Peer Review evaluation 2

Reviewer: Giorgio Paulon
Reviewee: Jesse Miller

Explanation of the exercises:

The .pdf file is clear and easy to read. The derivation of the full conditionals is exhaustive. We match on every result apart from the one that I already told you in person, that is, the full conditional for λ (or σ^2) has a shape parameter of n/2 and not of m/2, where m is the number of groups.

Modeling choice:

We used the same parametrization for the random effects model, but I have a couple of remarks:

- I think you should use an intercept (adding a column of 1's) and, if you want, the interaction term between display and logprice. The command model.matrix can be helpful in order to create the design matrix.

- Generalize the code for different X and Z matrices. It is easy to do it since you kept the terms well separated in your .pdf file

Coding tips:

As far as the coding is concerned, I have a few remarks that will hopefully help you:

- When coding, try to be as general as possible. Numbers should warn you: you should try to define variables, like p and q, that denote the dimension of the data structures instead of plugging in directly the number. In this way, if I give you a different dataset you do not have to go through all your code to change every line of it.

- In this example of hierarchical model, lists are very useful because they allow you to store different design matrices and observations for each store separately, and they are easy to call and to access in the Gibbs structure (see code attached).

- You do very well using crossprod anywhere you can: I just point out that instead of writing crossprod(X, X) you can just write crossprod(X) and the operation is the same: $X^T X$.

- As a general rule, I think it is a bad programming practice to initialize the chains to 0s or to 1s. I usually initialize them with NAs. In this way, if you're make some sort of error you will be more easily able to find it (see where the chain stopped and what is not working).

- Since we only use the precision matrix T^{-1} you can directly sample from the Wishart distribution so that you save a matrix inversion for each iteration of the Gibbs.

- Be careful when sampling from a multivariate normal distribution: you cannot use *rnorm* but you need to use the analogous *rmvnorm* in the *mvtnorm* package. Or you can use your own function ;)

Peer Review evaluation 2

Reviewer: Giorgio Paulon
Reviewee: Jesse Miller

- In order to speed up the performance of the algorithm, I still couldn't avoid the nested for looping (my code is not super fast either) but I improved your algorithm with the following tricks:
    - Storing T^{-1} instead of T to save a matrix inversion at each iteration
    - Precaching when possible, for example the sum of the $X^T X$ in the β update
    - Vectorized update of T^{-1} using a matrix operation

  This should not allow to run at least 10000 iterations more easily, and to use burnin and thinning to facilitate the convergence of the algorithm.

Peer Review evaluation 2

Reviewer: Giorgio Paulon
Reviewee: Jesse Miller

R code rewritten keeping your notation:

```
rm(list=ls())
setwd('~/Desktop/Semester 2/Statistical Modeling II/')
library(mvtnorm)

cheese_data <- read.csv("SDS383D-master/data/cheese.csv",header=TRUE)
stores <- unique(cheese_data[,1])
m <- length(stores)
n <- nrow(cheese_data)
log_cheese_data <- cheese_data
log_cheese_data[,2:3] <- log(log_cheese_data[,2:3])


# I would do a couple of things here:
# - first of all, I generalized the model in order for it to accept a different design
#   matrix for the fixed and for the random effects (call them X and Z)
# - secondly, I added an intercept term and an interaction term between display and
#   logprice

# Matrix of the fixed effects
X <- model.matrix(~ price + disp + disp:price, data = log_cheese_data)
# Matrix of the fixed effects
Z <- model.matrix(~ price + disp + disp:price, data = log_cheese_data)
# The model.matrix command is handy, you just specify the names of the columns and it
# creates the design matrix


# In order to be as general as possible, instead of using the "2" (dimension of your
# previous data structures), you can define p (number of fixed effects covariates) and
# q (number of random effects covariates)
p <- ncol(X)
q <- ncol(X)

# Store data and design matrices in a list: this will save you a lot of computation after
yi = list()
Xi = list()
Zi = list()
for (i in 1:m){
  yi[[i]] = log_cheese_data[log_cheese_data[,1] == stores[i], 3]
  Xi[[i]] = X[log_cheese_data[,1] == stores[i],]
  Zi[[i]] = Z[log_cheese_data[,1] == stores[i],]
}


# I added burnin and thinning to better understand the convergence of the chain
```

