Bayesian hierarchical spatial models for disease mapping in the presence of missing covariates

Appendix A: R codes for analyzing the Scottish lip cancer dataset (single covariate)

```
library(rstan)
library(INLA)
library(Matrix)
library(sf)
library(spdep)
set.seed(123)
source("mungeCARdata4stan.R")
source("scotland_data.R")
y = data$y;
x = 0.1 * data$x;
E = data$E;
nbs = mungeCARdata4stan(data$adj, data$num);
N = nbs$N;
node1 = nbs$node1;
node2 = nbs$node2;
N_edges = nbs$N_edges;
#Build the adjacency matrix using INLA library functions
adj.matrix = sparseMatrix(i=nbs$node1,j=nbs$node2,x=1,symmetric=TRUE)
#The ICAR precision matrix (note! This is singular)
Q=  Diagonal(nbs$N, rowSums(adj.matrix)) - adj.matrix
#Add a small jitter to the diagonal for numerical stability (optional but recommended)
Q_pert = Q + Diagonal(nbs$N) * max(diag(Q)) * sqrt(.Machine$double.eps)

# Compute the diagonal elements of the covariance matrix subject to the
# constraint that the entries of the ICAR sum to zero.
#See the inla.qinv function help for further details.
Q_inv = inla.qinv(Q_pert, constr=list(A = matrix(1,1,nbs$N),e=0))
```

```
#Compute the geometric mean of the variances, which are on the diagonal of Q.inv
scaling_factor = exp(mean(log(diag(Q_inv))))
# Introduce missingness in X
miss_prop <- 0.10
miss_idx <- sample(seq_len(N), size = floor(miss_prop * N))
obs_idx  <- setdiff(seq_len(N), miss_idx)

X_obs_values <- x[obs_idx]
xmiss=x[miss_idx]

# 8. Stan data list
stan_data <- list(
  N = N,
  N_edges = N_edges,
  node1 = node1,
  node2 = node2,
  N_obs = length(obs_idx),
  N_miss = length(miss_idx),
  idx_obs = obs_idx,
  idx_miss = miss_idx,
  X_obs_values = X_obs_values,
  Y = y,
  E = E,
  scaling_factor = scaling_factor
)
stan_model_code <- "
data {
  int<lower=1> N;
  int<lower=1> N_edges;
  int<lower=1, upper=N> node1[N_edges];
  int<lower=1, upper=N> node2[N_edges];
```

```
  int<lower=0> N_obs;
  int<lower=1, upper=N> idx_obs[N_obs];
  vector[N_obs] X_obs_values;
  int<lower=0> N_miss;
  int<lower=1, upper=N> idx_miss[N_miss];
  int<lower=0> Y[N];
  vector<lower=0>[N] E;
  real<lower=0> scaling_factor;
}

parameters {
  real alpha;
  real beta;
  // BYM2 parameters for X (spatial structure)
  real mu_X;
  vector[N] phi_X;
  vector[N] theta_X;
  real<lower=0> sigma_X;
  real<lower=0, upper=1> rho_X;
  real<lower=0> noise_X;  // Measurement error for observed X

  // Missing values (will be estimated)
  vector[N_miss] X_miss;
  // Outcome parameters
  vector[N] phi_u;
  vector[N] theta_u;
  real<lower=0> sigma_u;
  real<lower=0, upper=1> rho_u;
}
transformed parameters {
  vector[N] X_full;
```

```
  vector[N] X_true;
  vector[N] u;
  vector[N] log_risk;
    // Underlying spatial process for X
  X_true = mu_X + sigma_X * (
    sqrt(rho_X) * phi_X / sqrt(scaling_factor) +
    sqrt(1 - rho_X) * theta_X
  );
    // Spatial effect for outcome
  u = sigma_u * (
    sqrt(rho_u) * phi_u / sqrt(scaling_factor) +
    sqrt(1 - rho_u) * theta_u
  );

  // Full X vector for outcome model
  X_full[idx_obs] = X_obs_values;      // Fixed observed values
  X_full[idx_miss] = X_miss;           // Estimated missing values

  log_risk = log(E) + alpha + beta * X_full + u;
}
model {
  // Likelihood for observed X - with small measurement error
