

# Simulation

Jiahao Cao

12/10/2024

## Simulation Example

This is an R Markdown document for the paper

### Bayesian Spatio-temporal Small Area Modeling: A Case Study of Estimating Late-Stage Melanoma Incidence in Texas.

Let  $t \in [T]$  denote the discrete time of interest, and let  $i \in [M]$  represent the geographical units (e.g., counties). For each diagnosed individual  $j \in [n_{it}]$  within area  $i$  at time  $t$ , let  $y_{itj}$  denote the stage of melanoma. Specifically,  $Y_{itj} = 1$  if the individual is in late-stage melanoma, and  $Y_{itj} = 0$  if the individual is in the early stage. At each time  $t$ , for each individual  $j$  in area  $i$ , let  $\mathbf{x}_{itj} \in \mathbb{R}^{p+1}$  denote the corresponding unit-level covariates, and let  $\mathbf{z}_{it} \in \mathbb{R}^q$  denote the area-level covariates. Additionally, the population is categorized into different strata  $\{1, 2, \dots, H\}$  based on cross-tabulated demographic variables such as sex and race/ethnicity. We use  $h_{itj} \in [H]$  and  $g_{itj}$  to represent the stratum and the age, respectively, of the individual  $j$  in area  $i$  at time  $t$ .

The proposed model takes the following form:

$$y_{itj} \mid p_{itj} \sim \text{Bernoulli}(p_{itj}),$$
$$\theta_{itj} := \text{logit}(p_{itj}) = \mathbf{x}_{itj}^\top \boldsymbol{\beta} + \mathbf{z}_{it}^\top \boldsymbol{\alpha} + G_{itj} + \delta_i + \epsilon_{it},$$

where  $\epsilon_{it}$  is a latent spatiotemporal process,  $\delta_i$  is a latent area-level effect, and  $G_{itj}$  represents the effect of an individual's age  $g_{itj}$ , given the corresponding stratum  $h_{itj}$  at time  $t$ . Our focus is on estimating the regression coefficients  $\boldsymbol{\beta} = (\beta_1, \dots, \beta_p)^\top$ ,  $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_q)^\top$ , as well as the age effect  $G$  across different times and strata.

We generate data according to the above model where the spatial domain contains  $M = 254$  Texas counties, and  $T = 10$  years are considered. Given each county  $i \in [M]$  and year  $t \in [T]$ ,  $n_{it} = 30$  observations are collected, with the population equally assigned into  $H = 3$  strata. Overall, we obtain 76,200 observations. We generate  $q = 20$  county-level covariates independently from a standard normal distribution. The regression coefficients are set to  $\alpha_i = 1$  if  $i \leq 2$  and  $\alpha_i = 0$  otherwise, mimicking a sparse regression scenario. For the basis functions, we select  $J = L = 8$ .

## 1. Data generation

```
library(sp)
library(nimble)
library(splines2)
# Read Texas county data
```

```

load('Data/polygons_list_bycounty.RData') # county_list, polygons_list_bycounty
load('Data/Adj_mat.RData')
Adj_mat = as.matrix(Adj_mat) # 254*254
source('Functions//Basis_functions.R')

n_it = 30 # number of samples in each county at each time
T = 10
num_group = 3
M = dim(Adj_mat)[1]
n = n_it*M*T*num_group
q = 20
q0 = 2

L = 8
degree = 3
J = 8

# Generate covariates
set.seed(0)
Z = array(rnorm(T*M*q), dim = c(T, M, q))
DATA = expand.grid(stage = rep(1, n_it/num_group), group = seq(num_group), county = seq(M), dxyear = seq(T))
DATA$age = runif(n)

county_design = DATA$county
h_design = DATA$group
t_design = DATA$dxyear
H = length(unique(h_design))

# Basis function
internal_nodes = as.vector(quantile(DATA$age,
                                     probs = seq(0,1,length.out=J-degree+1)[-c(1,J-degree+1)]))
Phi = Bspline.Basis(DATA$age, r = J, knots = internal_nodes, degree = degree)
MI = array(0, dim = c(T, M, L))
for(t in 1:T){
  MI[t,,] = MoransI.Basis(cbind(matrix(1, M, 1), Z[t,,]), r = L, A = Adj_mat)
}

# Generate coefficients
alpha = c(rep(1,q0),rep(0,q-q0))

xi = array(1, dim = c(H, T, J))
for (h in 1:H) { xi[h,,] = h }
B = 0.8*diag(J)
sigmasq_xi = 0.001
for (h in 1:H) { for (t in 2:T){ for(i_J in 1:J){
  xi[h, t, i_J] = rnorm(1, inprod(B[i_J, 1:J], xi[h, t - 1, 1:J]), sd = sqrt(sigmasq_xi))
}}}

eta = matrix(1, T, L)
A = 0.8*diag(L)
sigmasq_eta = 0.001
for (t in 2:T){for(i_L in 1:L){