Peer Review evaluation 2

Reviewer: Giorgio Paulon
Reviewee: Jesse Miller

```
Gibbs_size <- 6000
burnin <- 1000
thin <- 1

# Create structures for parameters and initiaize their values: you will notice some
# small changements, I just tried to be as general as possible in the dimension
# specification. Moreover, I replaced your T matrix with Tinv. The explanation will be
# given later, but you don't need to store T at each interation since you are only
# using its inverse.
sigmasq <- rep(1,Gibbs_size)
B <- matrix(1, nrow = p, ncol = Gibbs_size)
g <- array(0, dim = c(m, q, Gibbs_size))
Tinv <- array(0,dim=c(q,q,Gibbs_size))
Tinv[,,1] <- diag(rep(1, q))

# Apart from these small changements, I encourage you to use NA's as initial values
# instead of 1 (that will eventually be overwritten). In this way, if there is an error
# in your code you will be able to find it. Initializing to 0's and 1's is a bad
# programming pratice I found out not a long time ago :)



# Hyperparamenters
nu <- 2
V <- diag(rep(1, q))

RSS <- rep(0, m) # used for sigmasq update
prod_for_T_update <- rep(0, m) # used for T update
sumXTX <- matrix(0, ncol=p, nrow=p) # used for B update
sumXTymXg <- rep(0, p) # used for B update

# Precache the computation of sumXTX
for(i in 1:m){
  sumXTX <- sumXTX + crossprod(Xi[[i]])
}

for (iter in 2:Gibbs_size){
  if(iter %% 10 == 0){print(iter)}

  # Update sigma2: the differences w.r.t your code are simply how I indexed the data
  # structures and the first parameter in the gamma distribution (should be the total
  # number of observations and not the total number of groups). And, if you want to
  # compute temp^T * temp, just use crossprod(temp) and not crossprod(temp, temp), you
  # can save some time ;)
  for (i in 1:m){
    temp <- yi[[i]] - as.numeric(Zi[[i]] %*% g[i,,iter-1]) -
      as.numeric(Xi[[i]] %*% B[,iter-1])
    RSS[i] <- crossprod(temp)
```

```
  }
  sigmasq[iter] <- 1/rgamma(1, n/2, sum(RSS)/2)


  # Update T:
  # - I vectorized your code because there is a simple matrix operation that
  #   leads to what you call prod_for_T_update.
  # - Moreover, instead of sampling from an inverse Wishart for T and inverting the
  #   matrix at each iteration, you can directly sample T^{-1} from a Wishart
  #   distribution which will be your precision matrix. Therefore you don't need T,
  #   but only what I called Tinv
  prod_for_T_update <- tcrossprod(g[i,,iter-1])
  Tinv[,,iter] <- rwish(nu + m, V + prod_for_T_update)


  # Update the gamma_i's:
  # - now that you have Tinv there is no need for the double inversion
  # - I did not understand why you were using a univariate normal distribution to sample
  #   the gammas. It should be 2-dimensional (or p-dimensional in this case)
  for(i in 1:m){
    Tstar <- solve(Tinv[,,iter] + (1/sigmasq[iter]) * crossprod(Zi[[i]]))
    kappastar <- (1/sigmasq[iter]) * Tstar %*% (crossprod(Zi[[i]], yi[[i]] - Xi[[i]] %*% B[,iter-1]))
    g[i,,iter] <- rmvnorm(1, kappastar, Tstar)
  }

  # Update Beta:
  # - you're sampling again from a univariate normal distribution
  # - be careful, you were summing over and over again during the Gibbs iterations!
  #   You need to reset sumXTymXg to 0 after each iteration!
  # - Moreover, sumXTX can be precached at the very beginning once for all!
  for(i in 1:m){
    sumXTymXg <- sumXTymXg + crossprod(Xi[[i]], yi[[i]] - Xi[[i]] %*% g[i,,iter])
  }
  Sigmastar <- solve(diag(rep(1, p)) + (1/sigmasq[iter]) * sumXTX)
  mustar <- Sigmastar %*% ((1/sigmasq[iter])*sumXTymXg)
  B[,iter] <- rmvnorm(1, mustar, Sigmastar)

  # Reset the auxiliary quantity sumXTymXg
  sumXTymXg <- rep(0, p)
}

# Let us take the burnin out and eventually thin the chain
sigmasq <- sigmasq[seq(burnin+1, Gibbs_size, by = thin)]
B <- B[,seq(burnin+1, Gibbs_size, by = thin)]
g <- g[,,seq(burnin+1, Gibbs_size, by = thin)]
Tinv <- Tinv[,,seq(burnin+1, Gibbs_size, by = thin)]
```

