Spatial non-parametric Bayesian clustered regression coefficients

This webpage is created as an online supplementary material for the manuscript **Spatial non-parametric Bayesian clustered regression coefficients**. We present our modeling code using the Nimble package, as well as code to perform posterior inference and clustering.

Generate Simulated Data

To provide an example, we adopt a spatial layout akin to the arrangement of counties in Georgia, as demonstrated by (Ma et al., 2019), encompassing a total of 159 counties. In our simulated data set, we distribute three observations within each geographical region. This leads us to define the essential parameters, allowing us to subsequently simulate a complete data set. Where the true cluster distribution is give as:

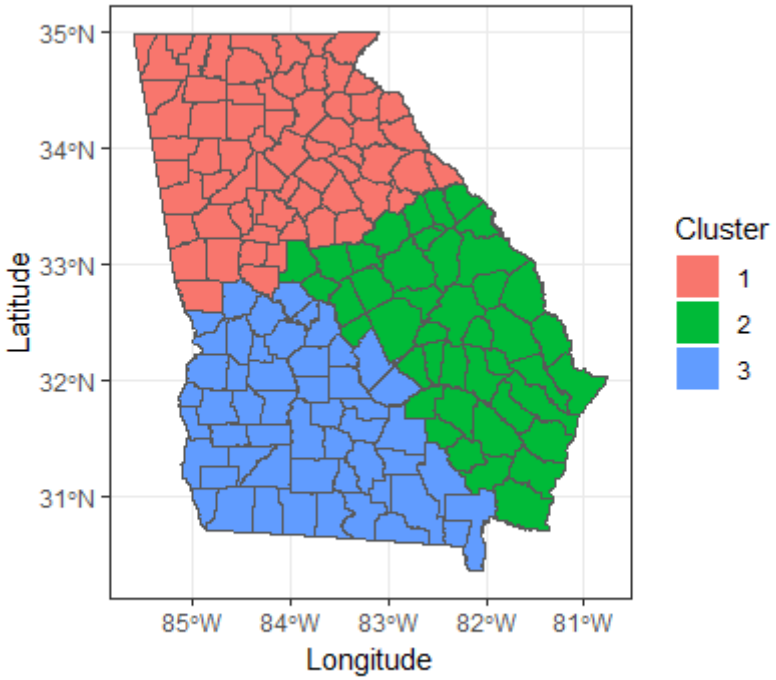


Figure 1: cluster assignment for Georgia counties used for simulation studies.

```
library(MASS)
library(nimble)
library(coda)
library(ClusterR)
library(mclust)
library(ggplot2)
library(tidyverse)
```

```r
library(sf)
library(dplyr)
library(geosphere)
library(ggpubr)

distMat <- readRDS("./GAcentroidgcd.rds")
centroids <- as.data.frame(readRDS("GAcentroids.rds"))
dd <- distMat

# 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, ]
```

```r


# 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)

#Converts the 'x' and 'y' values from the 'centroids' to radians
centroids <- centroids
cons = centroids$x * pi/180
const= centroids$y * pi/180

new_cen<- data.frame(cons,const)
```

In this chunk, the distance matrix and centroid coordinates obtained from external files. We sets a deterministic seed for reproducibility and assigns data points to one of three clusters based on the conditions defined by the centroid coordinates. The data is then processed to generate beta values specific to each cluster. Using the distance matrix, it creates six different kernel weighted functions. These functions, combined with the beta values, produce six coefficients for each data point. By generating multiple covariates based on a given range parameter, the code ultimately constructs a dataset Y from GWR model where each data point is drawn from a normal distribution with a mean determined by these coefficients and covariates. The code then transforms the centroid coordinates to radians, preparing them for potential the analysis.

```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 Code 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)


    b[i, 1:6] <- bm[latent[i], 1:6]
     bm[i, 1:6] ~ dmnorm(mu[ 1:6,latent[i]], Tau[1:6, 1:6,latent[i]])
    latent[i] ~ dcat(pi[i,1:M])}
  #
  for (m in 1:M) {
    mu[ 1:6,m] ~ dmnorm(mu0[1:6], Tau[1:6,1:6,m])
    Tau[1:6,1:6,m] ~ dwish(D1[1:6,1:6 ], c)
    Sigma[ 1:6,1:6,m] <- inverse(Tau[1:6,1:6,m])

    for (j in 1:6) {
      mu0[j] ~ dnorm(0, 1)
    }
  }


  for (i in 1:S) {
    pi[i,1] <- vs[i, 1]
    for (j in 2 : M){ pi[i, j ] <- vs[i, j ]*prod(vsout[i, 1 : ( j - 1)])
    }}


  for(j in 1 : (M - 1)){
    V[j] ~ dbeta(1, 1)

    for (i in 1:S) {
      #
      weight[i,j]<- exp(-((new_cen[i, 1] - knot[j, 1])^2)/r[j,1]^2) * exp(-((new_cen[i, 2]- knot[j, 2])

      vs[i,j]<- weight[i,j]*V[j]
      vsout[i,j]<- 1-vs[i,j]

    }
    knot[j, 1]~dunif(0,10)
    knot[j, 2]~dunif(0,10)
    r[j, 1]<- h^2/2
    r[j, 2]<- h^2/2
  }

  for (k in 1:S){vs[k,M]<- 1}

 lambda~dunif(0,D)
   h~dunif(0,5)
})
```

