Nearest Neighbor Gaussian Processes (NNGP) based models in Stan

Lu Zhang

11/5/2017

1 NNGP Based Models

Nearest Neighbor Gaussian Processes (NNGP) based models is a family of highly scalable Gaussian processes based models. In brief, NNGP extends the Vecchia's approximation (Vecchia (1988)) to a process using conditional independence given information from neighboring locations. In this section, I will briefly review response and random-effects NNGP models. For more details of NNGP, please refer to Datta et al. (2016).

1.1 Spatial regression model

We envision a spatial regression model at any location s

$$y(s) = m_\theta(s) + w(s) + \epsilon(s) \ , \quad \epsilon(s) \overset{iid}{\sim} \mathcal{N}(0, \tau^2) \tag{1}$$

where, usually, $m_\theta(s) = x(s)^\top \beta$ and $w(s)$ is a latent spatial process capturing spatial dependence. Let S be the set of n observed locations. If we model the process $w(s)$ with a Gaussian process $w(s) \sim GP(0, C_\theta(\cdot, \cdot))$, then a customary Bayesian hierarchical models for observations on $S = \{s_1, \dots, s_N\}$ can be constructed as

$$p(\theta) \ \times \ \mathcal{N}(w(S) \mid 0, C_\theta(S, S)) \ \times \ \mathcal{N}(y(S) \mid m_\theta(S) + w(S), \tau^2 I_n) \tag{2}$$

Under the assumption that $w(s)$ follows a Gaussian process, one can construct the outcome process $y(s)$ using convolution over the latent process $w(s)$. If we integrate out $w(s)$ and model response $y(s)$ with a Gaussian process directly, then the parameters set will collapse from $\{\theta, w\}$ to $\{\theta\}$. In a Bayesian setting, θ will be sampled from its posterior distribution

$$p(\theta \mid y(S)) \ \propto \ p(\theta) \ \times \ \mathcal{N}(y(S) \mid m_\theta(S), C_\theta(S, S) + \tau^2 I_n) \tag{3}$$

To distinguish these two models, we shall call the former as a random-effects model, and the latter as a response model.

1.2 Random-effects and Response NNGP models

Nearest neighbor Gaussian process (NNGP) provides an alternative to the Gaussian process in the models discussed in the preceding subsection. The likelihoods of two models basing on NNGP derived from the original Gaussian process coincide with the Vecchia's approximation (Vecchia 1988) of the original models. In particular, a random-effects NNGP model has a posterior distribution proportional to

$$p(\theta) \times \mathcal{N}(w(S) \mid C^*) \times \mathcal{N}(y(S) \mid m_\theta(S) + w(S), \tau^2 I_n) \tag{4}$$

where $C^{*-1} = (I - A)^\top D^{-1}(I - A)$ is the precision matrix of the latent process $w(s)$ over S. Here A is sparse and strictly lower triangular with at most M ($M \ll N$) non-zero entries in each row, and D is a diagonal matrix. One can readily calculate the determinant of C^* by the product of the diagonal elements in D. The likelihood of $w(S)$ based on precision matrix C^{*-1} serves as a good approximation to the likelihood of $w(s)$ in (2), while the storage and computational burden of the former is linear in N.

A response NNGP model yields posterior distribution:

$$p(\theta \mid y(S)) \propto p(\theta) \times \mathcal{N}(y(S) \mid m_\theta(S), \{C_\theta + \tau^2 I\}^*) \tag{5}$$

where $\{C_\theta + \tau^2 I\}^{*-1} = (I - A)^\top D^{-1}(I - A)$, analogous to random-effects NNGP model, can be treated as an approximation of $\{C_\theta(S, S) + \tau^2 I_n\}^{-1}$. Notice that although one can obtain the response model by integrating out latent process in a random-effects model, the corresponding NNGP model doesn't have this property.

1.3 Construction of A, D

The details of the Matrix A, D and two models can be found in Finley et al. (2017). Here we use the response NNGP model to show how to construct matrix A and D. Let $N(s_i)$ be at most M closest points to s_i among the locations indexed less that i. For the i th row ($i > 1$) of A, the nonzero entries appear in the positions indexed by $N(s_i)$ are obtained as row vectors

$$A(i, N(s_i)) = C_\theta(s_i, N(s_i))(C_\theta(N(s_i), N(s_i)) + \tau^2 I)^{-1} \tag{6}$$

And the i th element on the diagonal of D satisfies

$$D(i, i) = C_\theta(s_i, s_i) + \tau^2 - C_\theta(s_i, N(s_i))(C_\theta(N(s_i), N(s_i)) + \tau^2 I)^{-1} C_\theta(N(s_i), s_i)$$

These equations are derived from the distribution of $E[y(s_i)|y(N(s_i))]$. The nonzero entries in each row of A are precisely the weights obtained by predicting $y(s_i)$, or "kriging", based upon the values of $y(s)$ at neighboring locations, i.e.,

$N(s_i)$. And the diagonal elements in D are the variance of $y(s_i)$ conditioning on its' neighbors in the "past" $y(N(s_i))$.

