Spatio Tamporal Data Analysis

Hyeonki Seo

Due: Oct 11th 2021

1

1.1 Visualize profile loglikelihood function of ρ for $[0.005, 0.5]$ interval

In this problem, X is generaged by sampling from uniform distribution whose supports are 0 to 1, beta is one and sigma2 is 0.1. Under this setting, profile loglikelihood is calculated for each $\rho \in [0.0005, 0.5]$. I generate 100 ρs with equal differences. Maximum log likelihood value is 293.7087 as $\rho = 0.105$.Code and Figure are below.

```
# pll simulation
rhos = seq(0.005, 0.5, length.out=100)
pll_rhos = rep(0, 100)
for (i in 1:100){
  pll_rhos[i] = pll(rhos[i], d, Y, X)
}
```

Listing 1: codes for visualizing profile loglikelihood

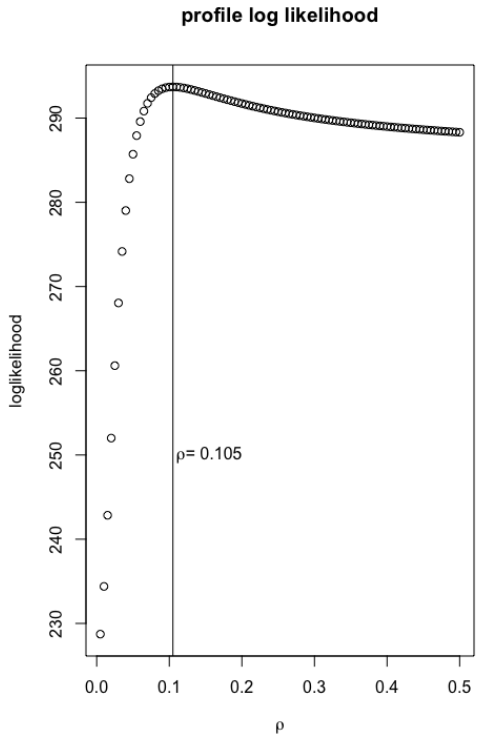

Figure 1: profile loglikelihood with different ρ

1.2 Obtain MLE for ρ, β, σ^2

With $\hat{\rho}^{mle}$, We can get $\hat{\beta}^{mle}, \hat{\sigma^2}^{mle}$ by plugging in mle of rho in formulas. So, the only thing need to optimize is ρ. and this can be done by R code(optimize). formula, codes and results are below.

$$\hat{\beta} = (X^T K(\hat{\rho}) X)^{-1} X^T K(\hat{\rho})^{-1} Y$$

$$\hat{\sigma} = \frac{1}{n}(Y - X\hat{\beta})^T K(\hat{\rho})^{-1}(Y - X\hat{\beta})$$

```
# optimization and printing results
rho_mle <- optimize(pll, c(0.005,0.5), d=d, Y=Y, Xmat=X, maximum=TRUE)$maximum
pll(rho_mle, d, Y, X, return=TRUE)

-----------------------------------------------------------------
$rho
[1] 0.1051027

$beta
             [,1]
[1,] -0.01270188
[2,]  0.20559177
[3,] -0.07928086

$sigma2
          [,1]
[1,] 0.109068

$p
          [,1]
[1,] 293.7087
attr(,"logarithm")
[1] TRUE
```

Listing 2: codes for obtaining mle

$$\hat{\rho}_{mle} = 0.1051027$$

$$\hat{\beta}_{mle} = \begin{bmatrix} \hat{\beta}_{0mle} \\ \hat{\beta}_{1mle} \\ \hat{\beta}_{2mle} \end{bmatrix} = \begin{bmatrix} -0.0127 \\ 0.2056 \\ -0.0792 \end{bmatrix}$$

$$\hat{\sigma^2}_{mle} = 0.109068$$

2

Here is some code description. Number of Train data is 80% of N, Number Test data is rest of it. From data model, probability is generated by logit function because response variable Y follows binomial distribution which has form of

$$p = \frac{exp(X\beta + W)}{1 + exp(x\beta + W)}, \quad where\ W\ is\ latent\ process$$

From the Process model, Latent process W follows normal distribution with mean 0 and covariance $\sigma^2 K(\phi)$ where K is matern covariance function.(which is changed from given nimble code) From the parameter model, prior distributions of each parameters are

$$\sigma^2 \sim InverseGamma(0.2, 0.2)$$

$$\phi \sim U(0, 1)$$

$$\beta_i \sim N(0.10^2),\ i = 1, 2$$

Number of iteration is 200,000 and get rid of first 10000 samples(burn-in) and when make prediction, I use threshold of 0.5.

2.1 Posterior mean, Highest posterior density, Acceptance rate

Table 1 shows result of Posterior mean, Lower and Upper bound of 95% credible interval, Acceptance rate of each parameters. In most of cases, True β is contained in HPD interval except for β_2 in $N = 400$ case. However,In all cases, true ϕ,^{2} value is not contained. Acceptance rate of each parameters are around 44%

		Posterior mean	95% Lower	95% Upper	Acceptance rate
N = 200	β_1	0.493	-0.122	1.122	0.440
	β_2	0.813	0.200	1.431	0.439
	ϕ	0.049	0.001	0.085	0.441
	σ^2	0.238	0.065	0.837	0.421
N = 400	β_1	0.913	0.494	1.329	0.441
	β_2	0.562	0.157	0.965	0.440
	ϕ	0.061	0.010	0.098	0.445
	σ^2	0.035	0.021	0.051	0.434
N = 800	β_1	1.046	0.759	1.345	0.440
	β_2	0.865	0.574	1.169	0.440
	ϕ	0.002	0.000	0.008	0.451
	σ^2	0.061	0.019	0.189	0.412
N = 1600	β_1	0.857	0.646	1.058	0.440
	β_2	0.999	0.785	1.216	0.440
	ϕ	0.006	0.003	0.008	0.439
	σ^2	0.184	0.131	0.337	0.431

Table 1: Posterior mean, HPD and Acceptance rate for each N

2.2 Computing time & Prediction accuracy

Computing time and Prediction accuracy are shown in Table 2. When the number of sampled doubles, the calculation time is increasing from 4.6 to 9.61 times. By this, It can be seen that the Computational complexity is $O(N^3)$.