```

```

    eta[t, i_L] = rnorm(1, inprod(A[i_L, 1:L], eta[t - 1, 1:L]), sd = sqrt(sigmasq_eta) )
  }}

delta = rep(0, M)

# Generate response
DATA$prob = rep(0, n)
for(i in 1:n){
  DATA$prob[i] = ilogit(inprod(Z[t_design[i], county_design[i], 1:q], alpha[1:q]) +
    inprod(Phi[i, 1:J], xi[h_design[i], t_design[i], 1:J]) +
    inprod(MI[t_design[i], county_design[i], 1:L], eta[t_design[i], 1:L]) +
    delta[county_design[i]])
  DATA$stage[i] = rbinom(1, 1, p = DATA$prob[i])
}
y = DATA$stage

save(DATA, Z, alpha, Phi, xi, MI, eta, delta, file = 'Data/data.RData')

```

## 2. Model Fitting

```

nimbleOptions(MCMCusePredictiveDependenciesInCalculations = TRUE)

code <- nimbleCode({
  for(i in 1:n){
    y[i] ~ dbern(prob[i])
    prob[i] <- ilogit(inprod(Z[t_design[i], county_design[i], 1:q], alpha[1:q]) +
      inprod(Phi[i, 1:J], xi[h_design[i], t_design[i], 1:J]) +
      inprod(MI[t_design[i], county_design[i], 1:L], eta[t_design[i], 1:L]) +
      delta[county_design[i]])
  }

  # Age AR
  for (h in 1:H) {
    for(i_J in 1:J) xi[h, 1, i_J] ~ dnorm(mu_xi[i_J], var = kappasq_1)
    for (t in 2:T){ for(i_J in 1:J){
      xi[h, t, i_J] ~ dnorm( inprod(B[i_J, 1:J], xi[h, t - 1, 1:J]), var = sigmasq_xi)
    }}
  }
  for(i_J in 1:J) mu_xi[i_J] ~ dnorm(0, var = kappasq_0)
  for(i_J in 1:J){
    B[i_J, i_J] ~ dnorm(1, var = kappasq_B)
    for(ii_J in (seq(from = 1, to = J)[-i_J])){ B[i_J, ii_J] ~ dnorm(0, var = kappasq_B) }
  }
  sigmasq_xi ~ dinvgamma(a_xi, b_xi)

  # Spatial AR
  for (t in 2:T){for(i_L in 1:L){
    eta[t, i_L] ~ dnorm( inprod(A[i_L, 1:L], eta[t - 1, 1:L]), var = sigmasq_eta)
  }}

```

```

for(i_L in 1:L) eta[1, i_L] ~ dnorm(0, var = kappasq_2)
for(i_L in 1:L){
  A[i_L, i_L] ~ dnorm(1, var = kappasq_A)
  for(ii_L in (seq(from = 1, to = L)[-i_L])){ A[i_L, ii_L] ~ dnorm(0, var = kappasq_A) }
}
sigmasq_eta ~ dinvgamma(a_eta, b_eta)

# alpha S-and-S
for (i_q in 1:q) {
  include[i_q] ~ dbern(0.5) # Spike-and-slab prior
  alpha_latent[i_q] ~ dnorm(0, var = kappasq_alpha)
  alpha[i_q] <- alpha_latent[i_q] * include[i_q]
}

# prior for delta
for(i in 1:M){
  delta[i] ~ dnorm(mu_delta, var = sigmasq_delta)
}
mu_delta ~ dnorm(0, var = 100)
sigmasq_delta ~ dinvgamma(1, 1)

})

kappasq_alpha = 100
kappasq_0 = 100
kappasq_1 = 100
kappasq_B = 100
a_xi = 1
b_xi = 1
kappasq_2 = 100
kappasq_A = 100
a_eta = 1
b_eta = 1

constants <- list(
  n = n,
  T = T,
  M = M,
  q = q,
  J = J,
  L = L,
  H = H,
  h_design = h_design,
  t_design = t_design,
  county_design = county_design,
  Z = Z,
  Phi = Phi,
  MI = MI,
  kappasq_alpha = kappasq_alpha, # regression coeff
  kappasq_0 = kappasq_0, kappasq_1 = kappasq_1, kappasq_B = kappasq_B, a_xi = a_xi, b_xi = b_xi,
  kappasq_2 = kappasq_2, kappasq_A = kappasq_A, a_eta = a_eta, b_eta = b_eta
)