2 Code NNGP Based Model in Stan

In this section, I will use a simulation data to show how to code NNGP based models efficiently in Stan.

2.1 Intro of simulation data

We generated response Y along with a covariate x at $n = 500$ randomly sited locations in a unit square domain by the following model:

$$y(s) = \beta_0 + x(s)\beta_1 + w(s) + \epsilon(s), \qquad \epsilon(s) \sim N(0, \tau^2) \tag{8}$$

where the zero-centered spatial random effect $w(s)$ were sampled from a Gaussian process with a covariance function C_θ specified by exponential:

$$C_\theta(s_i, s_j) = \sigma^2 \exp(-\phi||s_i - s_j||), \quad s_i, s_j \in S \tag{9}$$

The predictor x were generated from $N(0, 1)$. The setting of parameters is listed in the code.

```
rmvn <- function(N, mu = 0, V = matrix(1)){
  P <- length(mu)
  if(any(is.na(match(dim(V), P))))
    stop("Dimension problem!")
  D <- chol(V)
  t(matrix(rnorm(N * P), ncol = P) %*% D + rep(mu, rep(N,
P)))
}

set.seed(1234)
N <- 500
coords <- cbind(runif(N), runif(N))
X <- as.matrix(cbind(1, rnorm(N)))
B <- as.matrix(c(1, 5))
sigma.sq <- 2
tau.sq <- 0.1
phi <- 3 / 0.5

D <- as.matrix(dist(coords))
R <- exp(-phi*D)
w <- rmvn(1, rep(0, N), sigma.sq*R)
Y <- rnorm(N, X %*% B + w, sqrt(tau.sq))
```

2.2 Data block in Stan

The following block shows the elements needed for NNGP based models

```
data {
    int<lower=1> N;
    int<lower=1> M;
    int<lower=1> P;
    vector[N] Y;
    matrix[N, P + 1] X;
    int NN_ind[N - 1, M];
    matrix[N - 1, M] NN_dist;
    matrix[N - 1, (M * (M - 1) / 2)] NN_distM;
}
```

Here the design matrix X contains an initial column of 1s, P is the number of regression coefficients, and M is the number of nearest neighbors (maximum number of elements in each row of $n \times n$ sparse matrix A). Notice that we provide three matrices `NN_ind`, `NN_dist` and `NN_distM`:

`NN_ind` is a two-dimensional array of indices whose $i-1$th row shows at most M closest points to s_i among the locations indexed less that i.

`NN_dist` is a matrix whose $i-1$th row contains the distance of ith location to its selected neighbors.

`NN_dist` is a matrix whose $i-1$th row contains the strictly lower triangular part of the distance matrix of the selected neighbors of ith location.

These three matrices are required for constructing the sparse lower triangular matrix A, and the diagonal matrix D. Since they are fixed across the MCMC updates, we recommend user to provide them in the data segment. Next, I will show how to efficiently generate the matrices listed above.

2.2.1 Build neighbor index

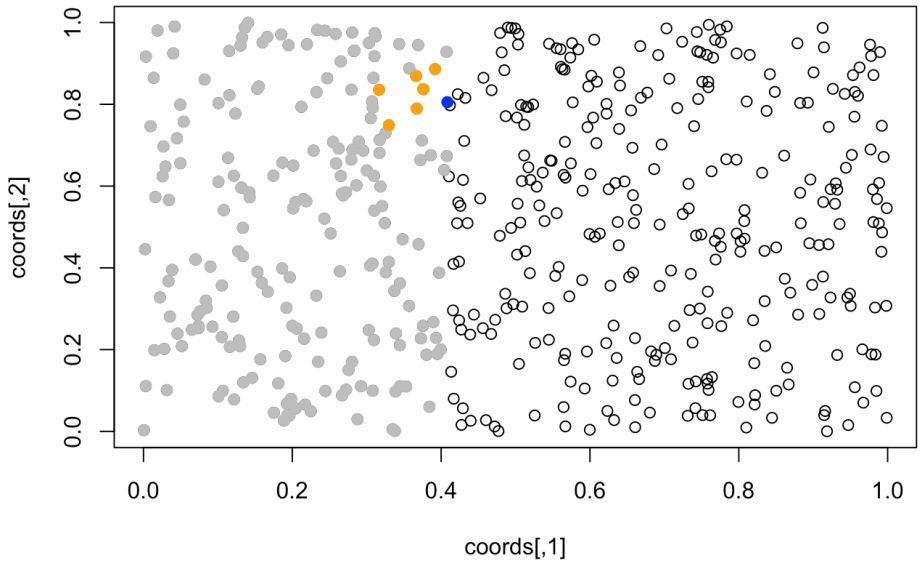
File “NNmatrix.R” provides a wrapper function in R that uses package spNNGP, which has a fast algorithm of building neighbor index, to generate the required matrices in the Stan data segment. Here `n.omp.threads` indicates the number of threads to use for parallel processing.

```
source("NNmatrix.R")
M = 6                  # Number of Nearest Neighbors
NN.matrix <- NNMatrix(coords = coords, n.neighbors = M, n
.omp.threads = 2)
str(NN.matrix)
```