	Computing time	Prediction accuracy
$N = 200$	433.124 sec	0.7
$N = 400$	1999.061 sec	0.65
$N = 800$	19223.61 sec	0.68125
$N = 1600$	130110.947 sec	0.678125

Table 2: Computing time and Prediction accuracy for each N

2.3 Traceplot

Beta becomes converge around True value in all cases. Sigma, on the other hand, becomes converge to zero which is not close to the true value. ϕ does not converge in all case. Which is because we don't have any pre-knowledge of ϕ that we take prior distribution of Uniform unlike σ^2 which follows inverse gamma distribution, and β which follows normal distribution.

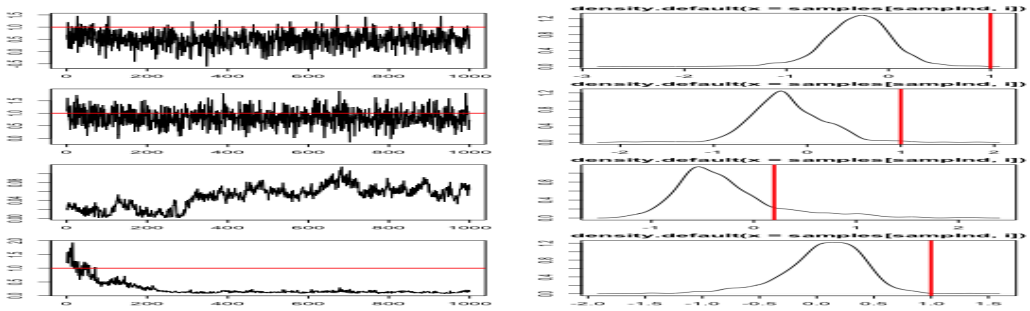

Figure 2: N=200

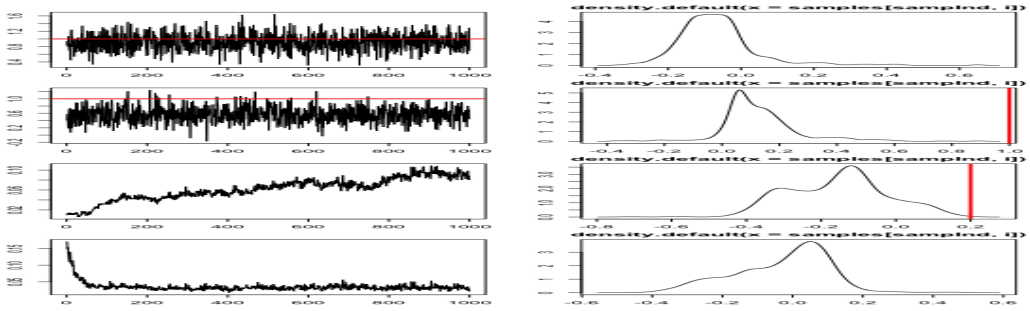

Figure 3: N=400

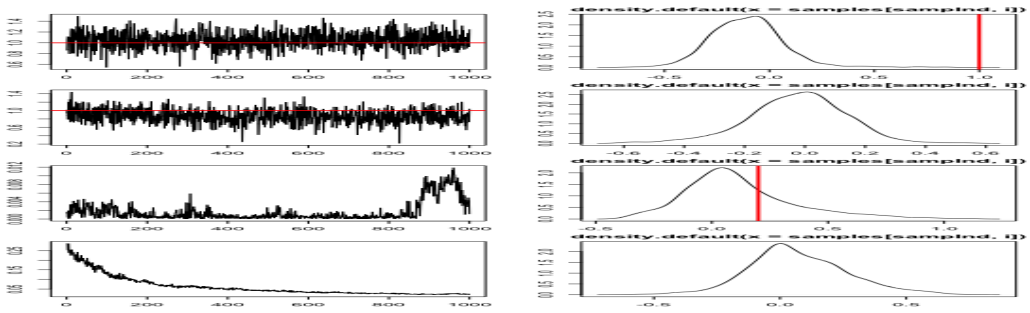

Figure 4: N=800

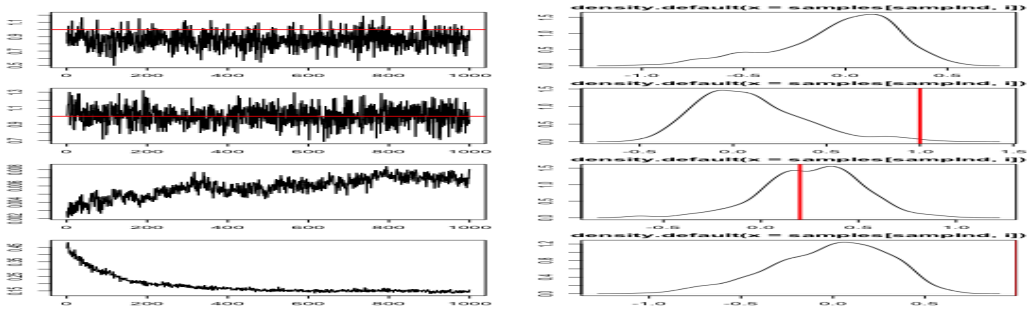

Figure 5: N=1600

3 R code

```
########################
######question 1#######
########################

library(mvtnorm)
library(fields)
library(sp)

source("helper.fns.R")

## Simulate data
set.seed(3)
n <- 200
x <- matrix(runif(n*2, min=0, max=1), ncol = 2) # locations
X <- cbind(1, x) # trend surface model ( design matrix )
head(X)
beta <- c(0, 1, 1); sigma2 <- 0.1; rho <- 0.1 # true parameters
d <- rdist(x)
Sigma <- sigma2 * exp(-d/rho) # Exponential covariance model
Y <- drop(rmvnorm(1, mean = X %*% beta, Sigma = Sigma)) # sample from multivatiate normal with true p
head(Y)

plot.point.ref(x, Y)

determinant(Sigma, logarithm = TRUE)$modulus # log value of determinant

## Profile log-likelihood, ignoring a constant
## 즉 정해진 rho 에서 베타 햇, 시그마제곱 햇을 만드는 것.
##
```

```
pll <- function(rho, d, Y, Xmat, verbose = FALSE, return = FALSE){
  n <- length(Y)
  K <- exp(-d/rho)
  beta.hat <- solve(t(Xmat) %*% solve(K, Xmat)) %*% t(Xmat) %*% solve(K, Y)
  resid <- drop(Y - Xmat %*% beta.hat)
  sigma2.hat <- t(resid) %*% solve(K, resid) / n
  p <- - 0.5 * n * log(sigma2.hat) - 0.5 * determinant(K, log = TRUE)$modulus
  if (verbose) print(c(rho, pll = p))
  if (return) p <- list(rho=rho, beta=beta.hat, sigma2=sigma2.hat, p=p)
  return(p)
}  ## p is loglikelihood value


# pll simulation
rhos = seq(0.005, 0.5, length.out=100)
pll_rhos = rep(0, 100)

for (i in 1:100){
  pll_rhos[i] = pll(rhos[i], d, Y, X)
}

max_rho = rhos[which(pll_rhos == max(pll_rhos))]
head(pll_rhos)
max_rho

pll(max_rho, d, Y, X)

##plotting
par(mfrow = c(1,1), mar = c(5,5,5,5))
plot(x = rhos, y = pll_rhos, xlab = expression(rho), main = 'profile log likelihood',
     ylab = 'loglikelihood')
lines(c(max_rho,max_rho), c(0,300))
text(x = max_rho + 0.05, y = 250 , labels = expression(rho*'= 0.105'))


# ml estimation
?optimize
?optim
rho_mle <- optimize(pll, c(0.005,0.5), d=d, Y=Y, Xmat=X, maximum=TRUE)$maximum
pll(rho_mle, d, Y, X, return=TRUE)
# rho is 0.105102, beta =  ..., sigma2 = 0.109068, ll = 293.7087


############################
########question 2 #######
############################

source("http://www.stat.psu.edu/~mharan/batchmeans.R")
library(nimble);library(mvtnorm);library(fields)
## Using Ming-Hui Chen's paper in Journal of Computational and Graphical Stats.
hpd <- function(samp,p=0.05){
  ## to find an approximate (1-p)*100% HPD interval from a
  ## given posterior sample vector samp

  r <- length(samp)
  samp <- sort(samp)
```

```
  rang <- matrix(0,nrow=trunc(p*r),ncol=3)
  dimnames(rang) <- list(NULL,c("low","high","range"))
  for (i in 1:trunc(p*r)) {
    rang[i,1] <- samp[i]
