Stan Model Script for Hyman et al. 2022

A. Challen Hyman

3/1/2022

R code

This is the raw code to run the Stan models

```
# Load required libraries
library(rstan)

# Load data
My.Data <- read.csv("Hyman et al. 2022 Processed Data.csv")

# Subset to required data
My.Data <- My.Data[which(My.Data$Year>1995 & My.Data$Year<2017),]

# Load neighborhood matrix
W <- as.matrix(read.csv('W.csv'))

# Matrix with number of neighbors in diagonal
D <- diag(colSums(W))


Node_matrix <- NULL
for (j in 1:ncol(W)){
  for (i in 1:nrow(W)){
    if(W[i,j] == 1){
      Edge_vector <- c(i,j)
      Node_matrix <- rbind(Node_matrix,Edge_vector)
    }
  }
}

N_edges <- length(which(W ==1))
Node_1 <- Node_matrix[,1]
Node_2 <- Node_matrix[,2]

## Set response variable (Count outcomes)
y <- as.integer(My.Data$Count)

## Number of polygons
N <- length(y)
```

```r
## Identity matrix
I <- diag(N)

# Create Design Matrix for models
My.Data$Pred <- log(My.Data$Pred100+My.Data$Pred200+1)
X <- cbind(1,
           as.numeric((My.Data$Secchi))*-1,
           as.numeric((My.Data$Seagrass)),
           as.numeric((My.Data$Marsh)),
           as.numeric((My.Data$Marsh))*as.numeric((My.Data$Secchi))*-1,
           as.numeric((My.Data$Pred)))

Offset <- as.numeric((My.Data$Tow_Dist))
My.Data$Management <- 2
My.Data$Management[which(My.Data$Year <2009)] <- 1

X <- cbind(X,lm(y~as.factor(My.Data$Management)+as.factor(My.Data$Tributary),
                x = T)$x[,2:4])

colnames(X) <- c("Int", "Secchi", "Seagrass",
                 "Marsh", "Marsh:Secchi", "log Pred",
                 "Management", "Rapp", "York")

# Create data list for rstan
dataList <- list('N' = N, # See data section  in 'CAR_POIS.stan' for definitions
                 'y' = y,
                 'W' = W,
                 'E' = Offset,
                 'D' = D,
                 'K' = ncol(X),
                 'N_edges' = N_edges,
                 'Node_1' = Node_1,
                 'Node_2' = Node_2,
                 'SP' = ncol(W),
                 'Year' = as.integer(My.Data$Year)-min(My.Data$Year)+1,
                 'T' = length(unique(My.Data$Year)),
                 'Group' = as.integer(as.factor(My.Data$Tributary)),
                 'G' = length(unique(My.Data$Tributary)),
                 'P_groups' = My.Data$ID,
                 'X' = X   # Predictor variable
)


# Run models
Model_1 <- stan('Hyman_Chapter_1_Model_1.stan', data = dataList,
                chains = 4, iter = 30000)
Model_1@stanmodel@dso <- new("cxxdso")
saveRDS(Model_1 , file = "Model_1_unscaled.rds")

Model_2 <- stan('Hyman_Chapter_1_Model_2.stan', data = dataList,
                chains = 4, iter = 30000)
Model_2@stanmodel@dso <- new("cxxdso")
saveRDS(Model_2 , file = "Model_2_unscaled.rds")
```

```r
Model_3a <- stan('Hyman_Chapter_1_Model_3a.stan', data = dataList,
                 chains = 4, iter = 30000)
Model_3a@stanmodel@dso <- new("cxxdso")
saveRDS(Model_3a , file = "Model_3a_unscaled.rds")

Model_3b <- stan('Hyman_Chapter_1_Model_3b.stan', data = dataList,
                 \chains = 4, iter = 30000)
Model_3b@stanmodel@dso <- new("cxxdso")
saveRDS(Model_3b , file = "Model_3b_unscaled.rds")

Model_4 <- stan('Hyman_Chapter_1_Model_4.stan', data = dataList,
                chains = 4, iter = 30000)
Model_4@stanmodel@dso <- new("cxxdso")
saveRDS(Model_4 , file = "Model_4_unscaled.rds")
```

