Cluster GeographicallyWeighted Regression Using Gaussian Mixture Models and Dirichlet Processes for Spatial Heterogeneous Data.

This webpage is created as an online supplementary material for the manuscript **Cluster Geographically Weighted Regression Using Gaussian Mixture Models and Dirichlet Processes for Spatial Heterogeneous Data** . We present our modeling code using the nimble package (Valpine et al. 2017), as well as code to perform posterior inference and clustering.

1. The Bayesian geographically weighted regression code

Generate Simulated Data

As an illustration, we utilize the spatial arrangement of Louisiana counties similar to (Ma et al., 2021), which comprises a total of 64 counties. In our simulated data, we allocate three observations to each county. Consequently, we establish the necessary constants and proceed to generate a simulated data set.

```
N<- 192
S<- 64

set.seed(1)
beta<- c(2,0,0,4,8)
x1<- rnorm(192)
x2<- rnorm(192)
x3<- rnorm(192)
x4<- rnorm(192)
x5<- rnorm(192)
y  <- cbind(x1, x2, x3, x4, x5) %*% beta + rnorm(192)
#Replicate y 192 times to get a square matrix
y <- matrix(y, nrow = length(y), ncol = length(y), byrow = FALSE)
```

This code represents a BGWR model using nimble, incorporating vectorization techniques for improved performance and handling of large spatial data:

```
# Load required library
library(MASS)
library(nimble)
library(coda)
library(ClusterR)
library(mclust)
library(ggplot2)
library(tidyverse)
library(sf)
library(dplyr)
library(geosphere)
library(ggpubr)

dnorm_vec2 <- nimbleFunction(
```

```
  run = function(x = double(1), mean = double(1), sd = double(1),
                 log = integer(0, default = 0)) {
    returnType(double(0))
    logProb <- sum(dnorm(x, mean, sd, log = TRUE))
    if(log) return(logProb)
    else return(exp(logProb))
  })
registerDistributions('dnorm_vec2')

GWRCode <- nimbleCode({
  for (i in 1:S){
    y[1:N,i] ~ dnorm_vec2(b[i, 1] * x1[1:N] + b[i, 2] * x2[1:N] + b[i, 3] *
                            x3[1:N] + b[i, 4] * x4[1:N]+ b[i, 5] * x5[1:N],
                          1/(psi_y[i] * exp(-Dist[1:N, i]/lambda)))

    for(j in 1:5){
      b[i, j] ~ dnorm(0, tau=sigmainv)
    }
    psi_y[i] ~ dgamma(1, 1)
  }
  lambda ~ dunif(0, D)
  sigmainv ~ dgamma(1, 1)
})
```

To obtain posterior estimates of the proposed Geographically Weighted Regression (GWR) model, it is necessary to employ Markov Chain Monte Carlo (MCMC) algorithms to sample from the corresponding posterior distributions of the model's parameters. The advancement of software and computational techniques has been implemented to run the complex models. Ma and Chen (2019) provide a concise overview of several existing programs and software in this regard. In our study, we leverage the robust R package nimble (Valpine et al., 2017) to demonstrate the implementation of the Bayesian GWR model using nimble in R 3.4.1. A nimble model comprises four components, namely the model code, constants, data, and initial values for MCMC. The syntax of the model code bears resemblance to the BUGS language. For explanatory purposes, we define 'S' as the number of locations, 'N' as the number of observations, and 'p' as the dimension of the covariates vector (P= 5). When defining the model, we employ the nimble package's 'nimbleCode()' function.

Lines 3-10 defines a custom nimble function called **dnorm_vec2**. This function handles the calculation of the probability density function (PDF) or logarithm of the PDF for a normal distribution. It takes inputs x, **mean**, and **sd** for the values, mean, and standard deviation of the normal distribution, respectively. The log parameter, which is an optional argument with a default value of 0, determines whether the logarithm of the PDF is returned.

The code defines the GWRCode using the **nimbleCode()** function.. It involves a loop over i from 1 to S, where S represents the number of locations. In the GWRCode, the loop over i represents the spatial component of the GWR model. The response variable y at each location i is modeled using a linear combination of the covariates **x1** to **x5**, multipled by the coefficients $b[i, j]$. The exponential weighting scheme is applied using the distance matrix Dist and the bandwidth parameter lambda. The loop also defines the hierarchical prior for the coefficients $b[i, j]$ using a normal distribution with a mean of 0 and a precision (inverse variance) of sigmainv. The prior for **psi_y[i]** follows a gamma distribution with shape and rate parameters both equal to 1. Finally, the priors for the bandwidth parameter lambda and the precision sigmainv are specified. lambda is assigned a uniform prior distribution between 0 and D, and sigmainv follows a gamma distribution with shape and rate parameters both equal to 1.

Data List for Model

The next step involves defining the data list for the aforementioned model code. This data list includes the response variable, the covariates, and the distance matrix. To calculate the distance matrix using the great circle distance, you can use the following code. It is important to note that the entries in this matrix have been normalized to ensure a maximum value of 10.

```
netdist <- read.csv('country_dist.csv')

# Create longitude/latitude matrix
my_points <- matrix(c(netdist$Longitude,
                      netdist$Latitude),ncol=2)

colnames(my_points) <- c("longitude", "latitude")

gcd_matrix <- matrix(0, nrow = nrow(my_points), ncol = nrow(my_points))

# Calculate the cosine distance between each pair of points
for (i in 1:nrow(my_points)) {
  for (j in i:nrow(my_points)) {
    gcd_matrix[j,i] <- distCosine(my_points[i,], my_points[j,])
    gcd_matrix[i,j] <- distCosine(my_points[i,], my_points[j,])
  }
}


# Normalize matrix to have max value of 10
gcd_matrix_norm <- gcd_matrix / max(gcd_matrix) * 10
# Repeat the distance matrix three times as there are three observations per county
dist2 <- rbind(gcd_matrix_norm, gcd_matrix_norm, gcd_matrix_norm)

GWRData <- list(y = y, x1 = x1, x2 = x2, x3 = x3, x4 = x4, x5 = x5,
                Dist = dist2)
```

The provided code reads a CSV file named "country_dist.csv" that contains longitude and latitude coordinates for 64 regions in Louisiana. The coordinates are stored in the *netdist* dataframe. Further, a matrix named *my_points* is created using the longitude and latitude values from netdist. This matrix has two columns representing the longitude and latitude respectively. The matrix named *gcd_matrix* is initialized with zeros, and it will store the cosine distances between each pair of points. The *distCosine()* function is used to calculate the cosine distance between two points represented by their longitude and latitude coordinates. The nested for loop populates *gcd_matrix* with the calculated distances, taking advantage of symmetry by only calculating half of the distances and mirroring them.

The *gcd_matrix* is then normalized to have a maximum value of 10, resulting in gcd_matrix_norm. This normalization ensures that the values in the matrix fall within the range of 0 to 10. Finally, the dist2 matrix is created by repeating the netdist dataframe three times. This is done to match the number of observations (three) for each county in the data set.

The GWRData list is constructed with multiple elements: *y, x1, x2, x3, x4, x5* and Dist. These elements represent the response variable (y), covariates ($x1, x2, x3, x4, x5$), and the distance matrix (Dist). The dist2 matrix is assigned to the Dist element of the GWRData list.

Next, we need to define a constant list that includes the fixed quantities used in the model code. The number of locations is represented by S, the number of observations is denoted by N, and D represents the upper limit of the uniform distribution used for the bandwidth parameter.

```
GWRConst <- list(S = S, N = N, D = 50)
```

Lastly, we have assigned initial values to the parameters

```
GWRInits <- list(psi_y = rep(1, GWRConst$S),sigmainv=1, lambda = 10)
```

Run the Model

In nimble, we can use a one-line function to directly invoke the MCMC engine. This function typically takes the model code, data, constants, and initial values as inputs and provides various options for executing and controlling multiple chains, iterations, thinning intervals, and more.

The following code demonstrates the execution of a single MCMC chain with 5000 iterations, where the first 2000 iterations are designated as burn-in. As a result, the output will consist of 3000 posterior samples for the parameters b, psi_y, and lambda.