    rang[i,2] <- samp[i+(1-p)*r]
    rang[i,3] <- rang[i,2]-rang[i,1]
  }
  hpd <- rang[order(rang[,3])[1],1:2]
  return(hpd)
}

# Exponential Covariance Function
expCov<-function(distMat,phi){
  exp(-distMat/phi)
}

sqeCov<-function(distMat,phi){
  exp(-0.5*(distMat/phi)^2)
}

matCov<-function(distMat,phi){
  (1+(sqrt(5)*(distMat/phi))+((5*distMat^2)/(3*(phi^2))))*exp(-(sqrt(5)*(distMat/phi)))
}


# Matern Cov Function + Acceptance Rate function
Matern <- function(d, param = c(scale = 1, range = 1, smoothness = 2)) {
  scale <- param[1]
  range <- param[2]
  smoothness <- param[3]
  if (any(d < 0))
    stop("distance argument must be nonnegative")
  d <- d / range
  d[d == 0] <- 1e-10
  rootcon<-sqrt(2*smoothness)
  con <- (2^(smoothness - 1)) * gamma(smoothness)
  con <- 1 / con
  return(scale * con * ((rootcon*d)^smoothness) * besselK(rootcon*d, smoothness))
}
accRateFunc<-function(x){
  accRate<-(length(unique(x))-1)/(length(x)-1)
  return(accRate)
}


# Summary
summaryFunction<-function(mcmcDat,bmseThresh=0.01,time){

  # Parameters
  summaryMat<-rbind(apply(mcmcDat,2,mean),
                    apply(mcmcDat,2,hpd),
                    apply(mcmcDat,2,accRateFunc),
                    bmmat(mcmcDat)[,2],
                    abs(apply(mcmcDat,2,mean))*bmseThresh,
                    apply(mcmcDat,2,ess),
                    apply(mcmcDat,2,ess)/time)
```

```
  rownames(summaryMat)<-c("Mean","95%CI-Low","95%CI-High",
                          "Accept","BMSE",paste(bmseThresh,"x mean"),
                          "ESS","ESS/sec")
  return(summaryMat)
}

# NIMBLE FUNCTIONS -> 즉 위와 똑같은 코드를 님블 펑션으로 짜는 것

expcov <- nimbleFunction(
  run = function(dists = double(2), phi = double(0)) {
    returnType(double(2))
    n <- dim(dists)[1]
    result <- matrix(nrow = n, ncol = n, init = FALSE)
    for(i in 1:n){
      for(j in 1:n){
        result[i, j] <- exp(-dists[i,j]/phi)
      }
    }

    return(result)
  })


matcov <- nimbleFunction(
  run = function(dists = double(2), phi = double(0)) {
    returnType(double(2))
    n <- dim(dists)[1]
    result <- matrix(nrow = n, ncol = n, init = FALSE)
    for(i in 1:n){
      for(j in 1:n){
        result[i, j] <- (1+(sqrt(5)*(dists[i,j]/phi))+((5*dists[i,j]^2)/(3*(phi^2))))*exp(-(sqrt(5)*(
      }
    }

    return(result)
  })




######### 2. Simulate Dataset ###############


############################
######## N = 200 case#####
############################

#Parameters
set.seed(55555)
n=160 ; ncv=40 #  n = training data, ncv = test data set
beta=c(1,1) ; phi=0.2 ; sigma2=1

# Generate Locations
## Split into Model + Cross-Validation
```

```
gridLocation<-cbind(runif(n,min = 0,max = 1),runif(n,min = 0,max = 1))
CVgridLocation<-cbind(runif(ncv,min = 0,max = 1),runif(ncv,min = 0,max = 1))
comboLocation<-rbind(gridLocation,CVgridLocation)
distMatFull<-as.matrix(rdist(comboLocation))
# Create Indices
modInd<-1:n
CVInd<-(n+1):nrow(distMatFull)
distMatMod<-distMatFull[modInd,modInd]
# Covariates
XMat<-cbind(runif(n,-1,1),runif(n,-1,1))
XMatCV<-cbind(runif(ncv,-1,1),runif(ncv,-1,1))
XB<-XMat%*%beta
cvXB<-XMatCV%*%beta
XBFull<-rbind(XB,cvXB)

# Covariance Matrix
CovMat<-sigma2*matCov(distMatFull,phi)

# Latent Gaussian Random Field
gpWFull <- as.numeric(rmvnorm(n=1,mean=rep(0,nrow(CovMat)),Sigma = CovMat,method = "chol"))
pWFullBin<-exp(gpWFull+XBFull)/(1+exp(gpWFull+XBFull))

# Observations
obsFullBin<-sapply(pWFullBin,rbinom,n=1,size=1)