2.2.2 Check Neighbors (for fun)

We can use function `Check_Neighbors` in "NNmatrix.R" for checking the nearest neighbor index. It is important to point out that `NNMatrix` sorts coordinates on the first column before building the neighbor index. Thus we should use the sorted response and design matrix instead of the raw data in the data block.

```
Check_Neighbors(NN.matrix$coords.ord, n.neighbors = M, NN
.matrix, ind = 200)
```



2.3 Parameter and model block in Stan

We assign Gaussian priors for σ and τ, a uniform prior for ϕ and a Gaussian prior for $\beta = \{\beta_0, \beta_1\}$. The following is the parameter and model block for a random-effects NNGP model.

```
  parameters{
      vector[P + 1] beta;
      real<lower = 0> sigma;
      real<lower = 0> tau;
      real<lower = ap, upper = bp> phi;
      vector[N] w;
  }

  transformed parameters {
      real sigmasq = sigma^2;
      real tausq = tau^2;
  }

  model{
      beta ~ multi_normal_cholesky(uB, L_VB);
      sigma ~ normal(0, ss);
      tau ~ normal(0, st);
      w ~ nngp_w(sigmasq, phi, NN_dist, NN_distM, NN_ind,
N, M);
      Y ~ normal(X * beta + w, tau);
  }
```

A small modification will make the code work for a response NNGP model:

```
  parameters{
      vector[P + 1] beta;
      real<lower = 0> sigma;
      real<lower = 0> tau;
      real<lower = ap, upper = bp> phi;
  }

  transformed parameters {
      real sigmasq = sigma^2;
      real tausq = tau^2;
  }

  model{
      beta ~ multi_normal_cholesky(uB, L_VB);
      sigma ~ normal(0, ss);
      tau ~ normal(0, st);
      Y ~ nngp(X * beta, sigmasq, tausq, phi, NN_dist, NN
_distM, NN_ind, N, M);
  }
```

Here, the user-defined functions `nngp_w` and `nngp` will be given in the simulation study section.

2.4 User-defined likelihood function for NNGP models

The hardest part in coding NNGP in Stan is the user-defined likelihood, specifically, the function `nngp_w` and `nngp` in the last subsection. Here we use `nngp` to illustrate the main idea of coding NNGP likelihood.

The log-likelihood of $y(S)$ in (5) is given by:

$$-\frac{1}{2}\sum_{i=1}^{N}\log D_{ii} - \frac{1}{2}(y(S) - X(S)^\top\beta)^\top(I-A)^T D^{-1}(I-A)(y(S) - X(S)^\top\beta)$$

In the code below, vector `U` saves the results of $(I-A)(y(S) - X(S)^\top\beta)$, and vector `V` saves all the diagonal elements of Matrix D scaled by σ^2. With this notation, the log-likelihood can be simplified as

$$-\frac{1}{2}\{\sum_{i=1}^{N}\log V_i + N\log(\sigma^2) + \frac{1}{\sigma^2}U^\top(U./V)\} \tag{10}$$

where all the elements in the likelihood are vectors.

In the calculation of vector $U = (I-A)(y(S) - X(S)^\top\beta)$, since we know that matrix A has at most M nonzero elements and the index of nonzero elements is given in `NN_ind`, there is no need for saving the $n \times n$ matrix $I - A$. Instead, we use a for loop to calculate U. Within each iteration, we first use `NN_dist` and `NN_distM` along with the updated parameter to obtain $A(i, N(s_i))$ by (7) and $D(i,i)$ by (6), then use `NN_ind` and $y(S) - X(S)^\top\beta$ to calculate the i th element of U. The flop required in each iteration is in the order of M^3.

```
  functions{
      real nngp_lpdf(vector Y, vector X_beta, real sigmas
q, real tausq,
                     real phi, matrix NN_dist, matrix NN_
distM, int[,] NN_ind,
                     int N, int M){

          vector[N] V;
          vector[N] YXb;
          vector[N] U;
          int dim;
          int h;
          real kappa_p_1;
          real out;
          kappa_p_1 = tausq / sigmasq + 1;
          YXb = Y - X_beta;
          U = YXb;
```

```
          for (i in 2:N) {
              matrix[ i < (M + 1) ? (i - 1) : M, i < (M +
1) ? (i - 1): M]
              iNNdistM;
              matrix[ i < (M + 1) ? (i - 1) : M, i < (M +
1) ? (i - 1): M]
              iNNCholL;
              vector[ i < (M + 1) ? (i - 1) : M] iNNcorr;
              vector[ i < (M + 1) ? (i - 1) : M] v;
              row_vector[i < (M + 1) ? (i - 1) : M] v2;
              dim = (i < (M + 1))? (i - 1) : M;

              if(dim == 1){iNNdistM[1, 1] = kappa_p_1;}
              else{
                  h = 0;
                  for (j in 1:(dim - 1)){
                      for (k in (j + 1):dim){
                          h = h + 1;
                          iNNdistM[j, k] = exp(- phi * NN
_distM[(i - 1), h]);
                          iNNdistM[k, j] = iNNdistM[j, k]
;
                      }
                  }
                  for(j in 1:dim){
                      iNNdistM[j, j] = kappa_p_1;
                  }
              }

              iNNCholL = cholesky_decompose(iNNdistM);
              for (j in 1: dim){
                  iNNcorr[j] = exp(- phi * NN_dist[(i - 1
), j]);
              }

             v = mdivide_left_tri_low(iNNCholL, iNNcorr);

             V[i] = kappa_p_1 - dot_self(v);

             v2 = mdivide_right_tri_low(v', iNNCholL);

             for (j in 1:dim){
                  U[i] = U[i] - v2[j] * YXb[NN_ind[(i - 1
), j]];
              }
          }
```

```
          V[1] = kappa_p_1;
          out = - 0.5 * ( 1 / sigmasq * dot_product(U, (U
./ V)) +
                              sum(log(V)) + N * log(sigmasq))
;
          return out;
      }
  }
```