  // This identifies the spatial parameters using observed data
  X_obs_values ~ normal(X_true[idx_obs], noise_X);

  // Prior for missing values - they come from same spatial process
  X_miss ~ normal(X_true[idx_miss], noise_X);
    // Outcome model
  Y ~ poisson_log(log_risk);
    // ICAR priors
  target += -0.5 * dot_self(phi_X[node1] - phi_X[node2]);
```

```
  target += -0.5 * dot_self(phi_u[node1] - phi_u[node2]);

  // Soft constraints
  sum(phi_X) ~ normal(0, 0.001 * N);
  sum(phi_u) ~ normal(0, 0.001 * N);
    // Independent priors
  theta_X ~ std_normal();
  theta_u ~ std_normal();

  // Weakly informative priors
  mu_X ~ normal(mean(X_obs_values), 2);
  alpha ~ normal(0, 1);
  beta ~ normal(0, 1);
  sigma_X ~ normal(0, 1);
  sigma_u ~ normal(0, 1);
  noise_X ~ normal(0, 0.1);  // Small measurement error
  rho_X ~ beta(0.5, 0.5);
  rho_u ~ beta(0.5,0.5);
}
generated quantities {
  vector[N] mu = exp(log_risk);
  vector[N] y_rep;
  vector[N] log_lik;
    for (n in 1:N) {
    real capped_log_risk = fmin(log_risk[n], 20.0);
    y_rep[n] = poisson_log_rng(capped_log_risk);
    log_lik[n] = poisson_log_lpmf(Y[n] | log_risk[n]);
  }
}
"
```

```
fit <- stan(model_code = stan_model_code, data = stan_data, iter = 4000,  warmup = 2000, chains = 4, cores = 4, control = list(adapt_delta = 0.95, max_treedepth = 12))
```

Appendix B: R codes for analyzing the North Carolina SIDS data set (two covariates)

```
library(rstan)
library(INLA)
library(Matrix)
library(sf)
library(spdep)
# Load data
nc <- st_read(system.file("shape/nc.shp", package = "sf"))
# Response and offset
Y <- nc$SID79
total_cases <- sum(nc$SID79)
total_births <- sum(nc$BIR79)
E <- nc$BIR79 * (total_cases / total_births)
# Covariate 2: Previous SIDS rate (NOT count!)
X1 <- scale(nc$BIR79 / nc$AREA)       # Population density (p = 0.0039)
X2 <- scale(nc$SID74 / nc$BIR74)      # Previous SIDS rate (p = 0.0016)
# 4. Create spatial adjacency
N <- length(Y)
nb <- poly2nb(nc)
W <- nb2mat(nb, style = "B")
node1 <- c()
node2 <- c()
for(i in 1:(N-1)) {
  for(j in (i+1):N) {
    if(W[i,j] == 1) {
      node1 <- c(node1, i)
      node2 <- c(node2, j)
    }
```

```
  }
}
N_edges <- length(node1)
# 5. Compute scaling factor for ICAR
adj.matrix <- sparseMatrix(i = node1, j = node2, x = 1, symmetric = TRUE)
Q <- Diagonal(N, rowSums(adj.matrix)) - adj.matrix
Q_pert <- Q + Diagonal(N) * max(diag(Q)) * sqrt(.Machine$double.eps)
Q_inv <- inla.qinv(Q_pert, constr = list(A = matrix(1, 1, N), e = 0))
scaling_factor <- exp(mean(log(diag(Q_inv))))
# 6. Introduce missingness (10% in each covariate)
set.seed(123)
k=0.10;
miss_idx1 <- sample(seq_len(N), size = floor(k * N))
miss_idx2 <- sample(seq_len(N), size = floor(k * N))
obs_idx1 <- setdiff(seq_len(N), miss_idx1)
obs_idx2 <- setdiff(seq_len(N), miss_idx2)
# 7. Prepare Stan data
stan_data <- list(
  N = N,
  N_edges = N_edges,
  node1 = node1,
  node2 = node2,
  N_obs1 = length(obs_idx1),
  N_miss1 = length(miss_idx1),
  idx_obs1 = obs_idx1,
  idx_miss1 = miss_idx1,
  X1_obs_values = X1[obs_idx1],
  N_obs2 = length(obs_idx2),
  N_miss2 = length(miss_idx2),
  idx_obs2 = obs_idx2,
  idx_miss2 = miss_idx2,
```

```
  X2_obs_values = X2[obs_idx2],
  Y = Y,
  E = E,
  scaling_factor = scaling_factor
)
# 8. Stan model (simplified version for faster computation)
stan_model_final <- "
data {
  int<lower=1> N;