```

```

inits <- list(
  alpha_latent = rep(0, q), include = rep(1,q),
  xi = array(0, dim = c(H, T, J)), mu_xi = rep(0, J), sigmasq_xi = 1, B = diag(J),
  eta = matrix(0, T, L), sigmasq_eta = 1, A = diag(L),
  mu_delta = 0, sigmasq_delta = 1, delta = rep(0, M)
)
data = list(
  y = y
)
model <- nimbleModel(code, constants = constants, data = data, inits = inits)

initInfo <- model$initializeInfo()
uninitializedNodes <- initInfo$uninitializedNodes
warnings <- initInfo$warnings
errors <- initInfo$errors
cat("\nWarnings:\n")
print(warnings)
cat("\nErrors:\n")
print(errors)

# Set up the MCMC configuration
mcmcConf <- configureMCMC(model)
mcmcConf$setMonitors('alpha_latent', 'alpha', 'xi', 'eta',
                     'include', 'delta', 'mu_delta', 'sigmasq_delta')
# Build and compile the MCMC
mcmc <- buildMCMC(mcmcConf)
Cmodel <- compileNimble(model)
Cmcmc <- compileNimble(mcmc, project = model)

n_batches <- 200
batch_size <- 200
n_batches_save = 40
num_thin <- 20
mcmc.out = runMCMC(Cmcmc, niter=batch_size*n_batches,
                   nburnin=batch_size*(n_batches - n_batches_save),
                   thin=num_thin, nchains=1, setSeed = TRUE, progressBar = TRUE)

# ===== Samplers =====
# RW sampler (594)
#   - eta[] (80 elements)
#   - alpha_latent[] (20 elements)
#   - xi[] (240 elements)
#   - delta[] (254 elements)
# conjugate sampler (140)
#   - mu_xi[] (8 elements)
#   - B[] (64 elements)
#   - sigmasq_xi
#   - A[] (64 elements)
#   - sigmasq_eta
#   - mu_delta
#   - sigmasq_delta
# binary sampler (20)

```

```

# - include[] (20 elements)
# thin = 1: alpha, alpha_latent, delta, eta, include, mu_delta, sigmasq_delta, xi

samples_save = as.matrix(mcmc.out)
# Calculate WAIC
waic_results = calculateWAIC(Cmcmc)
save(samples_save, waic_results, file = 'Data/MCMCsamples.RData')

## Prediction
## -----## -----## -----## -----## -----## -----## -----
load(file = 'Data/MCMCsamples.RData')
num_save = batch_size * n_batches_save / num_thin
age_range = c(10,100)

# Designs for prediction
age_grid = seq(from = 20, to = 90, by = 5)
age_grid = (age_grid - age_range[1]) / (diff(age_range))
num_G = length(age_grid)
phi_grid = Bspline.Basis(age_grid, r = J, knots = internal_nodes, degree = degree)

# Posterior prediction
posterior_params = as.data.frame(samples_save)

# Extracting alpha vector
alpha <- as.matrix(posterior_params[,grep("^alpha", colnames(posterior_params))])

# Extracting 3-d array xi
xi <- array(0, dim = c(num_save, H, T, J))
for(h in 1:H) {
  for(t in 1:T) {
    for(j in 1:J) {
      xi[, h, t, j] <- as.matrix(posterior_params[,paste0("xi[", h, ", ", t, ", ", j, "]")])
    }
  }
}

# Extracting matrix eta
eta <- array(0, dim = c(num_save, T, L))
for(t in 1:T) {
  for(l in 1:L) {
    eta[, t, l] <- as.matrix(posterior_params[,paste0("eta[", t, ", ", l, "]")])
  }
}

# Extracting delta
delta <- as.matrix(posterior_params[,grep("^delta", colnames(posterior_params))])

# county-level prediction
prob_summary_M_H_T_Age_MCMC = array(0, dim = c(num_save, M,H,T,num_G))
prob_summary_M_H_T_Age = array(0, dim = c(M,H,T,num_G))
prob_summary_M_H_T_Age_upper = array(0, dim = c(M,H,T,num_G))
prob_summary_M_H_T_Age_lower = array(0, dim = c(M,H,T,num_G))