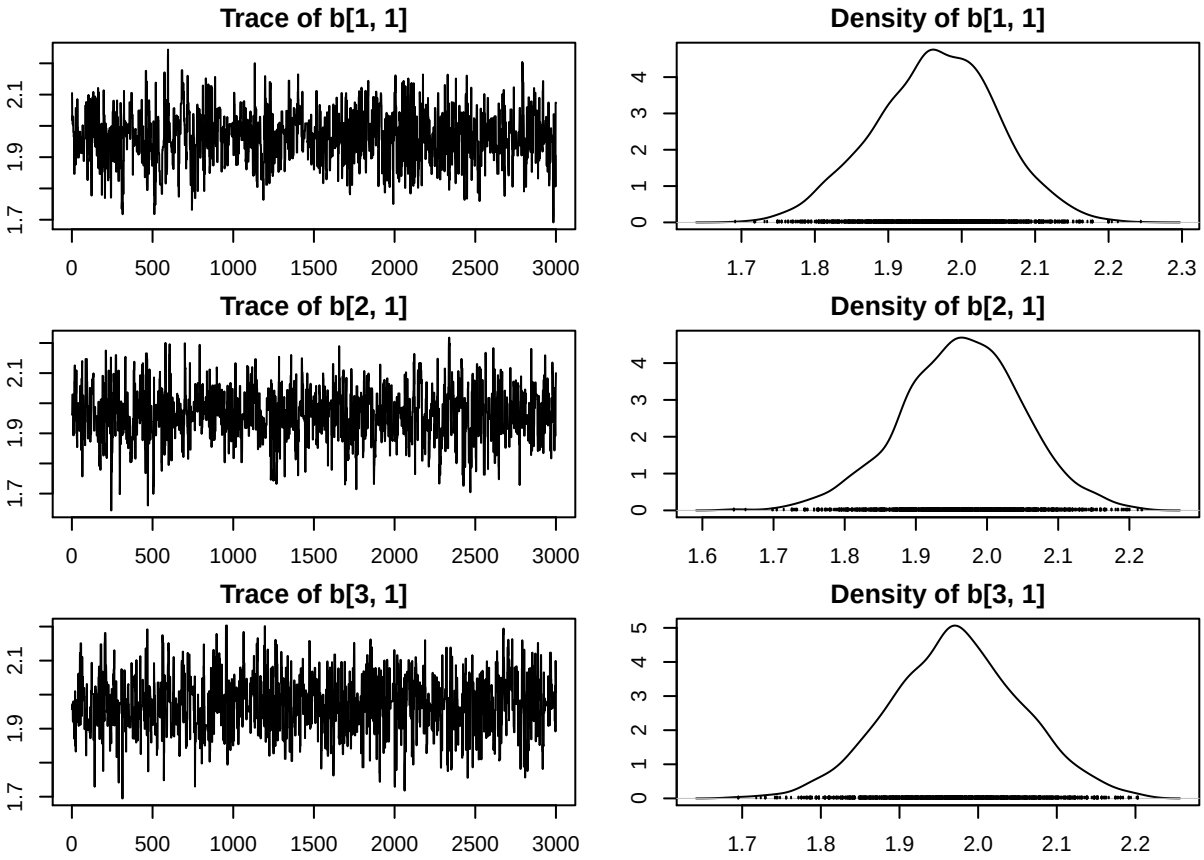
```
mcmc_output <- nimbleMCMC(code = GWRCode, data = GWRData, constants = GWRConst,
  inits = GWRInits, thin = 1, niter = 5000, nchains = 1, nburnin = 2000,
  monitors = c("b", "psi_y", "lambda"), setSeed = TRUE,
  samplesAsCodaMCMC = TRUE,summary=TRUE)
```

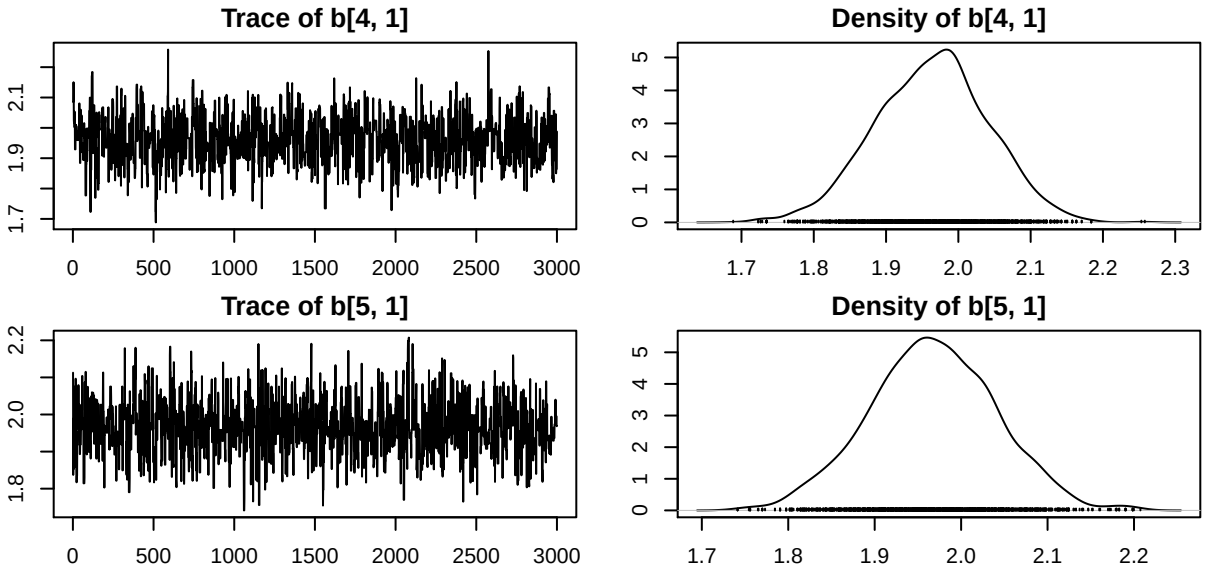
```
## |-------------|-------------|-------------|-------------|
## |-----------------------------------------------------|
```

Posterior Convergence Diagnostics and Estimation

The coda package (Plummer et al., 2006) offers convenient tools for performing posterior convergence diagnostics. It also provides useful functions for computing various percentiles of the posterior distribution, which are often of great interest in posterior inference.

```
pos_mcmc <- as.mcmc(mcmc_output)
par(mar = c(2, 2, 2, 2))
## plot the first five parameter estimates
plot(pos_mcmc$samples[,1:5])
```


Trace of b[1, 1]
Density of b[1, 1]
Trace of b[2, 1]
Density of b[2, 1]
Trace of b[3, 1]
Density of b[3, 1]



To calculate the posterior percentiles you can apply the *summary()* function to the *pos_mcmc* object.

```
pos_summ <- pos_mcmc$summary
str(pos_summ)
```

```
##  num [1:385, 1:5] 1.97 1.96 1.97 1.96 1.97 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:385] "b[1, 1]" "b[2, 1]" "b[3, 1]" "b[4, 1]" ...
##   ..$ : chr [1:5] "Mean" "Median" "St.Dev." "95%CI_low" ...
```

```
head(pos_mcmc$summary)
```

```
##             Mean   Median    St.Dev. 95%CI_low 95%CI_upp
## b[1, 1] 1.965267 1.966115 0.08310816  1.804599  2.126965
## b[2, 1] 1.964250 1.966808 0.08533229  1.791906  2.130293
## b[3, 1] 1.971916 1.971834 0.08372231  1.805765  2.138458
## b[4, 1] 1.963287 1.966216 0.07796904  1.809648  2.111861
## b[5, 1] 1.967033 1.965125 0.07316191  1.820293  2.111588
## b[6, 1] 1.959232 1.958422 0.07765915  1.809907  2.110899
```

2. Local Baysian Geogrpahically weighted regression and reversabile jump

In the previous section, we introduced a method called Bayesian Geographically Weighted Regression (GWR) with a hierarchical prior. By considering the hierarchical structure of the model, we can effectively capture the variations in covariate effects throughout the study area. Building upon this approach, our goal is to identify which covariates have a significant impact on specific locations while having minimal influence on others. To

achieve this, we use a combination of Bayesian statistics and a technique called Reversible Jump Markov Chain Monte Carlo (RJMCMC). By implementing the RJMCMC algorithm within the Bayesian GWR framework, we can explore and compare various models that represent different combinations of covariates in each location. This algorithm allows us to selectively include or exclude covariates based on their importance at specific locations, effectively capturing the spatial variations in covariate relationships with the response variable.