3 Simulation study

Now let's run the NNGP based models for the simulation data in the last section. First set parameters of priors:

```
P = 1                   # number of regression coefficient
s
ss = 3 * sqrt(2)        # scale parameter in the normal pr
ior of sigma
st = 3 * sqrt(0.1)      # scale parameter in the normal pr
ior of tau
ap = 3/1; bp = 3/0.1    # upper and lower bound of phi
```

3.1 Response NNGP models in Stan

The following chunk is the R code for running response NNGP models. It's worth mentioning that we use the response and design matrix sorted by the order from `NNMatrix` instead of the raw `Y` and `X` in the data block.

```
library(rstan)
options(mc.cores = parallel::detectCores())
data <- list(N = N, M = M, P = P,
             Y = Y[NN.matrix$ord], X = X[NN.matrix$ord, ]
,    # sorted Y and X
             NN_ind = NN.matrix$NN_ind,
             NN_dist = NN.matrix$NN_dist,
             NN_distM = NN.matrix$NN_distM,
             uB = rep(0, P + 1), VB = diag(P + 1)*1000,
             ss = ss, st = st, ap = ap, bp = bp)

myinits <-list(list(beta = c(1, 5), sigma = 1, tau = 0.4,
phi = 20),
               list(beta = c(5, 5), sigma = 1.5, tau = 0.
2, phi = 5),
               list(beta = c(0, 0), sigma = 2.5, tau = 0.
1, phi = 9))

parameters <- c("beta", "sigmasq", "tausq", "phi")
samples <- stan(
  file = "nngp_response.stan",
  data = data,
  init = myinits,
  pars = parameters,
  iter = 400,
  chains = 3,
  thin = 1,
  seed = 123
)
```

The full Stan program for response NNGP model is in the file "nngp_response.stan".

```
writeLines(readLines('nngp_response.stan'))
```

```
  /* Response NNGP model */

  functions{
      real nngp_lpdf(vector Y, vector X_beta, real sigmas
q, real tausq,
                     real phi, matrix NN_dist, matrix NN_
distM, int[,] NN_ind,
                     int N, int M){

          vector[N] V;
          vector[N] YXb;
```

```
          vector[N] U;
          int dim;
          int h;
          real kappa_p_1;
          real out;
          kappa_p_1 = tausq / sigmasq + 1;
          YXb = Y - X_beta;
          U = YXb;

          for (i in 2:N) {
              matrix[ i < (M + 1) ? (i - 1) : M, i < (M +
1) ? (i - 1): M]
              iNNdistM;
              matrix[ i < (M + 1) ? (i - 1) : M, i < (M +
1) ? (i - 1): M]
              iNNCholL;
              vector[ i < (M + 1) ? (i - 1) : M] iNNcorr;
              vector[ i < (M + 1) ? (i - 1) : M] v;
              row_vector[i < (M + 1) ? (i - 1) : M] v2;
              dim = (i < (M + 1))? (i - 1) : M;

              if(dim == 1){iNNdistM[1, 1] = kappa_p_1;}
              else{
                  h = 0;
                  for (j in 1:(dim - 1)){
                      for (k in (j + 1):dim){
                          h = h + 1;
                          iNNdistM[j, k] = exp(- phi * NN
_distM[(i - 1), h]);
                          iNNdistM[k, j] = iNNdistM[j, k]
;
                      }
                  }
                  for(j in 1:dim){
                      iNNdistM[j, j] = kappa_p_1;
                  }
              }

              iNNCholL = cholesky_decompose(iNNdistM);
              for (j in 1: dim){
                  iNNcorr[j] = exp(- phi * NN_dist[(i - 1
), j]);
              }

             v = mdivide_left_tri_low(iNNCholL, iNNcorr);

             V[i] = kappa_p_1 - dot_self(v);
```

```
            v2 = mdivide_right_tri_low(v', iNNCholL);

            for (j in 1:dim){
                 U[i] = U[i] - v2[j] * YXb[NN_ind[(i - 1
), j]];
             }
          }
          V[1] = kappa_p_1;
          out = - 0.5 * ( 1 / sigmasq * dot_product(U, (U
./ V)) +
                          sum(log(V)) + N * log(sigmasq))
;
          return out;
      }
  }

  data {
      int<lower=1> N;
      int<lower=1> M;
      int<lower=1> P;
      vector[N] Y;
      matrix[N, P + 1] X;
      int NN_ind[N - 1, M];
      matrix[N - 1, M] NN_dist;
      matrix[N - 1, (M * (M - 1) / 2)] NN_distM;
      vector[P + 1] uB;
      matrix[P + 1, P + 1] VB;
      real ss;
      real st;
      real ap;
      real bp;
  }

  transformed data {
      cholesky_factor_cov[P + 1] L_VB;
      L_VB = cholesky_decompose(VB);
  }

  parameters{
      vector[P + 1] beta;
      real<lower = 0> sigma;
      real<lower = 0> tau;
      real<lower = ap, upper = bp> phi;
  }

  transformed parameters {
```

```
      real sigmasq = sigma^2;
      real tausq = tau^2;
  }

  model{
      beta ~ multi_normal_cholesky(uB, L_VB);
      sigma ~ normal(0, ss);
      tau ~ normal(0, st);
      Y ~ nngp(X * beta, sigmasq, tausq, phi, NN_dist, NN
_distM, NN_ind, N, M);
  }
```