```

```

# Make prediction
for(i in 1:M){
  temp_i = delta[, i, drop = FALSE]
  for (t in 1:T){
    temp_it = temp_i + alpha[, 1:q]%% as.vector(Z[t,i,1:q]) +
              eta[, t, 1:L]%%as.vector(MI[t,i,1:L])
    for(h in 1:H){for(i_age in 1:num_G){
      prob_summary_M_H_T_Age_MCMC[,i,h,t,i_age] =
        as.vector(temp_it + xi[, h, t, 1:J]%% as.vector(phi_grid[i_age, 1:J]))
    }}
  }
}
prob_summary_M_H_T_Age_MCMC = ilogit( prob_summary_M_H_T_Age_MCMC )
prob_summary_M_H_T_Age <- apply(prob_summary_M_H_T_Age_MCMC, c(2, 3, 4, 5), mean)
prob_summary_M_H_T_Age_upper <- apply(prob_summary_M_H_T_Age_MCMC,
                                       c(2, 3, 4, 5), quantile, probs = c(0.95))
prob_summary_M_H_T_Age_lower <- apply(prob_summary_M_H_T_Age_MCMC,
                                       c(2, 3, 4, 5), quantile, probs = c(0.05))
prob_summary_M_H_T_Age_average_mcmc <- apply(prob_summary_M_H_T_Age_MCMC, 1, mean)

# Make prediction to all individuals
Phi_Data = Bspline.Basis(DATA$age, r = J, knots = internal_nodes, degree = degree)
prob_summary_individuals_MCMC = matrix(0, num_save, n)
prob_summary_individuals = rep(0, n)
prob_summary_individuals_upper = rep(0, n)
prob_summary_individuals_lower = rep(0, n)
for(i_individual in 1:n){
  i = county_design[i_individual]
  t = t_design[i_individual]
  h = h_design[i_individual]
  prob_summary_individuals_MCMC[,i_individual] = delta[, i, drop = FALSE] +
    alpha[, 1:q]%% as.vector(Z[t,i,1:q]) + eta[, t, 1:L]%%as.vector(MI[t,i,1:L])
    as.vector(xi[, h, t, 1:J]%% as.vector(Phi_Data[i_individual, 1:J]))
}

prob_summary_individuals_MCMC = ilogit( prob_summary_individuals_MCMC )
prob_summary_individuals <- apply(prob_summary_individuals_MCMC, 2, mean)
prob_summary_individuals_upper <- apply(prob_summary_individuals_MCMC, 2, quantile, probs = c(0.95))
prob_summary_individuals_lower <- apply(prob_summary_individuals_MCMC, 2, quantile, probs = c(0.05))
prob_summary_individuals_average_mcmc = apply(prob_summary_individuals_MCMC, 1, mean)

save(prob_summary_M_H_T_Age, prob_summary_M_H_T_Age_upper,
     prob_summary_M_H_T_Age_lower, prob_summary_M_H_T_Age_average_mcmc,
     prob_summary_individuals, prob_summary_individuals_upper,
     prob_summary_individuals_lower, prob_summary_individuals_average_mcmc,
     file = 'Data/Posterior_summary_ALL.RData')