NIMBLE offers a convenient implementation of the (RJMCMC) algorithm for variable selection. The RJMCMC algorithm improves the mixing and efficiency of the sampling process. When a coefficient is not part of the model (or its indicator is set to 0), it will not be sampled. As a result, it will not be influenced by its prior distribution in those cases.

```
dnorm_vec2 <- nimbleFunction(
  run = function(x = double(1), mean = double(1), sd = double(1), log = integer(0, default = 0)) {
    returnType(double(0))
    logProb <- sum(dnorm(x, mean, sd, log = TRUE))
    if(log) return(logProb)
    else return(exp(logProb))
  })
registerDistributions('dnorm_vec2')

GWRCode <- nimbleCode({
  for (i in 1:S) {
    y[1:N, i] ~ dnorm_vec2(gamma[i, 1]* b[i,1] * x1[1:N] +gamma[i, 2]*b[i, 2] * x2[1:N]
 + gamma[i, 3]*b[i, 3] * x3[1:N] + gamma[i, 4]*b[i, 4] * x4[1:N]+
gamma[i, 5]*b[i, 5] * x5[1:N], 1/(psi_y[i] * exp(-Dist[1:N, i]/lambda)))

    for(j in 1:5){
      b[i, j] ~ dnorm(0, 1)
    }

    psi_y[i] ~ dgamma(1, 1)
    gamma[i, 1] ~ dbern(psi[1])
    gamma[i, 2] ~ dbern(psi[2])
    gamma[i, 3] ~ dbern(psi[3])
    gamma[i, 4] ~ dbern(psi[4])
    gamma[i, 5] ~ dbern(psi[5])
  }

  # Priors for the Bernoulli success probabilities
  for (j in 1:5) {
    psi[j] ~ dbeta(1, 1)
  }

  #psi ~ dbeta(1, 1)
  lambda ~ dunif(0, D)
})
```

We incorporate variable selection into the Bayesian Geographically Weighted Regression (GWR) model usin the variable selection which is achieved through the use of indicator variables (gamma) that determine whether a particular covariate is included in the model for each location.

```
GWRData <- list(y = y, x1 = x1, x2 = x2, x3 = x3, x4 = x4, x5 = x5,
                Dist = dist2)

GWRConst <- list(S = S, N = N, D = 50)
```

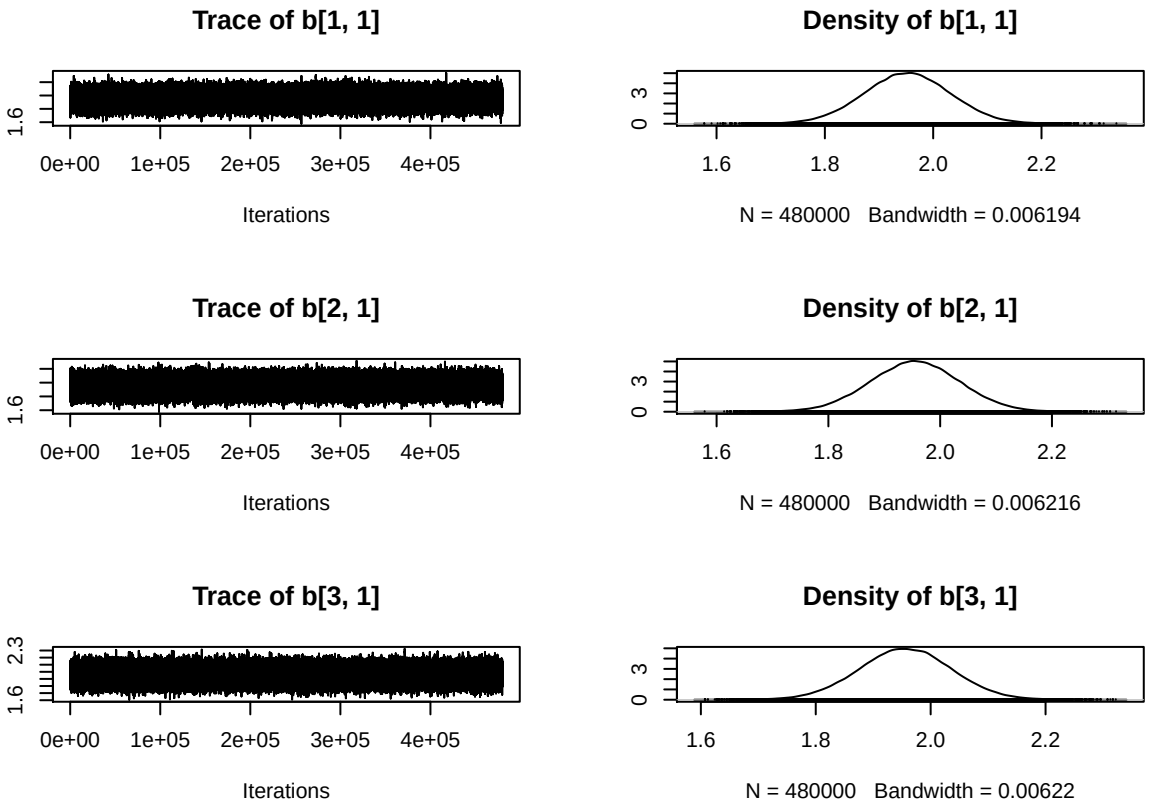
```

Inits <- list(psi_y = rep(1, GWRConst$S), lambda = 10,sigmainv=1,psi=rep(0.5,5),
              gamma = matrix(sample(0:1, S * 5, replace = TRUE), S, 5))


thinModel <- nimbleModel(
  code =GWRCode , data = GWRData, constants =GWRConst,
  inits =Inits)

cmodel <- compileNimble(thinModel)

RJexampleConf <- configureMCMC(thinModel)
```

```
## ===== Monitors =====
## thin = 1: b, lambda, psi, psi_y
## ===== Samplers =====
## RW sampler (385)
##   - b[]  (320 elements)
##   - psi_y[]  (64 elements)
##   - lambda
## conjugate sampler (5)
##   - psi[]  (5 elements)
## binary sampler (320)
##   - gamma[]  (320 elements)
```

```
configureRJ(conf = RJexampleConf,
            targetNodes = c("b[,1]","b[,2]","b[,3]","b[,4]","b[,5]"),
            indicatorNodes = c('gamma[,1]', 'gamma[,2]', 'gamma[,3]', 'gamma[,4]', 'gamma[,5]'),
            control = list(mean = 0, scale = 2))




Rmcmc <- buildMCMC(RJexampleConf)


Cmcmc <- compileNimble(Rmcmc)

samples <- runMCMC(Cmcmc, niter=500000,nburnin = 20000, nchains=1)
```

```
## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
```

```
mcmc_final<- as.mcmc(samples)
plot(mcmc_final[,c(1:3)])
```



we define the data required for the GWR model and the constants for the model simillar to befor then, we initialize the model parameters with appropriate starting values using the Inits list. The RJMCMC algorithm is configured using the configureRJ function, where we define the target nodes (parameters to be sampled) and indicator nodes (variables for variable selection) and set control options. The MCMC object (Rmcmc) is built based on the configuration. The MCMC sampling is performed using the runMCMC function, specifying the number of iterations, burn-in period, and number of chains. Finally, we convert the samples into an mcmc object for further analysis or visualization.

3. Cluster Analysis of Posterior Samples from BGWR Model

In this section, we conduct cluster analysis on a sample derived from the iterations of the Bayesian Geographically Weighted Regression (BGWR) method. We utilize two main probabilistic clustering algorithms: the Multivariate Gaussian Mixture Model and the Dirichlet Process. The dataset used in this analysis is based on the Georgia data set with known true clusters, which were obtained from the study by Ma et al. (2020). Additionally, the creation of the data set using the GWR model follows the approach outlined in Sagasawa et al. (2022). To evaluate the accuracy of the clustering models, we employ the Rand Index. The code for clustering the samples and computing the Rand Index is provided in the following code chunk.