3.2 Random-effects NNGP models for simulation study

The following chunk is the R code for running Random-effects NNGP models:

```
options(mc.cores = parallel::detectCores())
data <- list(N = N, M = M, P = P,
             Y = Y[NN.matrix$ord], X = X[NN.matrix$ord, ]
,    # sorted Y and X
             NN_ind = NN.matrix$NN_ind,
             NN_dist = NN.matrix$NN_dist,
             NN_distM = NN.matrix$NN_distM,
             uB = rep(0, P + 1), VB = diag(P + 1)*1000,
             ss = ss, st = st, ap = ap, bp = bp)

myinits <-list(list(beta = c(1, 5), sigma = 1, tau = 0.5,
phi = 20,
                    w_b1 = rep(0, N)),
               list(beta = c(5, 5), sigma = 1.5, tau = 0.
2, phi = 5,
                    w_b1 = rep(0.1, N)),
               list(beta = c(0, 0), sigma = 2.5, tau = 0.
1, phi = 9 ,
                    w_b1 = rep(0, N)))

parameters <- c("beta", "sigmasq", "tausq", "phi", "w")
samples_w <- stan(
  file = "nngp_random.stan",
  data = data,
  init = myinits,
  pars = parameters,
  iter = 400,
  chains = 3,
  thin = 1,
  seed = 123
)
```

The full Stan program for random-effects NNGP model is in the file "nngp_random.stan".

```
writeLines(readLines('nngp_random_b1.stan'))
```

```
  /* Random-effects NNGP model with spatial random effect
centered at intercept */

  functions{
      real nngp_w_lpdf(vector w_b1, real sigmasq, real ph
i, matrix NN_dist,
                       matrix NN_distM,int[,] NN_ind, int
N, int M,
                       real intercept){
```

```
          vector[N] V;
          vector[N] I_Aw;
          vector[N] w;
          int dim;
          int h;
          real out;
          w = w_b1 - intercept;
          I_Aw = w;

          for (i in 2:N) {
              matrix[ i < (M + 1)? (i - 1) : M, i < (M +
1)? (i - 1): M]
              iNNdistM;
              matrix[ i < (M + 1)? (i - 1) : M, i < (M +
1)? (i - 1): M]
              iNNCholL;
              vector[ i < (M + 1)? (i - 1) : M] iNNcorr;
              vector[ i < (M + 1)? (i - 1) : M] v;
              row_vector[i < (M + 1)? (i - 1) : M] v2;

              dim = (i < (M + 1))? (i - 1) : M;

              // get exp(-phi * NN_distM)

              if(dim == 1){iNNdistM[1, 1] = 1;}
              else{
                  h = 0;
                  for (j in 1:(dim - 1)){
                      for (k in (j + 1):dim){
                          h = h + 1;
                          iNNdistM[j, k] = exp(- phi * NN
_distM[(i - 1), h]);
                          iNNdistM[k, j] = iNNdistM[j, k]
;
                      }
                  }
                  for(j in 1:dim){
                      iNNdistM[j, j] = 1;
                  }
              }

              iNNCholL = cholesky_decompose(iNNdistM);
              for (j in 1: dim){
                  iNNcorr[j] = exp(- phi * NN_dist[(i - 1
), j]);
              }
```

```
              //vector[dim] v;
              v = mdivide_left_tri_low(iNNCholL, iNNcorr)
;

              V[i] = 1 - dot_self(v);

              v2 = mdivide_right_tri_low(v', iNNCholL);

              for (j in 1:dim){
                  I_Aw[i] = I_Aw[i] - v2[j] * w[NN_ind[(i
- 1), j]];
              }
          }
          V[1] = 1;
          out = - 0.5 * ( 1 / sigmasq * dot_product(I_Aw,
(I_Aw ./ V)) +
                          sum(log(V)) + N * log(sigmasq))
;
          return out;
      }
  }


  data {
      int<lower=1> N;
      int<lower=1> M;
      int<lower=1> P;
      vector[N] Y;
      matrix[N, P + 1] X;
      int NN_ind[N - 1, M];
      matrix[N - 1, M] NN_dist;
      matrix[N - 1, (M * (M - 1) / 2)] NN_distM;
      vector[P + 1] uB;
      matrix[P + 1, P + 1] VB;
      real ss;
      real st;
      real ap;
      real bp;
  }

  transformed data {
      cholesky_factor_cov[P + 1] L_VB;
      L_VB = cholesky_decompose(VB);
  }

  parameters{
```

```
      vector[P + 1] beta;
      real<lower = 0> sigma;
      real<lower = 0> tau;
      real<lower = ap, upper = bp> phi;
      vector[N] w_b1;
  }

  transformed parameters {
      real sigmasq = sigma^2;
      real tausq = tau^2;
  }

  model{
      beta ~ multi_normal_cholesky(uB, L_VB);
      sigma ~ normal(0, ss);
      tau ~ normal(0, st);
      w_b1 ~ nngp_w(sigmasq, phi, NN_dist, NN_distM, NN_i
nd, N, M, beta[1]);
      Y ~ normal(block(X, 1, 2, N, P) * tail(beta, P) + w
_b1, tau);
  }
```