Model 1

```stan
data {
  int N;
  int<lower = 0> K;                                // Number of predictors
  int<lower=1>   SP;                               // Number of polygons
  int<lower=0> y[N];                               // Count outcomes
  vector<lower=0>[N] E;                            // Offset
  matrix[N,K] X;                                   // Predictor matrix (includes int)
  int<lower=1, upper=SP> P_groups[N];              // Polygon id
}

transformed data {
  vector[N] log_E = log(E);                        // Transform offset to log scale
}

 parameters {
  vector[SP] theta;                                // Polygon random effect
  vector[K] beta;                                  // Coefficients of predictor matrix
  real<lower=0> tau_theta;                         // Global precision of random effects
}

model {
  for ( i in 1:N){
    y[i] ~ poisson_log(X[i,]*beta + theta[P_groups[i]]*inv(sqrt(tau_theta)) + log_E[i]);
  }
//Priors
  beta ~ normal(0,10);
  tau_theta ~ gamma(1,1);
  theta ~ normal(0, 1);
}
```

Model 2

```stan
data {
```

```
  int N;                                            // Number of observations
  int<lower = 0> K;                                 // Number of predictors
  int<lower = 0> T;                                 // Numbers of Years
  int<lower=1>   SP;                                // Number of polygons
  int<lower=0> y[N];                                // Count outcomes
  matrix[N,K] X;                                    // Predictor matrix (includes int)
  vector<lower=0>[N] E;                             // Offset
  int<lower=1, upper=T> Year[N];                    // Year id
  int<lower=1, upper=SP> P_groups[N];               // Polygon id
  matrix<lower=0, upper = 1>[SP,SP] W;              // Adjacency/neighborhood matrix
  matrix<lower=0>[SP,SP]  D;                        // Diagonal matrix
}

 transformed data {
  vector[N] log_E = log(E);                         // Transform offset to log scale
  vector[SP] zeros;
  zeros = rep_vector(0, SP);                        // Vector of zeros
}

 parameters {
  vector[SP] phi;                                   // Spatial random effect
  real<lower=0> tau_phi;                            // precision of spatial effects
  vector[K] beta;                                   // Coefficients of predictor matrix
  real<lower = 0, upper = 1> lambda;                // Spatial Autocorrelation parameter
}

 model {
 phi ~ multi_normal_prec(zeros, tau_phi^2 * (D - lambda*W));
 for ( i in 1:N){
    y[i] ~ poisson_log(X[i,]*beta + phi[P_groups[i]] + log_E[i]);
  }
// Priors
 tau_phi ~ gamma(1, 1);
 beta ~ normal(0,10);
}
```

Model 3a

```
data {
  int N;                                            // Number of observations
  int<lower = 0> K;                                 // Number of predictors
  int<lower = 0> T;                                 // Numbers of Years
  int<lower=1>   SP;                                // Number of polygons
  int<lower=0> y[N];                                // Count outcomes
  matrix[N,K] X;                                    // Predictor matrix (includes int)
  vector<lower=0>[N] E;                             // Offset
  int<lower=1, upper=T> Year[N];                    // Year id
  int<lower=1, upper=SP> P_groups[N];               // Polygon id
  matrix<lower=0, upper = 1>[SP,SP] W;              // Adjacency/neighborhood matrix
  matrix<lower=0>[SP,SP]  D;                        // Diagonal matrix
 }

transformed data {
```

```
  vector[N] log_E = log(E);                     // Transform offset to log scale
  vector[SP] zeros;
  zeros = rep_vector(0, SP);                    // Vector of zeros
}

 parameters {
  vector[SP] phi;                               // Spatial random effect
  real<lower=0> tau_phi;                        // precision of spatial effects
  vector[K] beta;                               // Coefficients of predictor matrix
  real<lower = 0, upper = 1> lambda;            // Spatial Autocorrelation parameter
  real<lower = 0, upper = 1> rho_global;        // Global autoregression term
  real<lower=0> tau_eta;                        // noise scale on latent state evolution
  vector[T] u;                                  // Latent state ("error term") matrix
}

  transformed parameters {
  real<lower=0> sigma_phi = inv(sqrt(tau_phi));  // convert precision to sigma (spatial)
  real<lower=0> sigma_eta = inv(sqrt(tau_eta));  // convert precision to sigma (temporal)
}
 model {
 phi ~ multi_normal_prec(zeros, tau_phi^2 * (D - lambda*W));
 for ( i in 1:N){
    y[i] ~ poisson_log(X[i,]*beta + phi[P_groups[i]] + log_E[i] + u[Year[i]]);
  }
  u[1] ~ normal(0, 5);
  for (t in 2:T){
      u[t] ~ normal(rho_global * u[t - 1], sigma_eta);
  }

// Priors
 tau_phi ~ gamma(1,1);
 tau_eta ~ gamma(1,1);
 beta ~ normal(0,10);
}
```