###################
# Model Sample
# Latent Process
gpWMod<-gpWFull[modInd]
# Expected Value
pWModBin<-pWFullBin[modInd]
# Observations
obsModBin<-obsFullBin[modInd]

# CV Sample
# Latent Process
gpWCV<-gpWFull[CVInd]
# Expected Value
pWCVBin<-pWFullBin[CVInd]
# Observations
obsCVBin<-obsFullBin[CVInd]

# Truth CVMSPE
truthCVMSPEBin<-mean((pWCVBin-obsCVBin)^2)

#save.image("SpatialData200.RData")

######### 3. Fit binomial Regression ###############
#load("SpatialData.RData")
# Nimble Model
model_string <- nimbleCode({

  # Data Model
  for(i in 1:n){
    p[i] <- exp(W[i]+XB[i])/(1+exp(W[i]+XB[i]))
```

```
    Z[i] ~ dbern(p[i])
  }

  # Constant and Cov Matrix
  XB[1:n]<-beta1*X[,1] + beta2*X[,2]
  covMat[1:n,1:n]<- matcov(dists[1:n,1:n],phi)
  fullCovMat[1:n,1:n]<- sigma2*covMat[1:n,1:n]

  # Process Model
  W[1:n] ~ dmnorm(mean = mn[1:n], cov = fullCovMat[1:n,1:n])

  # Parameter Model
  sigma2   ~  dinvgamma(0.2, 0.2)
  phi   ~  dunif(0,1)
  beta1 ~  dnorm(0, sd=sqrt(100))
  beta2 ~  dnorm(0, sd=sqrt(100))
})

niter=200000
consts   <- list(n=n,X=XMat,dists=distMatMod,mn=rep(0,n))
data     <- list(Z=obsModBin)
inits    <- list(beta1=rnorm(1),beta2=rnorm(1),phi=0.5,sigma2=2,
                 W=rnorm(n))

# Run MCMC

pt<-proc.time()
samples  <- nimbleMCMC(model_string, data = data, inits = inits,
                       constants=consts,
                       monitors = c("beta1", "beta2","phi","sigma2","W"),
                       samplesAsCodaMCMC=TRUE,WAIC=FALSE,summary=FALSE,
                       niter = niter, nburnin = 10000, nchains = 1)
ptFinal200<-proc.time()-pt
ptFinal200



# Summary
# Table
summaryMat<-list()
summaryMat[[1]]<-round(summaryFunction(samples[,c("beta1", "beta2","phi","sigma2")],
                                       time=ptFinal200[3]),3)
summaryMat[[2]]<-round(summaryFunction(samples[,1:n],
                                       time=ptFinal200[3]),3)
summaryMat[[1]]
apply(summaryMat[[2]],1,mean)
#save(summaryMat,samples,ptFinal200,file="BinomialMCMCResults200.RData")
load('BinimialMCMCResults200.RData')

# plot
pdf(file = "BinomialMCMCResults200.pdf",width=11,height=8.5)
par(mfrow=c(4,2),mar=c(2,2,2,2))
sampInd<-floor(seq(1,nrow(samples),length.out = 1000))
param_name <- c('beta1','beta2','phi','sigma2')
for(i in 1:4){
  plot.ts(samples[sampInd,param_name[i]]); abline(h=c(beta,phi,sigma2)[i],col="red",lwd=2)
```

```
  plot(density(samples[sampInd,i])); abline(v=c(beta,phi,sigma2)[i],col="red",lwd=2)
}
dev.off()

par(mfrow=c(2,2), mar = c(3,3,3,3))
acf(samples[,'beta1'])
acf(samples[,'beta2'])
acf(samples[,'phi'])
acf(samples[,'sigma2'])
dim(samples)


# prediction
params <-  summaryMat[[1]][1,]
params
predXB <- samples[,c('beta1','beta2')][180000:190000 ,] %*% t(XMatCV)


samples[,c('beta1','beta2')][180000:190000 ,]

dim(samples)[1]

rmvnorm(1, mean = 0, Sigma = samples[i,'sigma2']*matCov(distMatFull[CVInd,CVInd],samples[i,'phi']))

# Sample W_pred

Wpredmat <- matrix(NA, nrow = 10000, ncol = ncv)
dim(Wpredmat)
for (i in 1:10000){
  Wpredmat[i,] <- rmvnorm(1, mean = 0, Sigma = samples[i+180000,'sigma2']*matCov(distMatFull[CVInd,CV
}



p_pred <- matrix(NA, nrow = 10000, ncol = ncv)
for (i in 1:10000) {
  for (j in 1:ncv) {
    p_pred[i,j] <-  exp(Wpredmat[i,j] + predXB[i,j]) / (1+exp(Wpredmat[i,j] + predXB[i,j]))
  }
}

z_pred <- as.integer(apply(p_pred,2,mean) >= 0.5)
acc_200 = sum(z_pred == obsFullBin[CVInd]) / ncv