4 Results and Discussion

In this section, we will show the results of the simulation study, compare response and random-effects NNGP models, and provide suggestions on how to use NNGP based models.

4.1 Response NNGP model for simulation study

Response NNGP model is faster and easier to sample from posterior distribution than the random-effects NNGP models. The following shows the summary table and trace plot of the posterior samples from response NNGP model.

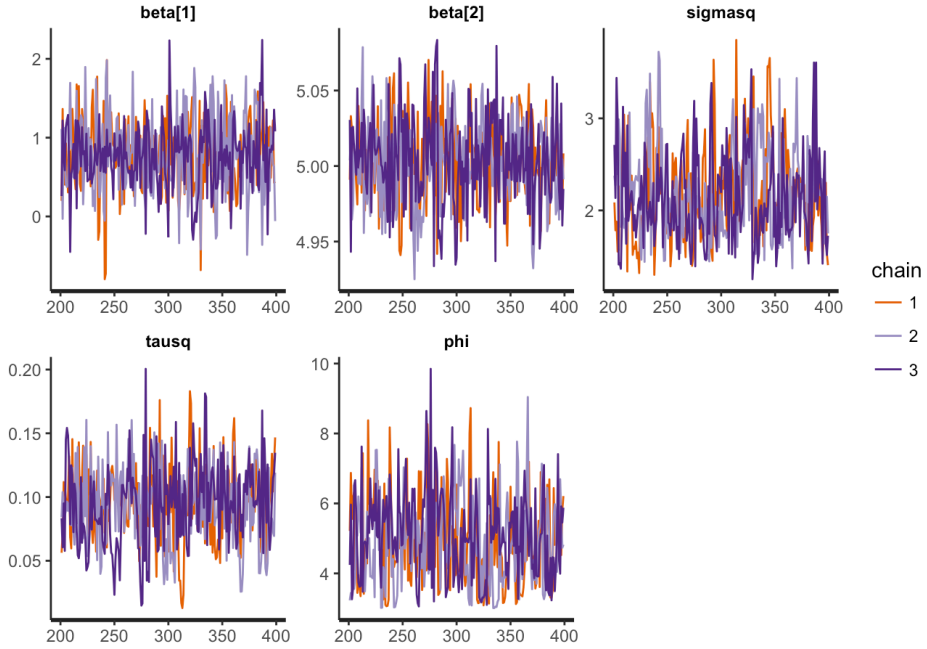
```
print(samples)
```

```
Inference for Stan model: nngp_response.
3 chains, each with iter=400; warmup=200; thin=1;
post-warmup draws per chain=200, total post-warmup draws=
600.

          mean se_mean   sd    2.5%     25%     50%    7
5%  97.5% n_eff
beta[1]    0.78    0.02 0.46   -0.19    0.46    0.79   1.
08   1.69   557
beta[2]    5.00    0.00 0.03    4.95    4.98    5.00   5.
02   5.06   507
sigmasq    2.19    0.03 0.50    1.43    1.79    2.13   2.
50   3.39   279
tausq      0.09    0.00 0.03    0.03    0.07    0.10   0.
12   0.15   284
phi        4.97    0.07 1.24    3.10    3.96    4.88   5.
79   7.59   307
lp__    -101.01    0.15 1.81 -105.44 -101.84 -100.62 -99.
73 -98.62   147
        Rhat
beta[1] 1.00
beta[2] 1.00
sigmasq 1.01
tausq   1.01
phi     1.01
lp__    1.00

Samples were drawn using NUTS(diag_e) at Thu Nov 16 18:54
:19 2017.
For each parameter, n_eff is a crude measure of effective
sample size,
and Rhat is the potential scale reduction factor on split
chains (at
convergence, Rhat=1).
```

```
stan_trace(samples)
```