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 . It is important to note that the entries in this matrix have been normalized to ensure a maximum value of 10.

```
p<- 6


##Replicate Y to get a square matrix
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,D1=diag(rep(1, p)),c = p + 1,
                new_cen=new_cen)

GWRConsts <- list(S = 159, M = 10, N = 159, D = 50)

dim_tau <- c(6, 6, GWRConsts$M)

# Create the array with all elements set to 1
Tau <- array(1, dim = dim_tau)

# Set the diagonal elements to 1
for (i in 1:dim_tau[3]) {
  Tau[, , i] <- diag(dim_tau[1]) * 1
}

GWRInits <- list( psi_y = rep(100, GWRConsts$S), lambda = 10,
                  mu0 = rep(0,6),latent = rep(1, GWRConsts $S),
                  Tau=Tau,knot = matrix(runif((GWRConsts$M-1)*2,0, 1), GWRConsts$M-1, 2)
                  ,h=1,V = rbeta(GWRConsts$M - 1, 1, 1))
```

The proposed model

The dnorm_vec2, using the Nimble framework. The function calculates the sum of log probabilities for a given vector using a normal distribution. After defining the function, it's registered for use in subsequent Nimble models.

The core of the code establishes a Bayesian Geographically Weighted Regression (GWR) model, termed GWRCode. This model attempts to capture spatial heterogeneity by allowing parameters to vary over space. It uses several probability distributions, including normal (dnorm), gamma (dgamma), multinormal (dmnorm), categorical (dcat), and beta (dbeta). A key feature of the GWR model is that it allows for local relationships between predictors and response variables, using weights based on spatial proximity.

The model incorporates latent variables to capture unobserved heterogeneities and knots for spatial locations that influence local parameter estimates. It calculates the weight for each observation based on its distance to these knots. Additionally, certain parameters, like mu0, are given weakly informative prior distributions.

Lastly, the code processes data (distance matrix, covariates, and response) and prepares it alongside certain constants to be used in the GWR model. The preparation includes the creation of a 3-dimensional array Tau for the covariance structure, where the diagonal elements are set to 1.

In essence, the code captures spatially varying relationships in data using a Bayesian GWR model and sets the stage for model fitting and inference using the Nimble framework.

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

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

Run the Model

In Nimble, we can directly run the MCMC engine using the following code. This function is designed to run the Markov Chain Monte Carlo (MCMC) process and typically requires the model code, data, and initial values as input. It offers various options for controlling aspects such as multiple chains, the number of iterations, thinning intervals, and more.

The following code example illustrates the execution of a single MCMC chain with a total of 5000 iterations, where the initial 2000 iterations are used as burn-in. Consequently, the output will provide 3000 posterior samples for the parameters: "pi", and "latent."