We begin by utilizing the spatial configuration of Georgia state to partition the map and generate an observed dataset Y and a covariate matrix X. Additionally, a matrix containing great circle distances is obtained from the external file 'GAcentroidgcs.rds'. To conduct our analysis, first, we conduct the analysis using BGWR. Then, in step two, we draw samples from the posterior of bGWR to perform clustering on the coefficients

```
## The spatial simulated data with true labeled
## In this section we created the data from GWR model without the intercept term
distMat <- readRDS("./GAcentroidgcd.rds")
centroids <- as.data.frame(readRDS("GAcentroids.rds"))
```

```r
N <- n <- S <- 159
```

Where the true cluster distribution is give as:

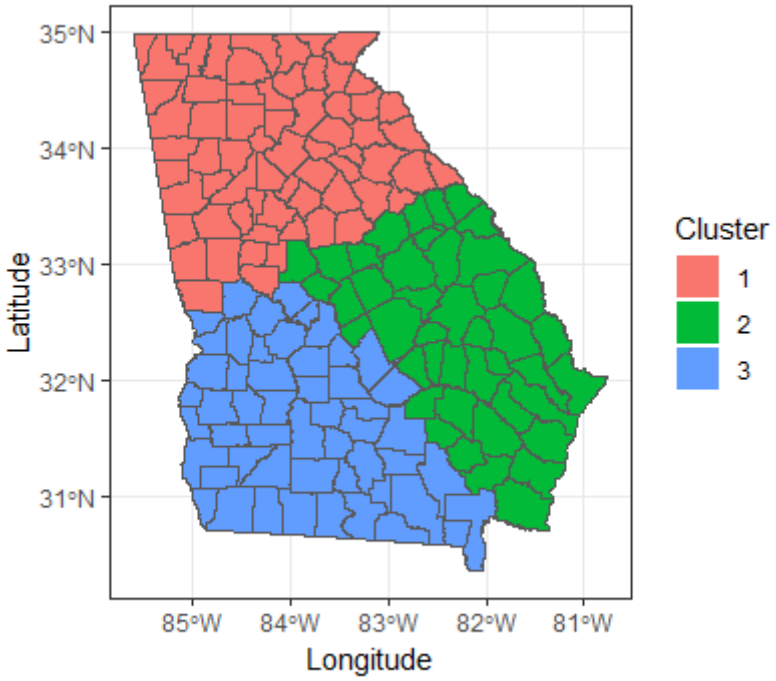


Figure 1: cluster assignment for Georgia counties used for simulation studies.

```r
# Generate true clustering settings based on centroids
# Set a random seed for reproducibility
set.seed(123)

asm <- c()
for (i in 1:nrow(centroids)) {
  if (centroids$x[i] - 2 * centroids$y[i] < -150) {
    asm[i] <- 1
  } else if (centroids$x[i] + centroids$y[i] > -51) {
    asm[i] <- 2
  } else {
    asm[i] <- 3
  }
}

dd <- distMat

# Create a matrix to store beta values for each cluster
betaMat <- t(matrix(nrow = 159, ncol = 6, byrow = TRUE))
for (i in 1:159) {
  ## cluster 1
```

```
  betaMat[,asm == 1] <- c(9, 0, -4, 0, 2, 5)
  ## cluster 2
  betaMat[,asm == 2] <- c(1, 7, 3, 6, 0, -1)
  ## cluster 3
  betaMat[,asm == 3] <- c(2, 0, 6, 1, 7, 0)
}

# Calculate six different weighted functions using Gaussian kernel
# (V1 to V6) based on dd and certain constants
V1 <- exp(-dd / (1 * 5))
V2 <- exp(-dd / (2 * 0.1))
V3 <- exp(-dd / (3 * 0.2))
V4 <- exp(-dd / 2 * 4)
V5 <- exp(-dd / 30 * 5)
V6 <- exp(-dd / 40 * 6)

# Calculate c1 to c6 based on V1 to V6 and betaMat
c1 <- V1[1, ] * (betaMat[1, ])
c2 <- V2[1, ] * betaMat[2, ]
c3 <- V3[1, ] * betaMat[3, ]
c4 <- V4[1, ] * betaMat[4, ]
c5 <- V5[1, ] * betaMat[5, ]
c6 <- V6[1, ] * betaMat[6, ]


# Combine c1 to c6 into a matrix Beta.true
Beta.true <- cbind(c1, c2, c3, c4, c5, c6)

# Set the number of samples
n <- 159


# Generation of sampling locations Sp
Sig.true<- 1

# Covariates with a given range parameter phi
phi <- 0.9
dd <- distMat
mat <- exp(-dd / phi)
x1 <- mvrnorm(1, rep(0, n), mat)
x2 <- mvrnorm(1, rep(0, n), mat)
x3 <- mvrnorm(1, rep(0, n), mat)
x4 <- mvrnorm(1, rep(0, n), mat)
x5 <- mvrnorm(1, rep(0, n), mat)
x6 <- mvrnorm(1, rep(0, n), mat)
X <- data.frame(x1, x2, x3, x4, x5, x6)
Mu <- apply(cbind(X) * Beta.true, 1, sum)

#Creat the data
Y <- rnorm(n, Mu, Sig.true)


# Define a custom density function dnorm_vec2 for nimble
```

```r
dnorm_vec2 <- nimbleFunction(
  run = function(x = double(1), mean = double(1), sd = double(1),
log = integer(0, default = 0)) { returnType(double(0))
    logProb <- sum(dnorm(x, mean, sd, log = TRUE))
    if (log) return(logProb)
    else return(exp(logProb))
  })
registerDistributions('dnorm_vec2')

# Define the GWRCode for the nimble model
GWRCode <- nimbleCode({
  for (i in 1:S) {
    y[1:N, i] ~ dnorm_vec2(b[i, 1] * x1[1:N] + b[i, 2] * x2[1:N] + b[i, 3] *
     x3[1:N] + b[i, 4] * x4[1:N] + b[i, 5] * x5[1:N] + b[i, 6] * x6[1:N],
                            1 / (psi_y[i] * exp(-Dist[1:N, i] / lambda)))

    #psi_y[i] ~ dgamma(100,100)
    for (j in 1:6) {
      b[i, j] ~ dnorm(0, tau = sigmainv)
    }

  }
  lambda ~ dunif(0, D)
  sigmainv ~ dgamma(1, 1)

})

Y <-  matrix(Y, nrow = length(Y), ncol = length(Y), byrow = FALSE)
# Prepare the data, constants, and initial values for the nimble model
GWRdata <- list(y = Y, x1 = X[, 1], x2 = X[, 2], x3 = X[, 3], x4 = X[, 4],
                x5 = X[, 5], x6 = X[, 6], Dist = distMat)
GWRConsts <- list(S = 159, M = 50, N = 159, D = 50)
GWRInits <- list( psi_y = rep(1, GWRConsts$S), lambda = 10,
                  sigmainv = 1)

# Perform MCMC sampling using nimble
mcmc.out_Spatial <- nimbleMCMC(code = GWRCode, data = GWRdata, constants = GWRConsts,
  inits = GWRInits, monitors = c("b","lambda"), niter = 5000, nburnin = 2000,
  nchains = 1, setSeed = TRUE)
```

```
## |-------------|-------------|-------------|-------------|
## |-------------------------------------------------------|
```

```r
# Convert nimble MCMC output to coda format
mcmc.out <- as.mcmc(mcmc.out_Spatial)

# Get the total number of iterations in the chain
total_iterations <- nrow(mcmc.out)

# Set the number of iterations you want to sample (e.g., 500)
sample_iterations <- 500
sampled_indices <- sample(1:total_iterations, sample_iterations, replace = FALSE)

# Extract the sampled rows from the mcmc.out object
```