acc_200
ptFinal200

#########################
####### N = 400 case####
#########################


#Parameters
set.seed(55555)
n=320 ; ncv=80 #  n = training data, ncv = test data set
beta=c(1,1) ; phi=0.2 ; sigma2=1
```

```
# Generate Locations
## Split into Model + Cross-Validation
gridLocation<-cbind(runif(n,min = 0,max = 1),runif(n,min = 0,max = 1))
CVgridLocation<-cbind(runif(ncv,min = 0,max = 1),runif(ncv,min = 0,max = 1))
comboLocation<-rbind(gridLocation,CVgridLocation)
distMatFull<-as.matrix(rdist(comboLocation))
# Create Indices
modInd<-1:n
CVInd<-(n+1):nrow(distMatFull)
distMatMod<-distMatFull[modInd,modInd]
# Covariates
XMat<-cbind(runif(n,-1,1),runif(n,-1,1))
XMatCV<-cbind(runif(ncv,-1,1),runif(ncv,-1,1))
XB<-XMat%*%beta
cvXB<-XMatCV%*%beta
XBFull<-rbind(XB,cvXB)

# Covariance Matrix
CovMat<-sigma2*matCov(distMatFull,phi)

# Latent Gaussian Random Field
gpWFull <- as.numeric(rmvnorm(n=1,mean=rep(0,nrow(CovMat)),Sigma = CovMat,method = "chol"))
pWFullBin<-exp(gpWFull+XBFull)/(1+exp(gpWFull+XBFull))

# Observations
obsFullBin<-sapply(pWFullBin,rbinom,n=1,size=1)


###################
# Model Sample
# Latent Process
gpWMod<-gpWFull[modInd]
# Expected Value
pWModBin<-pWFullBin[modInd]
# Observations
obsModBin<-obsFullBin[modInd]

# CV Sample
# Latent Process
gpWCV<-gpWFull[CVInd]
# Expected Value
pWCVBin<-pWFullBin[CVInd]
# Observations
obsCVBin<-obsFullBin[CVInd]

# Truth CVMSPE
truthCVMSPEBin<-mean((pWCVBin-obsCVBin)^2)

#save.image("SpatialData400.RData")

######### 3. Fit binomial Regression ###############
#load("SpatialData.RData")
# Nimble Model
model_string <- nimbleCode({

  # Data Model
```

```
  for(i in 1:n){
    p[i] <- exp(W[i]+XB[i])/(1+exp(W[i]+XB[i]))
    Z[i] ~ dbern(p[i])
  }

  # Constant and Cov Matrix
  XB[1:n]<-beta1*X[,1] + beta2*X[,2]
  covMat[1:n,1:n]<- matcov(dists[1:n,1:n],phi)
  fullCovMat[1:n,1:n]<- sigma2*covMat[1:n,1:n]

  # Process Model
  W[1:n] ~ dmnorm(mean = mn[1:n], cov = fullCovMat[1:n,1:n])

  # Parameter Model
  sigma2   ~  dinvgamma(0.2, 0.2)
  phi   ~  dunif(0,1)
  beta1 ~  dnorm(0, sd=sqrt(100))
  beta2 ~  dnorm(0, sd=sqrt(100))
})

niter=200000
consts   <- list(n=n,X=XMat,dists=distMatMod,mn=rep(0,n))
data     <- list(Z=obsModBin)
inits    <- list(beta1=rnorm(1),beta2=rnorm(1),phi=0.5,sigma2=2,
                 W=rnorm(n))

# Run MCMC

pt<-proc.time()
samples  <- nimbleMCMC(model_string, data = data, inits = inits,
                       constants=consts,
                       monitors = c("beta1", "beta2","phi","sigma2","W"),
                       samplesAsCodaMCMC=TRUE,WAIC=FALSE,summary=FALSE,
                       niter = niter, nburnin = 10000, nchains = 1)
ptFinal400<-proc.time()-pt
ptFinal400



# Summary
# Table
summaryMat<-list()
summaryMat[[1]]<-round(summaryFunction(samples[,c("beta1", "beta2","phi","sigma2")],
                                       time=ptFinal400[3]),3)
summaryMat[[2]]<-round(summaryFunction(samples[,1:n],
                                       time=ptFinal400[3]),3)
summaryMat[[1]]
apply(summaryMat[[2]],1,mean)
save(summaryMat,samples,ptFinal400,file="BinomialMCMCResults400.RData")
#load('BinimialMCMCResults400.RData')

# plot
pdf(file = "BinomialMCMCResults400.pdf",width=11,height=8.5)
par(mfrow=c(4,2),mar=c(2,2,2,2))
sampInd<-floor(seq(1,nrow(samples),length.out = 1000))
param_name <- c('beta1','beta2','phi','sigma2')
```

```
for(i in 1:4){