4.2 Random-effects NNGP models for simulation study

The following gives the summary table posterior samples and trace plots of the MCMC Chains from random-effects NNGP model:

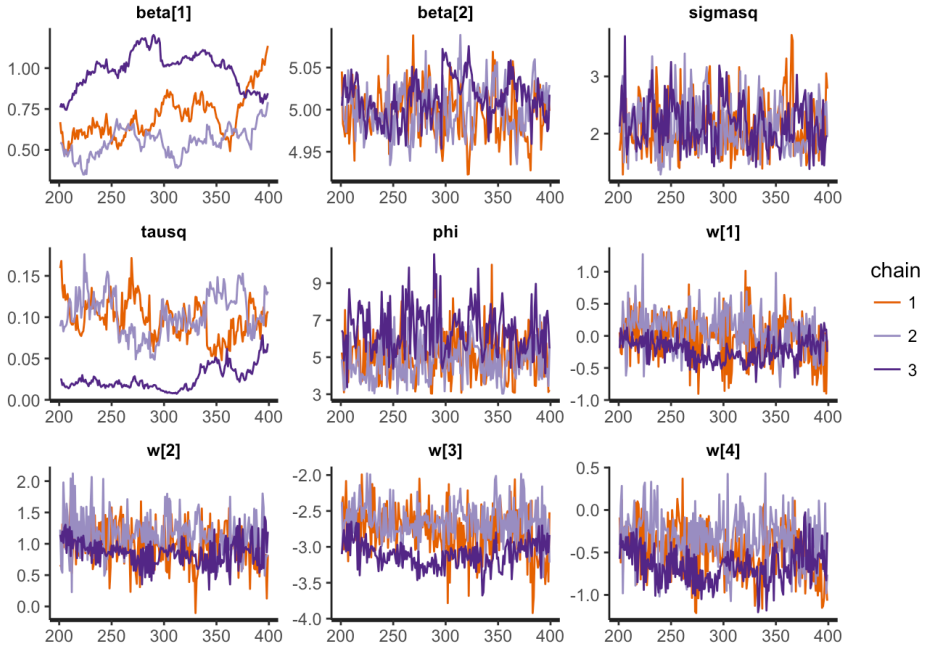
```
print(samples_w, pars = c("beta", "sigmasq", "tausq", "ph
i", "w[1]", "w[2]",
                          "w[3]", "w[4]"))
```

```
Inference for Stan model: nngp_random.
3 chains, each with iter=400; warmup=200; thin=1;
post-warmup draws per chain=200, total post-warmup draws=
600.

         mean se_mean   sd  2.5%   25%   50%   75% 97.5%
n_eff Rhat
beta[1]  0.75    0.15 0.21  0.42  0.58  0.70  0.96  1.15
2 2.25
beta[2]  5.00    0.00 0.03  4.95  4.98  5.00  5.02  5.06
40 1.06
sigmasq  2.13    0.03 0.44  1.44  1.81  2.06  2.40  3.11
266 1.00
tausq    0.08    0.03 0.04  0.01  0.03  0.08  0.11  0.15
2 2.20
phi      5.47    0.65 1.38  3.19  4.46  5.38  6.34  8.46
4 1.18
w[1]    -0.05    0.11 0.32 -0.63 -0.27 -0.07  0.15  0.61
8 1.13
w[2]     0.99    0.09 0.32  0.43  0.78  0.97  1.19  1.68
13 1.08
w[3]    -2.84    0.18 0.34 -3.43 -3.09 -2.85 -2.60 -2.18
4 1.30
w[4]    -0.49    0.12 0.31 -1.04 -0.72 -0.51 -0.26  0.11
7 1.14

Samples were drawn using NUTS(diag_e) at Thu Nov 16 19:01
:11 2017.
For each parameter, n_eff is a crude measure of effective
sample size,
and Rhat is the potential scale reduction factor on split
chains (at
convergence, Rhat=1).
```

```
stan_trace(samples_w, pars = c("beta", "sigmasq", "tausq"
, "phi", "w[1]",
                               "w[2]","w[3]", "w[4]"))
```



It is not surprising to see a slower convergence rate and worse mixing of the MCMC Chains. Response NNGP model marginalizes out the spatial effects w, yields a lower-dimensional parameter space, hence drastically improves the posterior geometry. While the parameters to be estimated in a random-effects NNGP model $\{\theta, w\}$ are highly correlated, and the number of parameters is on the scale of the number of observations. Thus the convergence rate of MCMC chains from a random-effects NNGP is slow because of the high correlation and dimension of the parameter space.