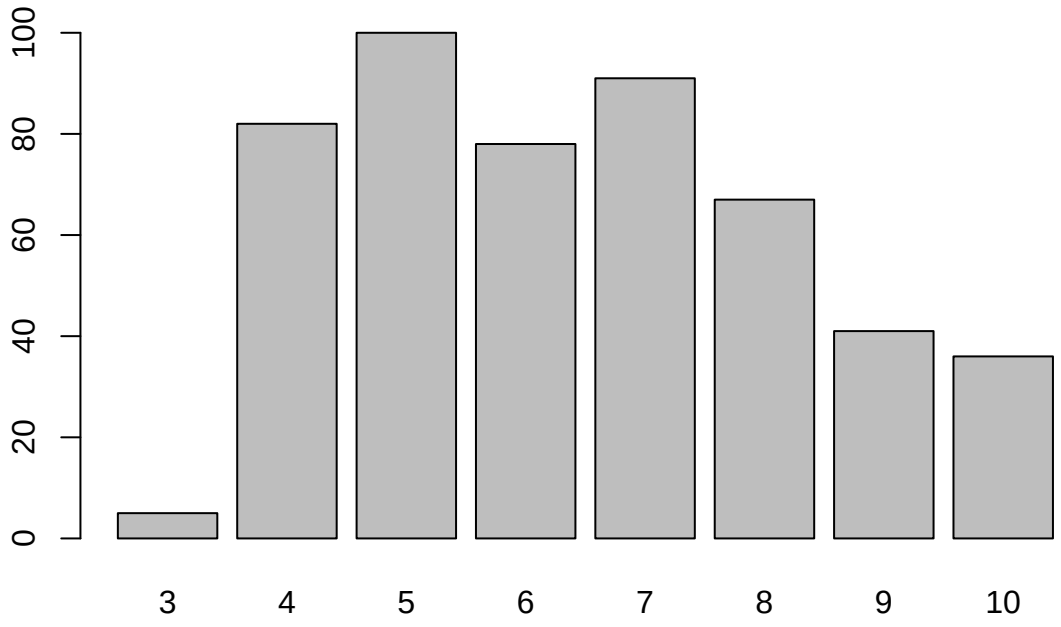
```r
sampled_mcmc_results <- mcmc.out[sampled_indices, ]
```

n the second step, we utilized a random sample of 500 iterations from the posterior as inputs for our clustering algorithm.

We initiated the process by sampling approximately 500 iterations from the beta coefficients obtained from BGWR. We then determined the optimal number of clusters for the Gaussian Mixture Model (GMM) using the Bayesian Information Criterion (BIC).

To keep track of the number of components in each isample, we created an empty variable called "n_cluster" and populated it with these values. Additionally. By plotting these values, we gained insightful visualizations, facilitating a deeper comprehension of the identified clusters.

```r
hist_data<- table(n_cluster)
barplot(hist_data)
```



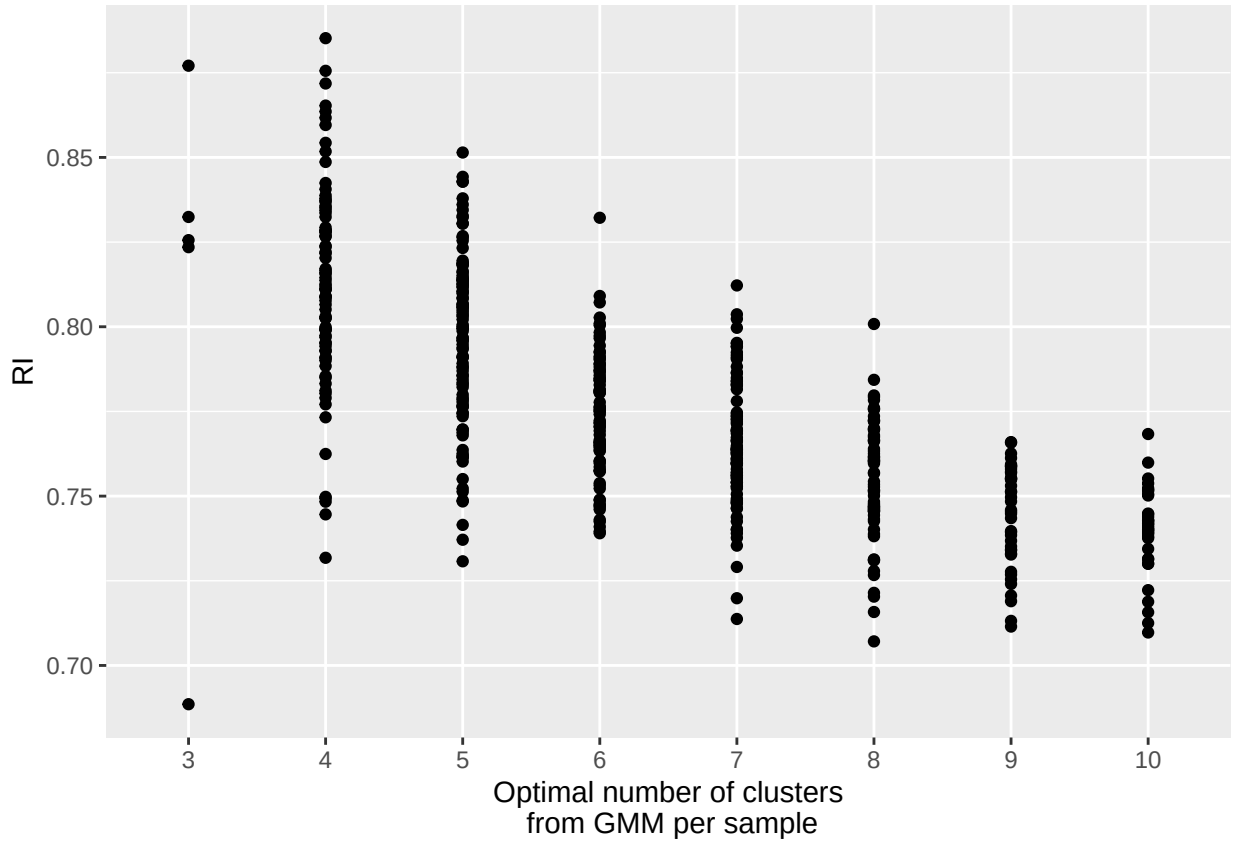
The Rand Index (RI) for each sample iteration can be computed as follows:

```r
for (i in 1:num_rows){
  RI[i] <- fossil::rand.index(gmm[[i]]$classification,as.numeric(asm))
}
```

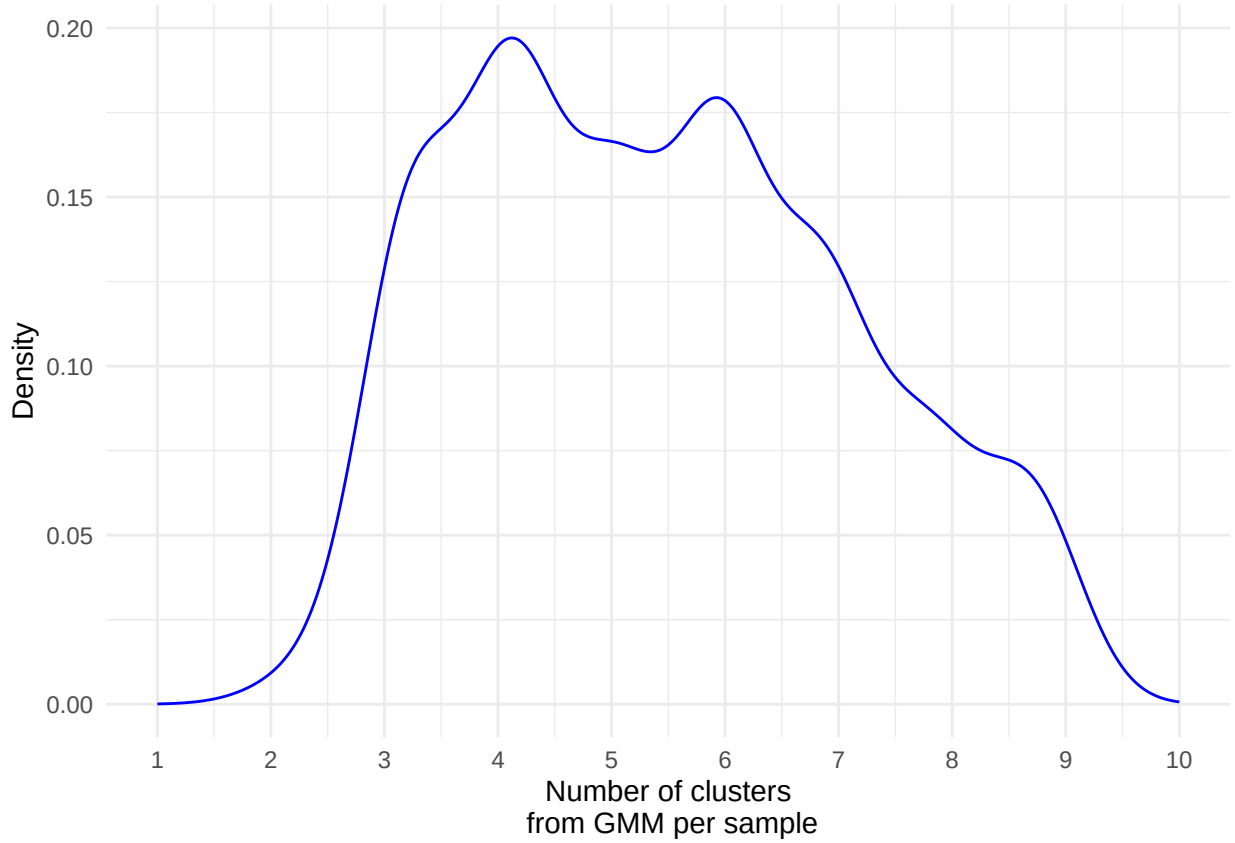
In order to visualize the empirical density and the scatter plot depicting the relationship between the optimal number of clusters from each sample and the Rand Index (RI), we can use the following approach:

```r
# Convert n_cluster and RI to data frames
data_df <- data.frame(n_cluster, RI)

# Scatter plot of RI against n_cluster using ggplot2
```

```r
ggplot(data_df, aes(x = as.factor(n_cluster), y = RI)) +
  geom_point() +
  labs( x = "Optimal number of clusters\n from GMM per sample", y = "RI")
```




```r
# Create the density object using the density function
density_g <- density(n_cluster)

# Create a sequence from 1 to 10
x_sequence <- seq(1, 10, length.out = length(density_g$x))

# Create a data frame with the density values and the sequence on the x-axis
df_density <- data.frame(x = x_sequence, density = density_g$y)

# Plot the density using ggplot2
ggplot(df_density, aes(x = x, y = density)) +
  geom_line(color = "blue") +
  xlab("Number of clusters\n from GMM per sample") +
  ylab("Density") +
  ggtitle(NULL) +
  scale_x_continuous(breaks = 1:10, labels = 1:10) +
  theme_minimal()
```