```

## Estimated Late-Stage Probability

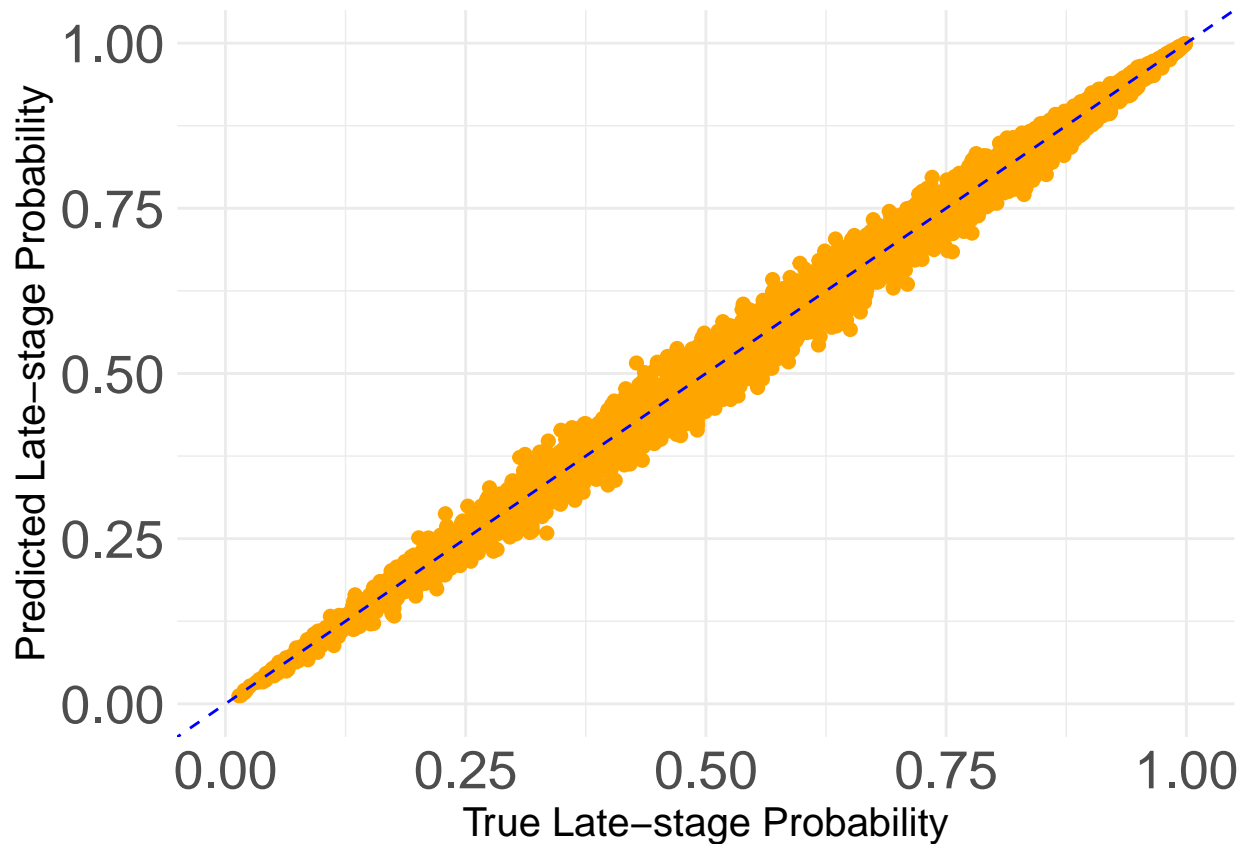
To investigate the estimation performance of the proposed model, we calculate the true average late-stage probabilities averaged over each cross-classification of county, year, and stratum, and compare them with their corresponding predictive values based on posterior mean estimates. The simulation results show that

the predictions match the true values very well, indicating that BMSTLM can accurately estimate the unobserved true late-stage probabilities.

```
library(tidyverse)
load(file = 'Data/data.RData')
load(file = 'Data/Posterior_summary_ALL.RData')
DATA_temp = DATA
DATA_temp$stage_pred = prob_summary_individuals
DATA_summary = DATA_temp %>% group_by(county, dxyear, group) %>%
  summarize(prob_truth = mean(prob, na.rm = TRUE), prob_pred = mean(stage_pred, na.rm = TRUE))
```

```
## 'summarise()' has grouped output by 'county', 'dxyear'. You can override using
## the '.groups' argument.
```

```
ggplot(DATA_summary, aes(x = prob_truth, y = prob_pred)) +
  geom_point(size = 2, color = 'orange') +
  geom_abline(intercept = 0, slope = 1, color = "blue", linetype = "dashed") +
  labs(x = "True Late-stage Probability", y = "Predicted Late-stage Probability") +
  xlim(0, 1) + ylim(0, 1) +
  theme_minimal() +
  theme(axis.title = element_text(size = 15), axis.text = element_text(size = 20),
        legend.position = NULL)
)
```