Model 3b

```
data {
  int N;
  int<lower = 0> K;                             // Number of predictors
  int<lower = 0> T;                             // Numbers of Years
  int<lower = 0> G;                             // Number of AR groups
  int<lower=1>   SP;                            // Number of polygons

  int<lower=0> y[N];                            // Count outcomes
  vector<lower=0>[N] E;                         // Offset
  matrix[N,K] X;                                // Predictor matrix (includes int)

  int<lower=1, upper=T> Year[N];                // Year id
  int<lower=1, upper=T> Group[N];               // Group id
  int<lower=1, upper=SP> P_groups[N];           // Polygon id

  matrix<lower=0, upper = 1>[SP,SP] W;          // Adjacency/neighborhood matrix
```

```
  matrix<lower=0>[SP,SP]  D;                      // Diagonal matrix

 }

 transformed data {
  vector[N] log_E = log(E);                       // Transform offset to log scale
  vector[SP] zeros;
  zeros = rep_vector(0, SP);                      // Vector of zeros
}

 parameters {
  vector[SP] phi;                                 // Spatial random effect
  vector[G-1] r_raw;                              // Ordinary random effects component for AR1 groups
  real<lower=0> tau_phi;                          // Precision of spatial effects
  vector[K] beta;                                 // Coefficients of predictor matrix
  real<lower=0> tau_eta;                          // noise scale on latent state evolution
  matrix[T,G] u;                                  // Latent state ("error term") matrix
  real<lower = 0, upper = 1> lambda;              // Spatial autocorrelation parameter
  real<lower = 0, upper = 1> rho_global;          // Global temporal autocorrelation parameter
}

  transformed parameters {
  real<lower=0> sigma_phi = inv(sqrt(tau_phi));  // convert precision to sigma (spatial)
  real<lower=0> sigma_eta = inv(sqrt(tau_eta));  // convert precision to sigma (temporal)
  vector[G] rho_local;                            // Local autoregression term
  vector[G] r = append_row(r_raw, -sum(r_raw));  // hard zum-to-zero of r offset terms
  for(i in 1:G) {
   rho_local[i] = inv_logit(logit(rho_global) + r[i]); // Derive local rho's from global rho value
  }

}

//Model
 model {
 phi ~ multi_normal_prec(zeros, tau_phi^2 * (D - lambda*W));
 for ( i in 1:N){
    y[i] ~ poisson_log(X[i,]*beta + phi[P_groups[i]] + log_E[i] + u[Year[i], Group[i]]);
      }
  u[1,] ~ normal(0, 5);
  for (t in 2:T){
    for (g in 1:G){
      u[t,g] ~ normal(rho_local[g] * u[t - 1,g], sigma_eta);
    }
  }

// Priors
 r_raw ~ normal(0,0.25);
 tau_phi ~ gamma(1,1);
 tau_eta ~ gamma(1,1);
 beta ~ normal(0,10);
}
```

Model 4

```
data {
  int N;                                        // Number of observations
  int<lower = 0> K;                             // Number of predictors
  int<lower = 0> T;                             // Numbers of years
  int<lower=1>   SP;                            // Number of polygons
  int<lower=0> y[N];                            // Count outcomes
  matrix[N,K] X;                                // Predictor matrix (includes int)
  vector<lower=0>[N] E;                         // Offset
  int<lower=1, upper=T> Year[N];                // Year id
  int<lower=1, upper=SP> P_groups[N];           // Polygon id
   matrix<lower=0, upper = 1>[SP,SP] W;         // Adjacency/neighborhood matrix
   matrix<lower=0>[SP,SP]  D;                   // Diagonal matrix

 }

 transformed data {
  vector[N] log_E = log(E);                     // Transform offset to log scale
  vector[SP] zeros;
  zeros = rep_vector(0, SP);                    // Vector of zeros
}

 parameters {
  matrix[SP,T] phi;                             // Temporally varying spatial random effect
  real<lower=0> tau_phi;                        // precision of spatial effects
  vector[K] beta;                               // Coefficients of predictor matrix
  real<lower = 0, upper = 1> lambda;            // Spatial Autocorrelation parameter
  real<lower = 0, upper = 1> rho;               // Global autoregression term
}

 model {
 phi[,1] ~ multi_normal_prec(zeros, tau_phi^2 * (D - lambda*W));
 for (t in 2:T){
      phi[,t] ~ multi_normal_prec(rho*phi[,t-1], tau_phi^2 * (D - lambda*W));
 }
 for (n in 1:N){
      y[n] ~ poisson_log(X[n,]*beta + phi[P_groups[n],Year[n]] + log_E[n]);
 }
//Priors
 tau_phi ~ gamma(1,1);
 beta ~ normal(0,10);
}
```