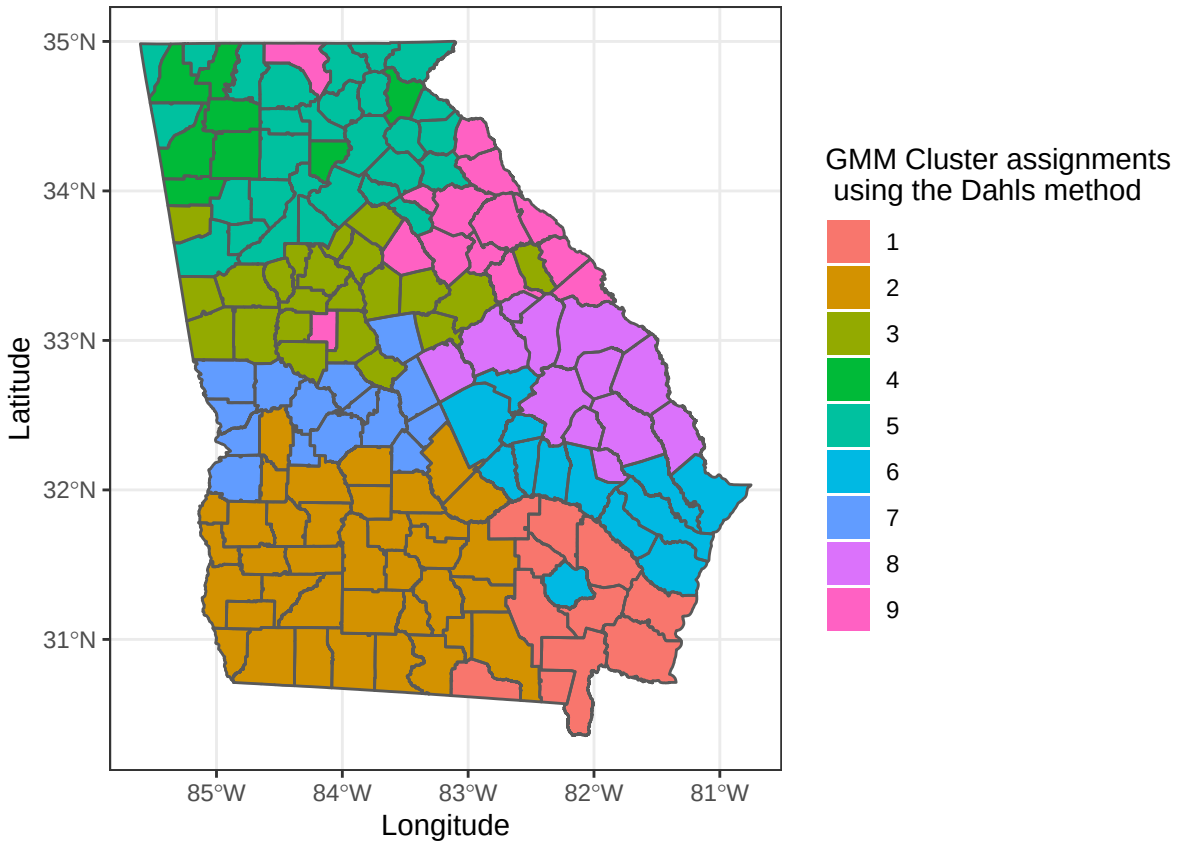
We test the accuracy of our clusters, and we determine the cluster configuration per region using two methods mentioned in the main paper. These methods include Dahl's method and the mode approach.In each of these setup we calculated the rand index and also we show the clusters distrubution on map

```r
###Dahl's method

# Get the number of rows samples (iterations) in the gmm list
num_rows <- length(gmm)

# Get the number of data points in each sample's classification
num_data_points <- length(gmm[[1]]$classification)

# Create a matrix to store the cluster assignments for all samples
classification_matrix <- matrix(NA, nrow = num_rows, ncol = num_data_points)

# Populate the classification matrix with the cluster assignments from each sample
for (i in 1:num_rows) {
  classification_matrix[i, ] <- gmm[[i]]$classification
}

# The 'classification_matrix' now contains the classifications for all samples

# Assign the 'classification_matrix' to 'latentZMat'
latentZMat <- classification_matrix

# Calculate the empirical probability matrix 'bBar'
membershipList <- purrr::map(1:nrow(latentZMat), .f = function(x) {
```

```r
  outer(latentZMat[x,], latentZMat[x, ], "==")
})
bBar <- Reduce("+", membershipList) / length(membershipList)

# Calculate the sum of squared differences 'lsDist'
lsDist <- purrr::map_dbl(membershipList, ~sum((.x - bBar) ^ 2))

# Find the optimal iteration using the smallest sum of squared differences
mcluster <- which.min(lsDist)

# Extract the final inferred cluster assignment from the optimal iteration
finalCluster <- as.numeric(latentZMat[mcluster[1],])

# Calculate the Rand Index (RI) to compare the final Cluster with the true clustering 'asm'
fossil::rand.index(as.numeric(finalCluster), asm)
```

```
## [1] 0.765783
```

```r
Georgia <- read_sf("Georgia_dat.shp") %>% filter(!st_is_empty(.))
mydata_and_myMap<- mutate(Georgia,finalCluster)

 ggplot() +
  xlab("Longitude") +
  ylab("Latitude") +
  theme_bw() +
  theme(legend.position = "right") +
  labs(fill='GMM Cluster assignments \n using the Dahls method') +
 geom_sf(data = mydata_and_myMap, aes(fill=factor(mydata_and_myMap$finalCluster)), color=NA) +
 geom_sf(data=mydata_and_myMap, fill=NA)
```