  int<lower=1> N_edges;
  int<lower=1, upper=N> node1[N_edges];
  int<lower=1, upper=N> node2[N_edges];
   int<lower=0> N_obs1;
  int<lower=1, upper=N> idx_obs1[N_obs1];
  vector[N_obs1] X1_obs_values;
  int<lower=0> N_miss1;
  int<lower=1, upper=N> idx_miss1[N_miss1];
   int<lower=0> N_obs2;
  int<lower=1, upper=N> idx_obs2[N_obs2];
  vector[N_obs2] X2_obs_values;
  int<lower=0> N_miss2;
  int<lower=1, upper=N> idx_miss2[N_miss2];
  int<lower=0> Y[N];
  vector<lower=0>[N] E;
  real<lower=0> scaling_factor;
}
parameters {
  real alpha;
  real beta1;
  real beta2;
  real mu_X1;
```

```
  vector[N] phi_X1;
  vector[N] theta_X1;
  real<lower=0> sigma_X1;
  real<lower=0, upper=1> rho_X1;
  real<lower=0> noise_X1;
    real mu_X2;
  vector[N] phi_X2;
  vector[N] theta_X2;
  real<lower=0> sigma_X2;
  real<lower=0, upper=1> rho_X2;
  real<lower=0> noise_X2;
  vector[N_miss1] X1_miss;
  vector[N_miss2] X2_miss;
   vector[N] phi_u;
  vector[N] theta_u;
  real<lower=0> sigma_u;
  real<lower=0, upper=1> rho_u;
}
transformed parameters {
  vector[N] X1_full;
  vector[N] X2_full;
  vector[N] X1_true;
  vector[N] X2_true;
  vector[N] u;
  vector[N] log_risk;
    X1_true = mu_X1 + sigma_X1 * (sqrt(rho_X1) * phi_X1 / sqrt(scaling_factor) +
                                  sqrt(1 - rho_X1) * theta_X1);
  X2_true = mu_X2 + sigma_X2 * (sqrt(rho_X2) * phi_X2 / sqrt(scaling_factor) +
  sqrt(1 - rho_X2) * theta_X2);

  u = sigma_u * (sqrt(rho_u) * phi_u / sqrt(scaling_factor) + sqrt(1 - rho_u) * theta_u);
```

```
  X1_full[idx_obs1] = X1_obs_values;
  X1_full[idx_miss1] = X1_miss;
  X2_full[idx_obs2] = X2_obs_values;
  X2_full[idx_miss2] = X2_miss;

  log_risk = log(E) + alpha + beta1 * X1_full + beta2 * X2_full + u;
}
model {
  // Covariate models
  X1_obs_values ~ normal(X1_true[idx_obs1], noise_X1);
  X1_miss ~ normal(X1_true[idx_miss1], noise_X1);
  X2_obs_values ~ normal(X2_true[idx_obs2], noise_X2);
  X2_miss ~ normal(X2_true[idx_miss2], noise_X2);
    // Outcome model
  Y ~ poisson_log(log_risk);

  // ICAR priors
  target += -0.5 * dot_self(phi_X1[node1] - phi_X1[node2]);
  target += -0.5 * dot_self(phi_X2[node1] - phi_X2[node2]);
  target += -0.5 * dot_self(phi_u[node1] - phi_u[node2]);
    // Sum-to-zero constraints
  sum(phi_X1) ~ normal(0, 0.001 * N);
  sum(phi_X2) ~ normal(0, 0.001 * N);
  sum(phi_u) ~ normal(0, 0.001 * N);
    // Priors
  theta_X1 ~ std_normal();
  theta_X2 ~ std_normal();
  theta_u ~ std_normal();

  mu_X1 ~ normal(mean(X1_obs_values), 0.5);
  mu_X2 ~ normal(mean(X2_obs_values), 0.5);
```

```
  alpha ~ normal(0, 1);
  beta1 ~ normal(0, 1);
  beta2 ~ normal(0, 1);
   sigma_X1 ~ normal(0, 0.5);
  sigma_X2 ~ normal(0, 0.5);
  sigma_u ~ normal(0, 1);
   noise_X1 ~ normal(0, 0.1);
  noise_X2 ~ normal(0, 0.1);
   rho_X1 ~ beta(0.5, 0.5);
  rho_X2 ~ beta(0.5, 0.5);
  rho_u ~ beta(0.5, 0.5);
}
generated quantities {
  vector[N] mu = exp(log_risk);
  vector[N] y_rep;
  vector[N] log_lik;
  for (n in 1:N) {
    real capped_log_risk = fmin(log_risk[n], 20.0);
    y_rep[n] = poisson_log_rng(capped_log_risk);
    log_lik[n] = poisson_log_lpmf(Y[n] | log_risk[n]);
  }
}
"
fit_final <- stan(model_code = stan_model_final, data = stan_data, iter = 4000,  warmup = 2000,
chains = 4, cores = 4, control = list(adapt_delta = 0.95, max_treedepth = 12))
```