Notice the trace plot of random-effects w are highly correlated with the intercept, we modified the code and make the random-effects $w(s)$ centered at the intercept. The code of modified random-effects NNGP model can be found in "nngp.R" and "nngp_random_b1.stan". Here we suppress the details of the code and show the results directly:

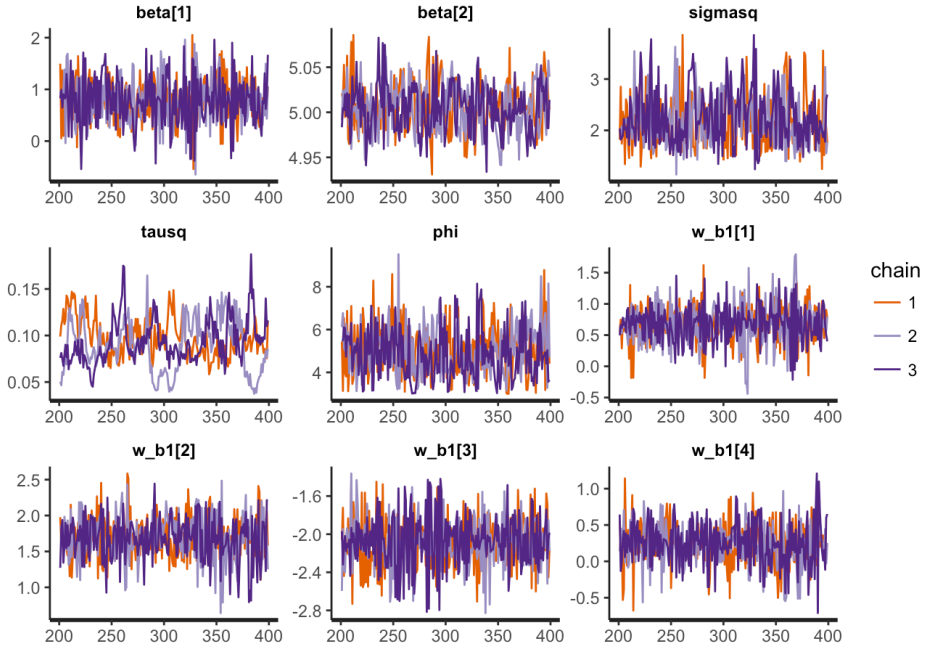
```
print(samples_wb1, pars = c("beta", "sigmasq", "tausq", "
phi", "w_b1[1]",
                            "w_b1[2]", "w_b1[3]", "w_b1[4
]"))
```

```
Inference for Stan model: nngp_random_b1.
3 chains, each with iter=400; warmup=200; thin=1;
post-warmup draws per chain=200, total post-warmup draws=
600.

         mean se_mean   sd  2.5%   25%   50%   75% 97.5%
n_eff Rhat
beta[1]  0.78    0.02 0.44 -0.05  0.49  0.79  1.07  1.65
600 1.00
beta[2]  5.01    0.00 0.03  4.96  4.99  5.00  5.02  5.06
354 1.00
sigmasq  2.20    0.03 0.50  1.45  1.82  2.12  2.52  3.39
381 1.00
tausq    0.09    0.00 0.02  0.05  0.08  0.09  0.11  0.14
47 1.02
phi      4.95    0.06 1.17  3.12  4.04  4.83  5.66  7.34
328 1.00
w_b1[1]  0.67    0.01 0.31  0.04  0.49  0.68  0.87  1.27
600 1.00
w_b1[2]  1.69    0.01 0.30  1.08  1.49  1.68  1.90  2.22
600 1.00
w_b1[3] -2.08    0.01 0.27 -2.63 -2.25 -2.07 -1.89 -1.52
600 1.00
w_b1[4]  0.24    0.01 0.29 -0.43  0.07  0.24  0.43  0.79
600 1.00

Samples were drawn using NUTS(diag_e) at Thu Nov 16 19:03
:56 2017.
For each parameter, n_eff is a crude measure of effective
sample size,
and Rhat is the potential scale reduction factor on split
chains (at
convergence, Rhat=1).
```

```
stan_trace(samples_wb1, pars = c("beta", "sigmasq", "taus
q", "phi", "w_b1[1]",
                                 "w_b1[2]", "w_b1[3]", "w
_b1[4]"))
```



4.3 Discussion

We recommend a response NNGP model for a large scale data analysis when the study focuses on the inference of parameter set θ. On the other hand, random-effects NNGP models are preferred when the study needs the recovery of latent process $w(s)$. However, the convergence rate of the MCMC Chains from random-effects model could be prohibitively slow when the sample size is large, so we only recommend coding random-effects NNGP model in Stan when the sample size is small. For recovering latent process when the sample size is large, Package spNNGP provides an algorithm for random-effects NNGP model, which implements a "sequential" Gibbs sampler for updating the latent process. Conjugate NNGP models are also good options for recovering latent process $w(s)$. More details of NNGP based models can be found in Finley et al. (2017)

References

Datta, Abhirup, Sudipto Banerjee, Andrew O Finley, and Alan E Gelfand. 2016. "Hierarchical Nearest-Neighbor Gaussian Process Models for Large Geostatistical Datasets" 111. Taylor & Francis: 800–812.

Finley, Andrew O, Abhirup Datta, Bruce C Cook, Douglas C Morton, Hans E Andersen, and Sudipto Banerjee. 2017. "Applying Nearest Neighbor Gaussian Processes to Massive Spatial Data Sets Forest Canopy Height Prediction Across Tanana Valley Alaska."

Vecchia, A. V. 1988. "Estimation and Model Identification for Continuous Spatial Processes."