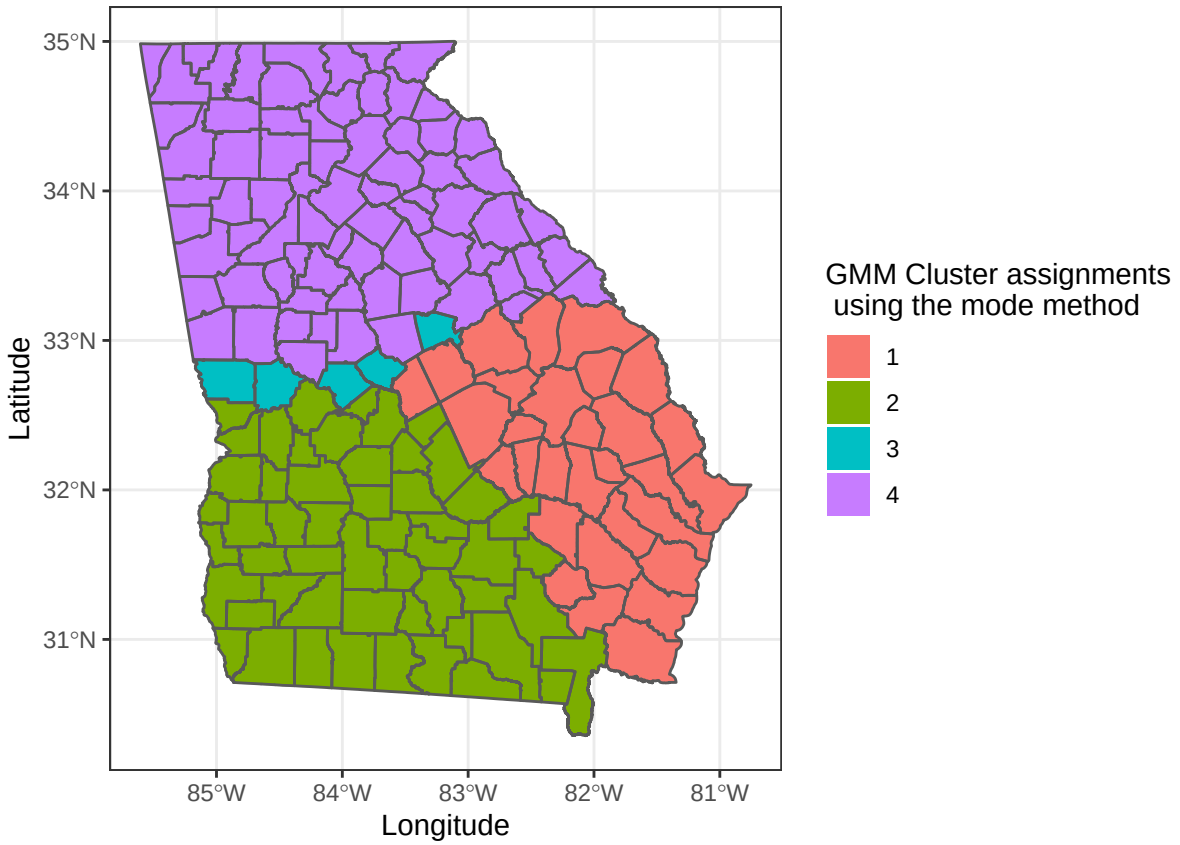


And from the mode

```
# Point estimate for the latent variables
latentZMat <- classification_matrix
latentPE <- as.numeric(unlist(apply(latentZMat, 2, FUN = function(x) {
  return(DescTools::Mode(x)[1])
})))
# Check number of clusters, and number of regions in each cluster

fossil::rand.index(asm, latentPE)
```

```
## [1] 0.8992915
```

```
Georgia <- read_sf("Georgia_dat.shp") %>% filter(!st_is_empty(.))
mydata_and_myMap<- mutate(Georgia,latentPE)

ggplot() +
  xlab("Longitude") +
  ylab("Latitude") +
  theme_bw() +
  theme(legend.position = "right") +
  labs(fill='GMM Cluster assignments \n using the mode method ') +
  geom_sf(data = mydata_and_myMap, aes(fill=factor(mydata_and_myMap$latentPE)), color=NA) +
 geom_sf(data=mydata_and_myMap, fill=NA)
```



In the next chunks we adopted the DPMM to do the clustering fro the samples from the BGWR model,the attached chuck show the results from one sample

```
sampled_mcmc_results <- mcmc.out[sampled_indices, ]

# Take the first sample
x1<- as.numeric(sampled_mcmc_results[1,1:159])
x2<-as.numeric(sampled_mcmc_results[1,160:318])
x3<- as.numeric(sampled_mcmc_results[1,319:477])
x4<-as.numeric(sampled_mcmc_results[1,478:636])
x5<- as.numeric(sampled_mcmc_results[1,637:795])
x6<- as.numeric(sampled_mcmc_results[1,796:954])

A<- data.frame(x1,x2,x3,x4,x5,x6)
library(rjags)
dp_normal_blocked <- "
model {
  for (i in 1:n) {
    y[i,1:p] ~ dmnorm(mu[,zeta[i]], Tau[,,zeta[i]])
    zeta[i] ~ dcat(pi[])
  }
  for (h in 1:H) {
    mu[1:p,h] ~ dmnorm(mu0, Tau[,,h])
    Tau[1:p,1:p,h] ~ dwish(D[,], c)
    Sigma[1:p,1:p,h] <- inverse(Tau[,,h])
  }
  # Stick breaking
```

```
  for (h in 1:(H-1)) { V[h] ~ dbeta(1, a) }
  V[H] <- 1
  pi[1] <- V[1]
  for (h in 2:H) {
    pi[h] <- V[h] * (1 - V[h-1]) * pi[h-1] / V[h-1]
  }
}
"
dat <- list(
  n = nrow(A),
  y = as.matrix(scale(A)),
  p = ncol(A),
   H = 10,
  a = 2,
  D = diag(1, ncol(A)),
  c = ncol(A) + 1,
  mu0 = c(0,0,0,0,0,0)
)

inits <- list(
  mu = matrix(rnorm(dat$p * dat$H, mean = 0, sd = 100), dat$p, dat$H),
  zeta = sample(1:dat$H, dat$n, replace = TRUE),
  Tau = array(diag(dat$p), dim = c(dat$p, dat$p, dat$H))
)

params <- c( "zeta",  "pi")

model <- jags.model(textConnection(dp_normal_blocked), data = dat, inits = inits,
                    n.chains = 1)
```

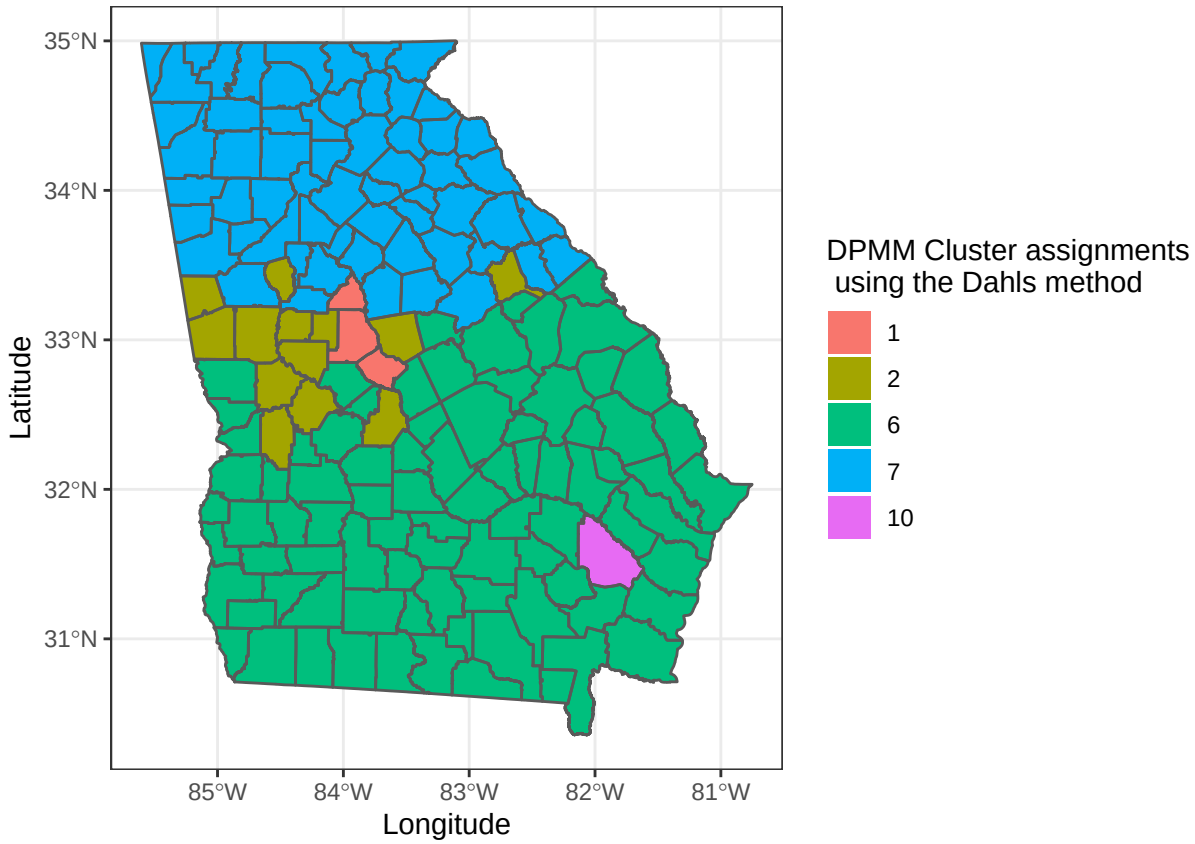
```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 159
##    Unobserved stochastic nodes: 188
##    Total graph size: 753
##
## Initializing model
```

```
samples <- coda.samples(model, variable.names = params, n.iter = 1000)


pos_mc<- as.mcmc(samples )

 latentZMat <-pos_mc[, grepl("zeta", colnames(pos_mc))]


# Clusters configurations using Dahl's method

membershipList <- purrr::map(1:nrow(latentZMat), .f = function(x) {
  outer(latentZMat[x,], latentZMat[x, ], "==")
})