  plot.ts(samples[sampInd,param_name[i]]); abline(h=c(beta,phi,sigma2)[i],col="red",lwd=2)
  plot(density(samples[sampInd,i])); abline(v=c(beta,phi,sigma2)[i],col="red",lwd=2)
}
dev.off()

par(mfrow=c(2,2), mar = c(3,3,3,3))
acf(samples[,'beta1'])
acf(samples[,'beta2'])
acf(samples[,'phi'])
acf(samples[,'sigma2'])
dim(samples)


# prediction
params <-  summaryMat[[1]][1,]
params
predXB <- samples[,c('beta1','beta2')][180000:190000 ,] %*% t(XMatCV)


samples[,c('beta1','beta2')][180000:190000 ,]

dim(samples)[1]

rmvnorm(1, mean = 0, Sigma = samples[i,'sigma2']*matCov(distMatFull[CVInd,CVInd],samples[i,'phi']))

# Sample W_pred

Wpredmat <- matrix(NA, nrow = 10000, ncol = ncv)
dim(Wpredmat)
for (i in 1:10000){
  Wpredmat[i,] <- rmvnorm(1, mean = 0, Sigma = samples[i+180000,'sigma2']*matCov(distMatFull[CVInd,CV
}



p_pred <- matrix(NA, nrow = 10000, ncol = ncv)
for (i in 1:10000) {
  for (j in 1:ncv) {
    p_pred[i,j] <-  exp(Wpredmat[i,j] + predXB[i,j]) / (1+exp(Wpredmat[i,j] + predXB[i,j]))
  }
}

z_pred <- as.integer(apply(p_pred,2,mean) >= 0.5)
acc_400 = sum(z_pred == obsFullBin[CVInd]) / ncv
acc_400
ptFinal400


############################
######### N = 800 case#####
############################


#Parameters
set.seed(55555)
```

```
n=640 ; ncv=160 #  n = training data, ncv = test data set
beta=c(1,1) ; phi=0.2 ; sigma2=1

# Generate Locations
## Split into Model + Cross-Validation
gridLocation<-cbind(runif(n,min = 0,max = 1),runif(n,min = 0,max = 1))
CVgridLocation<-cbind(runif(ncv,min = 0,max = 1),runif(ncv,min = 0,max = 1))
comboLocation<-rbind(gridLocation,CVgridLocation)
distMatFull<-as.matrix(rdist(comboLocation))
# Create Indices
modInd<-1:n
CVInd<-(n+1):nrow(distMatFull)
distMatMod<-distMatFull[modInd,modInd]
# Covariates
XMat<-cbind(runif(n,-1,1),runif(n,-1,1))
XMatCV<-cbind(runif(ncv,-1,1),runif(ncv,-1,1))
XB<-XMat%*%beta
cvXB<-XMatCV%*%beta
XBFull<-rbind(XB,cvXB)

# Covariance Matrix
CovMat<-sigma2*matCov(distMatFull,phi)

# Latent Gaussian Random Field
gpWFull <- as.numeric(rmvnorm(n=1,mean=rep(0,nrow(CovMat)),Sigma = CovMat,method = "chol"))
pWFullBin<-exp(gpWFull+XBFull)/(1+exp(gpWFull+XBFull))

# Observations
obsFullBin<-sapply(pWFullBin,rbinom,n=1,size=1)


###################
# Model Sample
# Latent Process
gpWMod<-gpWFull[modInd]
# Expected Value
pWModBin<-pWFullBin[modInd]
# Observations
obsModBin<-obsFullBin[modInd]

# CV Sample
# Latent Process
gpWCV<-gpWFull[CVInd]
# Expected Value
pWCVBin<-pWFullBin[CVInd]
# Observations
obsCVBin<-obsFullBin[CVInd]

# Truth CVMSPE
truthCVMSPEBin<-mean((pWCVBin-obsCVBin)^2)

#save.image("SpatialData800.RData")

######### 3. Fit binomial Regression ###############
#load("SpatialData.RData")
# Nimble Model
```

```
model_string <- nimbleCode({

  # Data Model
  for(i in 1:n){
    p[i] <- exp(W[i]+XB[i])/(1+exp(W[i]+XB[i]))
    Z[i] ~ dbern(p[i])
  }

  # Constant and Cov Matrix
  XB[1:n]<-beta1*X[,1] + beta2*X[,2]
  covMat[1:n,1:n]<- matcov(dists[1:n,1:n],phi)
  fullCovMat[1:n,1:n]<- sigma2*covMat[1:n,1:n]

  # Process Model
  W[1:n] ~ dmnorm(mean = mn[1:n], cov = fullCovMat[1:n,1:n])

  # Parameter Model
  sigma2   ~  dinvgamma(0.2, 0.2)
  phi   ~  dunif(0,1)
  beta1 ~  dnorm(0, sd=sqrt(100))
  beta2 ~  dnorm(0, sd=sqrt(100))
})

niter=200000
consts   <- list(n=n,X=XMat,dists=distMatMod,mn=rep(0,n))
data     <- list(Z=obsModBin)
inits    <- list(beta1=rnorm(1),beta2=rnorm(1),phi=0.5,sigma2=2,
                 W=rnorm(n))