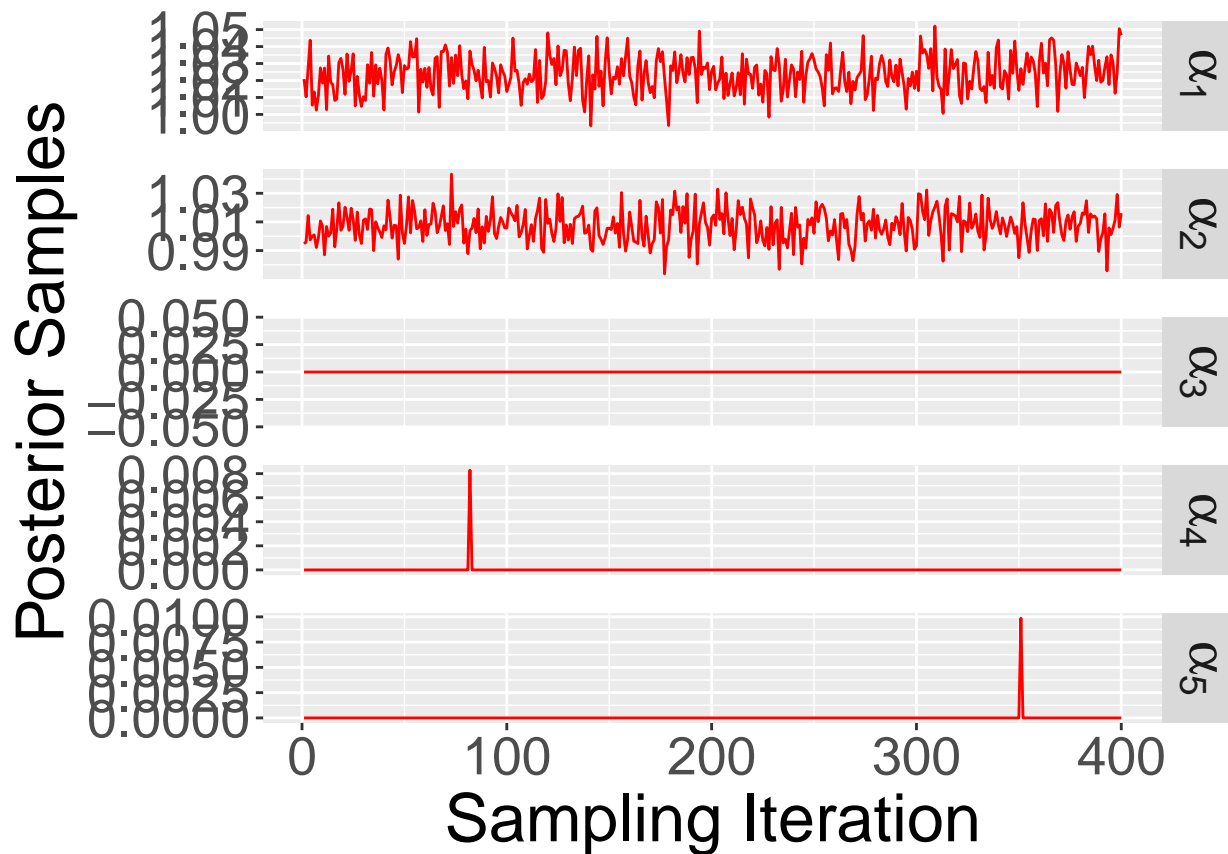




```
ggsave(file = 'Data/Prob_Overall_compare.pdf', width = 8, height = 6)
```

Finally, we present the trace plots for the estimates of the regression coefficients ( $\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5$ ). The trace plots indicate that BMSTLM accurately estimates the county-level regression covariates and succeeds in identifying non-zero coefficients.

```
load(file = 'Data/MCMCsamples.RData')
posterior_params = as.data.frame(samples_save)
mcmc_samples = posterior_params[,c('alpha[1]', 'alpha[2]', 'alpha[3]', 'alpha[4]', 'alpha[5]')]
df = data.frame(iter = seq(dim(mcmc_samples)[1]), value = as.vector(as.matrix(mcmc_samples)),
  variable = rep( c('alpha[1]', 'alpha[2]', 'alpha[3]', 'alpha[4]', 'alpha[5]'),
    each = dim(mcmc_samples)[1] ))
ggplot(df) + geom_path(aes(x = iter, y = value), color = 'red') +
  facet_grid(variable ~ ., scales = 'free_y', labeller = label_parsed) +
  labs(x = 'Sampling Iteration', y = "Posterior Samples")+
  theme(legend.key.size = unit(1, 'cm'),
    legend.title = element_text(size=20), legend.text = element_text(size=15)) +
  theme(axis.text=element_text(size=20), axis.title=element_text(size=25)) +
  theme(strip.text=element_text(size=20)) +
  theme(panel.spacing = unit(1, 'lines'))
```



```
ggsave(filename = 'Data/Traceplot.pdf', width = 10, height = 15)
```