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. In the next chunk the code perform clustering analysis on the posterior samples from the Bayesian model and visualize the resulting cluster assignments on a map of Georgia. The Dahl's and mode methods are used for clustering, and the final clusters are plotted on the map using ggplot.

```r
mcmc.out <- as.mcmc(mcmc.out_Spatial)
latentZMat <-mcmc.out

# Assuming latentZMat is a data frame or matrix
# and you want to extract columns with names containing "latent"
latentZMat<- latentZMat[, grepl("latent", colnames(latentZMat))]

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

## 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 inferenced result
## if there are multiple optimal iterations, take the first one
mcluster <- which.min(lsDist)
finalCluster <- as.numeric(latentZMat[mcluster[1],])
finalCluster <- ifelse(latentZMat[mcluster[1],] == 9, 1, ifelse(latentZMat[mcluster[1],] == 10, 2, ifels

table(finalCluster)
```

```
## finalCluster
```

```
##  1  2  3
## 65 42 52
```

```
# Get the total number of iterations in the chain
fossil::rand.index(as.numeric(finalCluster), asm)
```

```
## [1] 0.8785925
```

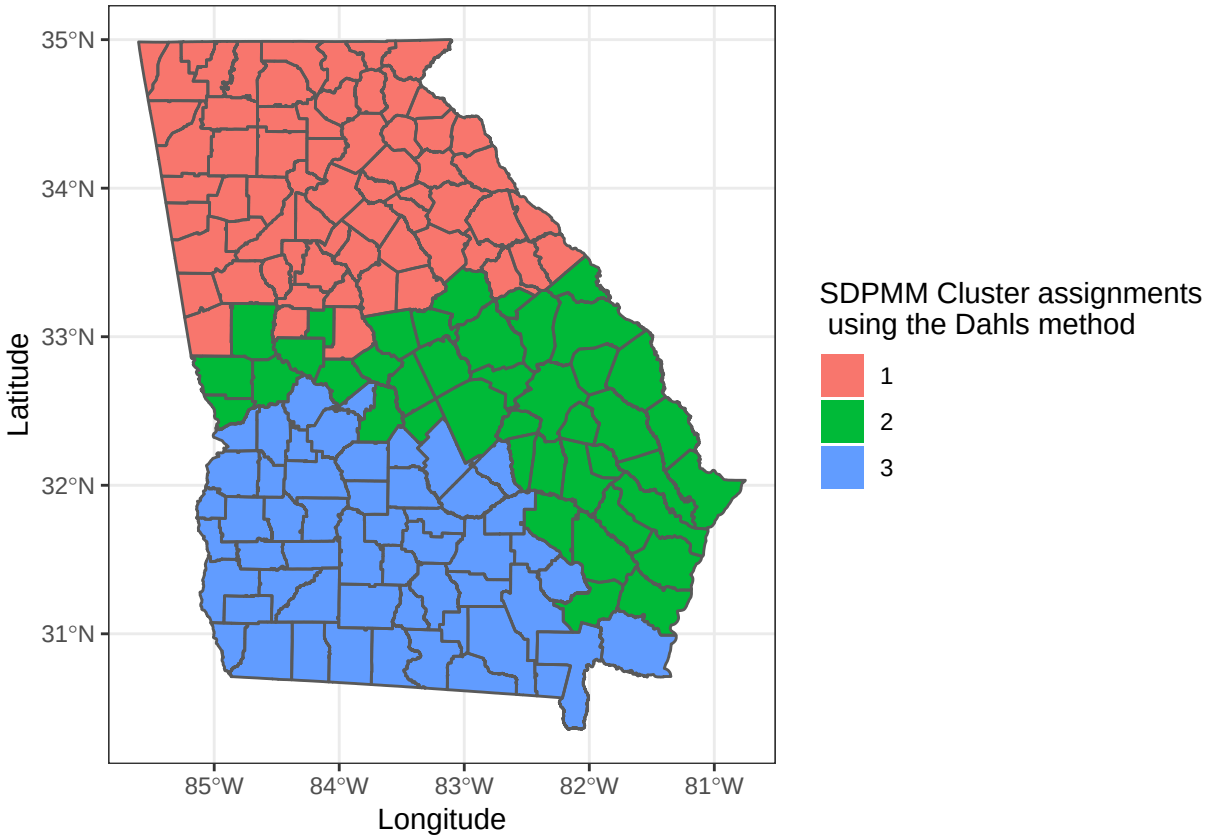
```
latentPE <- as.numeric(unlist(apply(latentZMat, 2, FUN = function(x) {
  return(DescTools::Mode(x)[1])
})))
# Check number of clusters, and number of regions in each cluster
latentPE<- ifelse(latentPE == 9, 1, ifelse(latentPE == 10, 2, ifelse(latentPE == 2, 3, latentPE)))
fossil::rand.index(asm, latentPE)
```

```
## [1] 0.8752488
```

