and draws from the entire parameter space for all three chains.

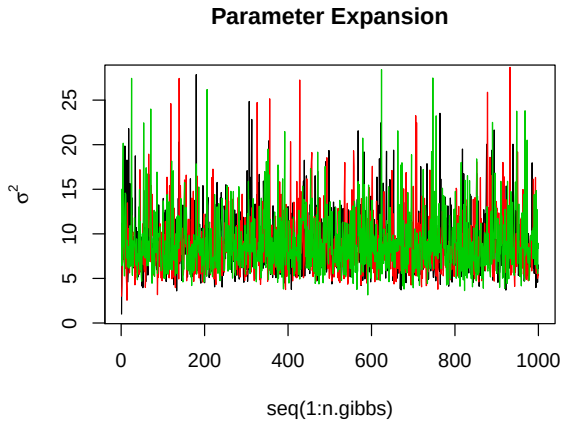








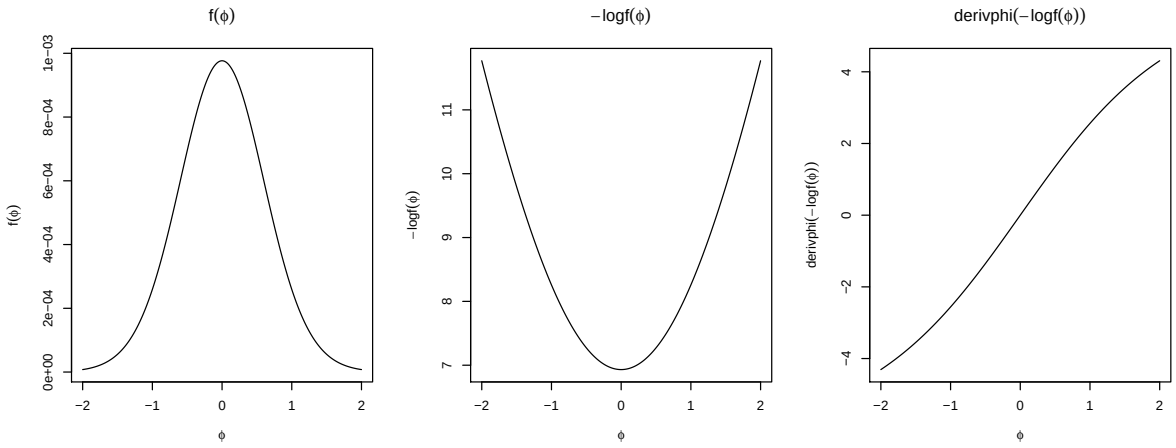




4. (a) Yes, I did read Sections 12.4 and 12.5, and I discussed them with Claire and Allison. The overall message was that the Gibbs and Metropolis algorithms are inefficient in the way that they move through the parameter space, and the HMC algorithm introduces the momentum parameter that is intended to speed convergence.

(b) The functions I wrote are shown below, as well as the plots.

```
postphi.fun <- function(phi, y, n, mu, sigma){
  exp(phi)^y*(1+exp(phi))^(-n)*exp(-(phi-mu)^2/(2*sigma^2))
}

dlogpost <- function(phi, y, n, mu, sigma){
  -y+n*exp(phi)/(1+exp(phi))+(phi-mu)/sigma^2
}
```



Watching the ball roll helped me understand how the momentum influences the jump from draw to draw. When moving towards lower density, the ball made smaller jumps, and when moving towards higher density, the ball made larger jumps. I think looking at the derivative of the negative log posterior function helped me understand that the momentum will be larger when moving towards areas of higher density and smaller when moving towards areas of lower density. Overall, I think it makes sense that the added momentum parameter, ϕ, helps the draws move more efficiently through the parameter space.

(c) Comments and questions:

- `samp1`, `samp2`, and `samp3` are S4 objects. They are not mcmc lists like the output returned by `coda.samples`. It seems convenient that you can use the extract function to get the draws for the parameter of interest.
- There is a traceplot function inside the `rstan` package similar to the one in the `coda`

package.

- The default burn-in is the number of iterations divided by 2. It's an adaptive burn in by default. If there is an adaptive warmup period, should we consider removing additional observations for a burn-in period after the adaptive warmup? It doesn't seem very set up to easily do this.
- In the linear model with the `mtcars` data, why do you have to specify beta[1] in the model code?
- Stan was written in C++.
- My biggest question is: where are the priors specified? I don't see this anywhere in the model code.
- The output is really similar to JAGs, with the same convergence diagnostics and summary measures.
- In the book, it said the algorithm used by STAN to obtain posterior draws is supposed to be more efficient than Gibbs sampling or the metropolis algorithm. But, it actually took longer for the models to run (with 2000 iterations) than a JAGs model would take to run with the same number of iterations. Is it that the efficiency isn't noticeable unless you are fitting a model with a large number of parameters?
- What is this piece for? `increment_log_prob(Ncens * exponential_ccdf_log(C, lambda));`?