Peer Review evaluation 2

Reviewer: Giorgio Paulon
Reviewee: Jesse Miller

```
# ====================================
# Analyse the convergence via traceplots
# ====================================

# Sigma2
par(mar=c(4,4,2,2), mfrow = c(1,1), family = 'Palatino')
plot(sigmasq, type = 'l', xlab = 'Iterations', ylab = bquote(sigma^2))

# The four fixed effects beta_1, ..., beta_4
par(mar=c(4,4,2,2), mfrow = c(2,2), family = 'Palatino')
for (i in 1:4)
  plot(B[i,], type = 'l', xlab = 'Iterations', ylab = bquote(beta[.(i)]))

# The 88 random effects gamma_1, ..., gamma_m for the intercept
par(mar=c(2,4,1,1), mfrow = c(5,5), family = 'Palatino')
for (i in 1:25)
  plot(g[i,1,], type = 'l', xlab = 'Iterations', ylab = bquote(gamma[.(0)~.(i)]))


# ==============================
# Analyse the posterior estimates
# ==============================

# Compute the posterior point estimates
betas.post <- rowMeans(B)
gammas.post <- apply(g, c(1,2), mean)

# Draw the demand curves
cols <- c('dodgerblue2','firebrick2')
par(mar=c(2,2,1,1), mfrow = c(5,5), family = 'Palatino')
for (i in 1:25){
  xgrid <- seq(min(Xi[[i]][,2]), max(Xi[[i]][,2]), length.out = 100)
  plot(exp(Xi[[i]][,2]), exp(yi[[i]]), type = 'p', col = cols[Xi[[i]][,3]+1], pch = 16, cex = 0.8)
  lines(exp(xgrid), exp((betas.post[1] + gammas.post[i,1]) + xgrid * (betas.post[2] +
gammas.post[i,2])), col = cols[1], lwd = 2)
  lines(exp(xgrid), exp((betas.post[1] + betas.post[3] + gammas.post[i,1]+ gammas.post[i,3]) + xgrid
* (betas.post[2] + betas.post[4] + gammas.post[i,2] + gammas.post[i,4])), col = cols[2], lwd = 2)
}


# Draw the fitted lines
cols <- c('dodgerblue2','firebrick2')
par(mar=c(2,2,1,1), mfrow = c(5,5), family = 'Palatino')
for (i in 1:25){
  xgrid <- seq(min(Xi[[i]][,2]), max(Xi[[i]][,2]), length.out = 100)
  plot(Xi[[i]][,2], yi[[i]], type = 'p', col = cols[Xi[[i]][,3]+1], pch = 16, cex = 0.8)
  lines(xgrid, (betas.post[1] + gammas.post[i,1]) + xgrid * (betas.post[2] + gammas.post[i,2]), col =
cols[1], lwd = 2)
```

```
  lines(xgrid, (betas.post[1] + betas.post[3] + gammas.post[i,1]+ gammas.post[i,3]) + xgrid *
(betas.post[2] + betas.post[4] + gammas.post[i,2] + gammas.post[i,4]), col = cols[2], lwd = 2)
}


# Fit four stores that are peculiar (few data points for one of the categories)
cols <- c('dodgerblue2','firebrick2')
xgrid <- seq(min(X[,2]), max(X[,2]), length.out = 100)
par(mar=c(2,2,2,2), mfrow = c(2,2), cex = 0.8, family = 'Palatino')
for (i in c(1,9,18,34)){
  plot(Xi[[i]][,2], yi[[i]], type = 'p', main = as.character(cheese_data$store[i]), col = cols[Xi[[i]][,3]+1],
pch = 16, cex = 0.8, xlim = c(min(X[,2]), max(X[,2])), ylim = c(min(log_cheese_data$vol),
max(log_cheese_data$vol)))
  lines(xgrid, (betas.post[1] + gammas.post[i,1]) + xgrid * (betas.post[2] + gammas.post[i,2]), col =
cols[1], lwd = 2)
  lines(xgrid, (betas.post[1] + betas.post[3] + gammas.post[i,1]+ gammas.post[i,3]) + xgrid *
(betas.post[2] + betas.post[4] + gammas.post[i,2] + gammas.post[i,4]), col = cols[2], lwd = 2)
}


# Same four stores via OLS
cols <- c('dodgerblue2','firebrick2')
xgrid <- seq(min(X[,2]), max(X[,2]), length.out = 100)
par(mar=c(2,2,2,2), mfrow = c(2,2), cex = 0.8, family = 'Palatino')
for (i in c(1,9,18,34)){
  plot(Xi[[i]][,2], yi[[i]], type = 'p', main = as.character(cheese_data$store[i]), col = cols[Xi[[i]][,3]+1],
pch = 16, cex = 0.8, xlim = c(min(X[,2]), max(X[,2])), ylim = c(min(log_cheese_data$vol),
max(log_cheese_data$vol)))
  fit = lm(yi[[i]] ~ -1 + Xi[[i]])
  betas.lm <- fit$coefficients
  lines(xgrid, (betas.lm[1]) + xgrid * (betas.lm[2]), col = cols[1], lwd = 2)
  lines(xgrid, (betas.lm[1]+ betas.lm[3]) + xgrid * (betas.lm[2] + betas.lm[4]), col = cols[2], lwd = 2)
}
# The shrinkage effect is evident for the stores whose OLS estimates were distorted.
```