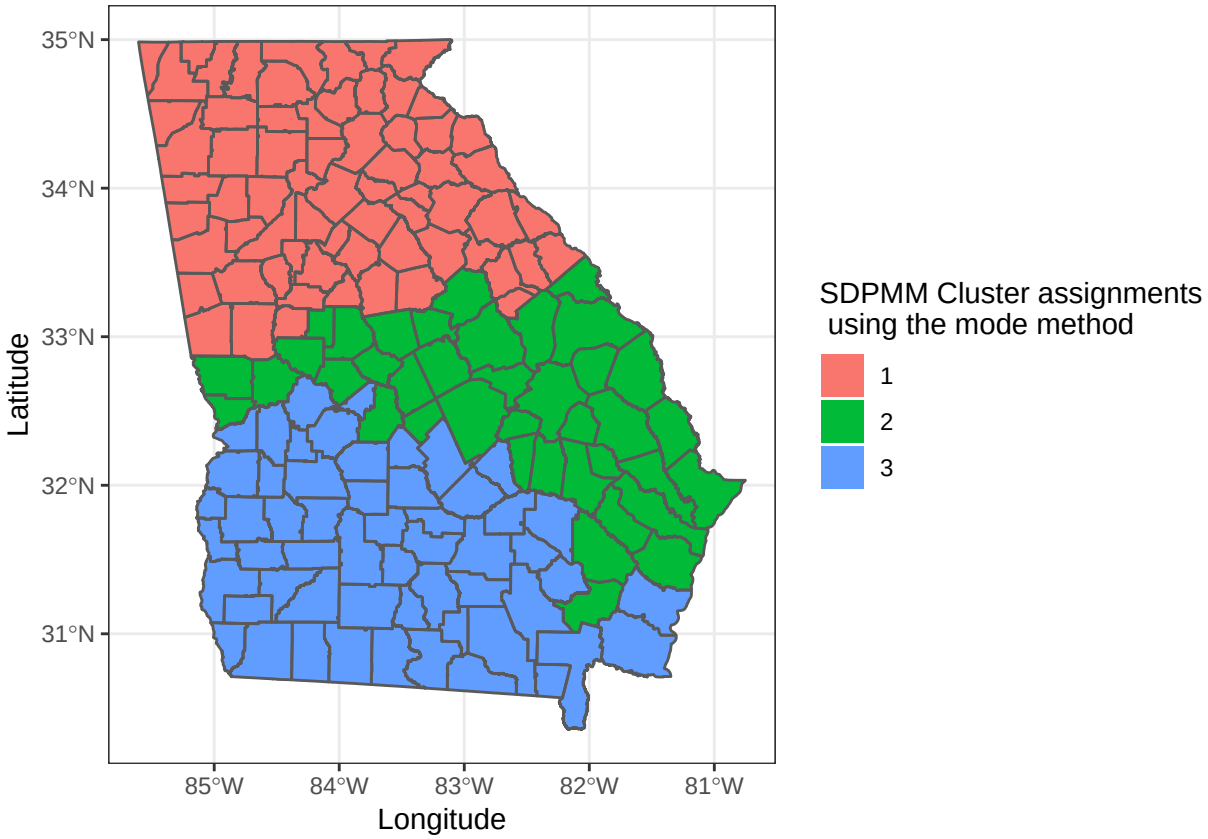
```
table(latentPE)
```

```
## latentPE
##  1  2  3
## 66 39 54
```

```
library(plyr)
library(dplyr)
library(sf)
Georgia <- read_sf("Georgia_dat.shp") %>% filter(!st_is_empty(.))

# Create a new dataset with finalCluster assignments
mydata_and_myMap <- cbind(Georgia, finalCluster,latentPE,asm)

# Create a ggplot map visualization
A<- ggplot() +
  xlab("Longitude") +
  ylab("Latitude") +
  theme_bw() +
  theme(legend.position = "right") +
  labs(fill = 'SDPMM 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)

A
```




```
B<- ggplot() +
  xlab("Longitude") +
  ylab("Latitude") +
  theme_bw() +
  theme(legend.position = "right") +
  labs(fill = 'SDPMM 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)


B
```



References

- 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/)
- 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.