```

```r
# The empirical probability matrix
bBar <- Reduce("+", membershipList) / length(membershipList)

# Sum of squared differences
lsDist <- purrr::map_dbl(membershipList, ~sum((.x - bBar) ^ 2))

# Find the optimal iteration, and take as the final inferences result
# If there are multiple optimal iterations, take the first one
mcluster <- which.min(lsDist)
finalCluster <- as.numeric(latentZMat[mcluster[1],])

fossil::rand.index(as.numeric(finalCluster), asm)
```

```
## [1] 0.7645888
```

```r
Georgia <- read_sf("Georgia_dat.shp") %>% filter(!st_is_empty(.))
mydata_and_myMap<- mutate(Georgia,finalCluster)
 ggplot() +
  xlab("Longitude") +
  ylab("Latitude") +
  theme_bw() +
  theme(legend.position = "right") +
  labs(fill='DPMM Cluster assignments \n using the Dahls method ') +
  geom_sf(data = mydata_and_myMap, aes(fill=factor(mydata_and_myMap$finalCluster)), color=NA) +
 geom_sf(data=mydata_and_myMap, fill=NA)
```

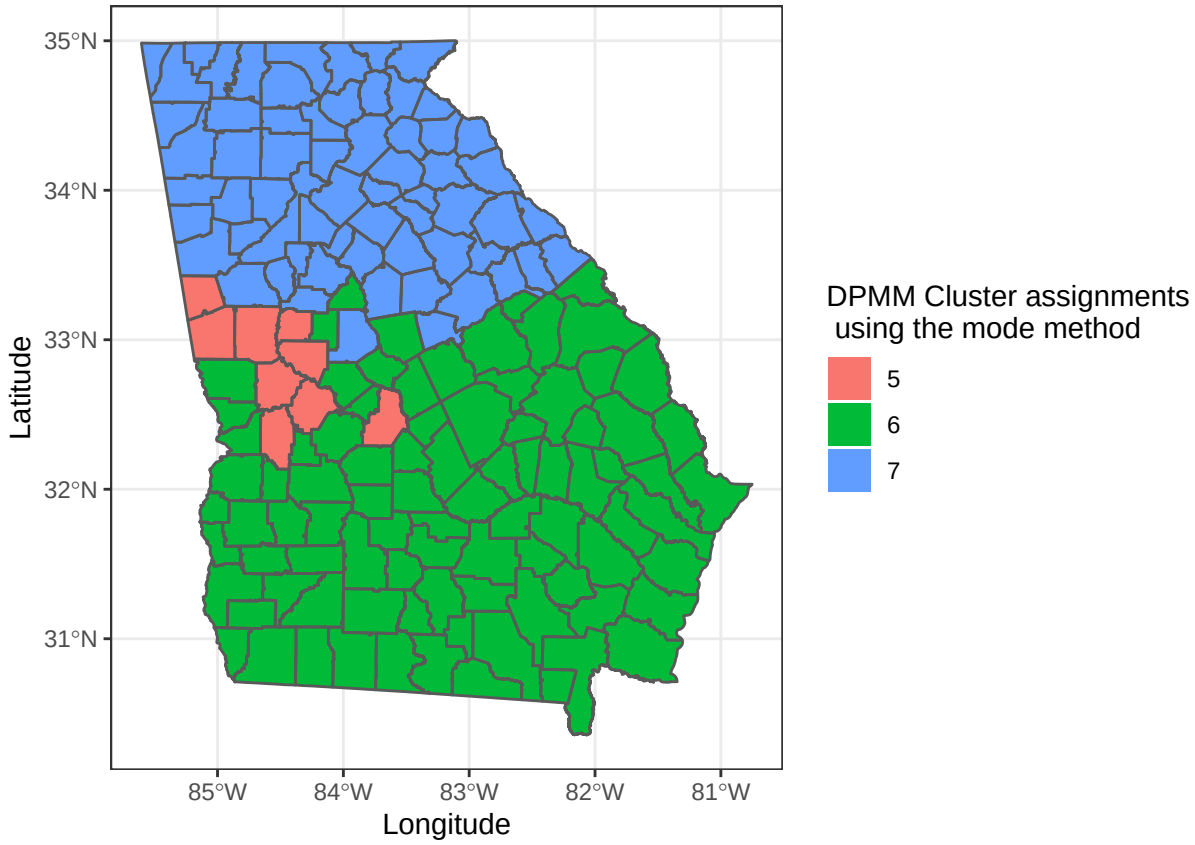


And from the mode,

```r
  latentZMat <-pos_mc[, grepl("zeta", colnames(pos_mc))]

latentPE <- as.numeric(unlist(apply(latentZMat, 2, FUN = function(x) {
  return(DescTools::Mode(x)[1])
})))
# Check number of clusters, and number of regions in each cluster

fossil::rand.index(asm, latentPE)
```

```
## [1] 0.7425364
```

```r
Georgia <- read_sf("Georgia_dat.shp") %>% filter(!st_is_empty(.))
mydata_and_myMap<- mutate(Georgia,latentPE)

ggplot() +
  xlab("Longitude") +
  ylab("Latitude") +
  theme_bw() +
  theme(legend.position = "right") +
  labs(fill='DPMM Cluster assignments \n using the mode method ') +
  geom_sf(data = mydata_and_myMap, aes(fill=factor(mydata_and_myMap$latentPE)), color=NA) +
 geom_sf(data=mydata_and_myMap, fill=NA)
```



References

- Ma, Z., & Chen, G. (2019). Bayesian Semiparametric Latent Variable Model with DP Prior for Joint Analysis: Implementation with nimble. *Statistical Modelling*, *20*(4), 347-368. (https://journals.sagepub.com/doi/abs/10.1177/1471082X18810118)
- Ma, Z., Xue, Y., & Hu, G. (2021). Geographically Weighted Regression Analysis for Spatial Economics Data: A Bayesian Recourse. *International Regional Science Review*, *44*(5), 582-604.(https://journals.sagepub.com/doi/full/10.1177/0160017620959823)
- Plummer, M., Best, N., Cowles, K., & Vines, K. (2006). CODA: Convergence Diagnosis and Output Analysis for MCMC. *R News*, *6*(1), 7-11. (https://journal.r-project.org/archive/)
- Ma, Z., Xue, Y., & Hu, G. (2020). Heterogeneous Regression Models for Clusters of Spatial Dependent Data. *Spatial Economic Analysis*, *15*(4), 459-475.(https://www.tandfonline.com/doi/full/10.1080/17421772.2020.1784989)
- Sugasawa, S., & Murakami, D. (2022). Adaptively Robust Geographically Weighted Regression. *Spatial Statistics*, *48*, 100623. (https://www.sciencedirect.com/science/article/pii/S2211675322000185)
- Valpine, P. D., Turek, D., Paciorek, C. J., Anderson-Bergman, C., Temple Lang, D., & Bodik, R. (2017). Programming with Models: Writing Statistical Algorithms for General Model Structures with NIMBLE. *Journal of Computational and Graphical Statistics*, *26*(2), 403-413.