# Run MCMC

pt<-proc.time()
samples  <- nimbleMCMC(model_string, data = data, inits = inits,
                       constants=consts,
                       monitors = c("beta1", "beta2","phi","sigma2","W"),
                       samplesAsCodaMCMC=TRUE,WAIC=FALSE,summary=FALSE,
                       niter = niter, nburnin = 10000, nchains = 1)
ptFinal800<-proc.time()-pt
ptFinal800



# Summary
# Table
summaryMat<-list()
summaryMat[[1]]<-round(summaryFunction(samples[,c("beta1", "beta2","phi","sigma2")],
                                       time=ptFinal800[3]),3)
summaryMat[[2]]<-round(summaryFunction(samples[,1:n],
                                       time=ptFinal800[3]),3)
summaryMat[[1]]


apply(summaryMat[[2]],1,mean)
save(summaryMat,samples,ptFinal800,file="BinomialMCMCResults800.RData")
ptFinal200[3]
ptFinal400[3] / ptFinal200[3]
```

```
ptFinal800[3] / ptFinal200[3]
ptFinal1600[3] / ptFinal200[3]


#load('BinimialMCMCResults800.RData')

# plot
pdf(file = "BinomialMCMCResults800.pdf",width=11,height=8.5)
par(mfrow=c(4,2),mar=c(2,2,2,2))
sampInd<-floor(seq(1,nrow(samples),length.out = 1000))
param_name <- c('beta1','beta2','phi','sigma2')
for(i in 1:4){
  plot.ts(samples[sampInd,param_name[i]]); abline(h=c(beta,phi,sigma2)[i],col="red",lwd=2)
  plot(density(samples[sampInd,i])); abline(v=c(beta,phi,sigma2)[i],col="red",lwd=2)
}
dev.off()

par(mfrow=c(2,2), mar = c(3,3,3,3))
acf(samples[,'beta1'])
acf(samples[,'beta2'])
acf(samples[,'phi'])
acf(samples[,'sigma2'])
dim(samples)


# prediction
params <-  summaryMat[[1]][1,]
params
predXB <- samples[,c('beta1','beta2')][180000:190000 ,] %*% t(XMatCV)


samples[,c('beta1','beta2')][180000:190000 ,]

dim(samples)[1]

rmvnorm(1, mean = 0, Sigma = samples[i,'sigma2']*matCov(distMatFull[CVInd,CVInd],samples[i,'phi']))

# Sample W_pred

Wpredmat <- matrix(NA, nrow = 10000, ncol = ncv)
dim(Wpredmat)
for (i in 1:10000){
  Wpredmat[i,] <- rmvnorm(1, mean = 0, Sigma = samples[i+180000,'sigma2']*matCov(distMatFull[CVInd,CV
}



p_pred <- matrix(NA, nrow = 10000, ncol = ncv)
for (i in 1:10000) {
  for (j in 1:ncv) {
    p_pred[i,j] <-  exp(Wpredmat[i,j] + predXB[i,j]) / (1+exp(Wpredmat[i,j] + predXB[i,j]))
  }
}

z_pred <- as.integer(apply(p_pred,2,mean) >= 0.5)
acc_800 = sum(z_pred == obsFullBin[CVInd]) / ncv
```

```
acc_800


############################
######### N = 1600 case#####
############################


#Parameters
set.seed(55555)
n=1280 ; ncv=320 #  n = training data, ncv = test data set
beta=c(1,1) ; phi=0.2 ; sigma2=1

# Generate Locations
## Split into Model + Cross-Validation
gridLocation<-cbind(runif(n,min = 0,max = 1),runif(n,min = 0,max = 1))
CVgridLocation<-cbind(runif(ncv,min = 0,max = 1),runif(ncv,min = 0,max = 1))
comboLocation<-rbind(gridLocation,CVgridLocation)
distMatFull<-as.matrix(rdist(comboLocation))
# Create Indices
modInd<-1:n
CVInd<-(n+1):nrow(distMatFull)
distMatMod<-distMatFull[modInd,modInd]
# Covariates
XMat<-cbind(runif(n,-1,1),runif(n,-1,1))
XMatCV<-cbind(runif(ncv,-1,1),runif(ncv,-1,1))
XB<-XMat%*%beta
cvXB<-XMatCV%*%beta
XBFull<-rbind(XB,cvXB)

# Covariance Matrix
CovMat<-sigma2*matCov(distMatFull,phi)

# Latent Gaussian Random Field
gpWFull <- as.numeric(rmvnorm(n=1,mean=rep(0,nrow(CovMat)),Sigma = CovMat,method = "chol"))
pWFullBin<-exp(gpWFull+XBFull)/(1+exp(gpWFull+XBFull))

# Observations
obsFullBin<-sapply(pWFullBin,rbinom,n=1,size=1)


##################
# Model Sample
# Latent Process
gpWMod<-gpWFull[modInd]
# Expected Value
pWModBin<-pWFullBin[modInd]
# Observations
obsModBin<-obsFullBin[modInd]

# CV Sample
# Latent Process
gpWCV<-gpWFull[CVInd]
# Expected Value
pWCVBin<-pWFullBin[CVInd]
# Observations
```

```
obsCVBin<-obsFullBin[CVInd]

# Truth CVMSPE
truthCVMSPEBin<-mean((pWCVBin-obsCVBin)^2)

#save.image("SpatialData1600.RData")

########## 3. Fit binomial Regression ###############
#load("SpatialData.RData")
# Nimble Model
model_string <- nimbleCode({

  # Data Model
  for(i in 1:n){
    p[i] <- exp(W[i]+XB[i])/(1+exp(W[i]+XB[i]))
    Z[i] ~ dbern(p[i])
  }

  # Constant and Cov Matrix
  XB[1:n]<-beta1*X[,1] + beta2*X[,2]
  covMat[1:n,1:n]<- matcov(dists[1:n,1:n],phi)
  fullCovMat[1:n,1:n]<- sigma2*covMat[1:n,1:n]

  # Process Model
  W[1:n] ~ dmnorm(mean = mn[1:n], cov = fullCovMat[1:n,1:n])

  # Parameter Model
  sigma2   ~  dinvgamma(0.2, 0.2)
  phi   ~  dunif(0,1)
  beta1 ~  dnorm(0, sd=sqrt(100))
  beta2 ~  dnorm(0, sd=sqrt(100))
})

niter=200000
consts   <- list(n=n,X=XMat,dists=distMatMod,mn=rep(0,n))
data     <- list(Z=obsModBin)
inits    <- list(beta1=rnorm(1),beta2=rnorm(1),phi=0.5,sigma2=2,
                 W=rnorm(n))

# Run MCMC

pt<-proc.time()
samples  <- nimbleMCMC(model_string, data = data, inits = inits,
                       constants=consts,
                       monitors = c("beta1", "beta2","phi","sigma2","W"),
                       samplesAsCodaMCMC=TRUE,WAIC=FALSE,summary=FALSE,
                       niter = niter, nburnin = 10000, nchains = 1)
ptFinal1600<-proc.time()-pt
ptFinal1600



# Summary
# Table
summaryMat<-list()
summaryMat[[1]]<-round(summaryFunction(samples[,c("beta1", "beta2","phi","sigma2")],
```

```
                                        time=ptFinal1600[3]),3)
summaryMat[[2]]<-round(summaryFunction(samples[,1:n],
                                        time=ptFinal1600[3]),3)
summaryMat[[1]]
apply(summaryMat[[2]],1,mean)
save(summaryMat,samples,ptFinal1600,file="BinomialMCMCResults1600.RData")
#load('BinimialMCMCResults1600.RData')

# plot
pdf(file = "BinomialMCMCResults1600.pdf",width=11,height=8.5)
par(mfrow=c(4,2),mar=c(2,2,2,2))
sampInd<-floor(seq(1,nrow(samples),length.out = 1000))
param_name <- c('beta1','beta2','phi','sigma2')
for(i in 1:4){
  plot.ts(samples[sampInd,param_name[i]]); abline(h=c(beta,phi,sigma2)[i],col="red",lwd=2)
  plot(density(samples[sampInd,i])); abline(v=c(beta,phi,sigma2)[i],col="red",lwd=2)
}
par(mfrow=c(2,2), mar = c(3,3,3,3))
acf(samples[,'beta1'])
acf(samples[,'beta2'])
acf(samples[,'phi'])
acf(samples[,'sigma2'])
dim(samples)


# prediction
params <-  summaryMat[[1]][1,]
params
predXB <- samples[,c('beta1','beta2')][180000:190000 ,] %*% t(XMatCV)


samples[,c('beta1','beta2')][180000:190000 ,]

dim(samples)[1]

rmvnorm(1, mean = 0, Sigma = samples[i,'sigma2']*matCov(distMatFull[CVInd,CVInd],samples[i,'phi']))

# Sample W_pred

Wpredmat <- matrix(NA, nrow = 10000, ncol = ncv)
dim(Wpredmat)
for (i in 1:10000){
  Wpredmat[i,] <- rmvnorm(1, mean = 0, Sigma = samples[i+180000,'sigma2']*matCov(distMatFull[CVInd,CV
}



p_pred <- matrix(NA, nrow = 10000, ncol = ncv)
for (i in 1:10000) {
  for (j in 1:ncv) {
    p_pred[i,j] <-  exp(Wpredmat[i,j] + predXB[i,j]) / (1+exp(Wpredmat[i,j] + predXB[i,j]))
  }
}

z_pred <- as.integer(apply(p_pred,2,mean) >= 0.5)
acc_1600 = sum(z_pred == obsFullBin[CVInd]) / ncv
```

acc_1600
ptFinal1600