Bivariate conditional spatial models: Case study in Section 5

Noel Cressie and Andrew Zammit-Mangion

Sunday, March 01, 2015

Setting up

An important component of this example is the slice sampler. Here we use the code by Jonathan Rougier, University of Bristol. As of writing, this was available from his personal webpage. Once downloaded this package may be installed using

```
install.packages("./slice_0.9.tar.gz",type="source",repos=NULL)
```

We will also be needing part of the `INLA` package and installation instructions for this can be found on the R-INLA homepage. Once these are installed we can load the required packages. As with the other vignette (simulation example in Section 3.2), we will also need `dplyr`, `tidyr`, and `Matrix` for core operations and `ggplot2`, `gridExtra`, `grid`, `extrafont` and `maptools` for plotting purposes.

```
library(dplyr)
library(tidyr)
library(Matrix)
library(INLA)
library(slice)
library(ggplot2)
library(gridExtra)
library(grid)
library(extrafont)
library(maptools)
```

Finally, we will also need the package `bicon` to facilitate matrix construction.

```
library(bicon)
```

As detailed in Cressie & Zammit-Mangion (2015), we consider three models in this case study. These vary only through the interaction function $b_o(h)$ that, for each model, is given as

$$\begin{array}{ll} \text{Model 1 (independence):} & b_o(h) \equiv 0, \\ \text{Model 2 (pointwise dependence):} & b_o(h) \equiv A\delta(h), \\ \text{Model 3 (diffused dependence):} & b_o(h) \equiv \begin{cases} A\{1 - (\|h-\Delta\|/r)^2\}^2, & \|h-\Delta\| \le r \\ 0, & \text{otherwise,} \end{cases} \end{array}$$

where $\Delta = (\Delta_1, \Delta_2)^{\mathrm{T}}$ is a shift parameter vector that captures asymmetry, r is the aperture parameter, and A is the amplitude. In Model 3, $b_o(h)$ is a shifted bisquare function in $\mathbb{R}^2$. The covariance functions $C_{11}(\cdot)$ and $C_{2|1}(\cdot)$ are Matérn covariance functions with $\nu_{11} = \nu_{2|1} = 1.5$.

The first thing we need to do is specify which model we are going to analyse. Since Model 3 is the most complicated, we will use this throughout this vignette. The second thing is to indicate whether we want to run the full MCMC scheme (takes a long time!) or just generate a few samples to make sure the code is working. For the purpose of this vignette we will use a flag `long_analysis` to indicate whether we want to run the full MCMC chain or not.

```
### Model choice
### 1. Independent
### 2. Pointwise interaction
### 3. Smoothing
model = 3
model_name <- switch(model,"independent","pointwise","moving_average")
img_path <- "../paper/art"                   ## Where to save the figures
show_figs <- 1                               ## Show the figures in document
print_figs <-  0                             ## Print figures to file (leave =0)
long_analysis <- 0                           ## Run a long MCMC chain (1) or not (0)
```

The data

The data were made available through the paper of Genton & Kleiber (2015) and is included in the package `extdata` folder. For convenience, these have also been included as part of the package after the data were preprocessed using

```
temps_path <- system.file("extdata","COTemp20040919.txt", package = "bicon")
temps <- read.table(temps_path,header=T)
names(temps) <- c("minT","maxT","lon","lat")
```

Thus, these data can simply be loaded using the `data` command

```
### Import data
###------------
data(temps)
```

The command above loads the data `temps` in the the global environment as a data frame. The data frame stores the minimum temperature (`minT`), the maximum temperature (`maxT`) and the lon-lat coordinates of the measurement stations (`lon` and `lat`).

```
print(head(temps))
```

```
##        minT      maxT       lon     lat
## 1  4.752128  7.796809 -103.1417 40.1550
## 2 -7.547872 -5.003191 -105.8919 38.9933
## 3 -6.447872 -1.603191 -105.4767 39.4047
## 4 -3.647872  4.496809 -108.8903 38.3189
## 5 -3.647872 -5.503191 -107.6872 38.5547
## 6  3.052128  7.796809 -102.1192 39.6567
```

From this data frame we extract Z_1 and Z_2 and concatenate them into one long vector Z. We also define `m1` as the number of observations of Y_1, `m2` as the number of observations of Y_2 and `m` as the total number of observations. Information pertaining to Z_1 and Z_2 are stored in the data frames `obs1` and `obs2`, respectively.

```
Z1 <- matrix(temps$minT)
Z2 <- matrix(temps$maxT)
Z <- rbind(Z1,Z2)
m1 <- length(Z1)
m2 <- length(Z2)
```

```
m <- m1 + m2

obs1 <- temps %>%
  mutate(x = lon, y = lat, z = minT)  %>%
  select(x,y,z)

obs2 <- temps %>%
  mutate(x = lon, y = lat, z = maxT)  %>%
  select(x,y,z)
```

Process discretisation

We approximate the processes as a sum of elemental basis functions (tent functions) constructed on a triangulation. The triangulation is formed using the mesher in the `INLA` package, while we provide a tailored function `initFEbasis` which takes information from the INLA mesher and casts it into a `Mesh` object. We provide several methods associated with the `Mesh` class which will be useful for plotting later on. Importantly, the `Mesh` object also contains information on the areas of the elements in the Voronoi tesselation, which will be used to approximate the integrations.

```
### Construct mesh
###------------
mesh <- inla.mesh.2d(loc= temps[c("lon","lat")],cutoff=0.04,max.edge=0.3)
D <- as.matrix(dist(mesh$loc[,1:2]))
Dvec <- as.double(c(D))
distmat11 <- distmat12 <- distmat21 <- distmat22 <-
    as.matrix(dist( as.matrix(temps[c("lon","lat")])))

Mesh <- initFEbasis(p = mesh$loc[,1:2],
                    t = mesh$graph$tv,
                    K = mesh$graph$vv)
```

```
##
##      PLEASE NOTE:  The components "delsgs" and "summary" of the
##      object returned by deldir() are now DATA FRAMES rather than
##      matrices (as they were prior to release 0.0-18).
##      See help("deldir").
##
##      PLEASE NOTE: The process that deldir() uses for determining
##      duplicated points has changed from that used in version
##      0.0-9 of this package (and previously). See help("deldir").
```

We next establish the dimension of our grids. Since we will be evaluating Y_1 and Y_2 on the same grid, `n1` = `n2`.

```
### Process models
###------------
n1 <- mesh$n
n2 <- mesh$n
n <- n1 + n2
```

As in the first vignette (simulation example in Section 3.2), we will approximate the integration using the rectangular rule. When using finite elements, this reduces to using the area of the Voronoi tessellation as the weight for the function values.

We first compute the vector of displacements h which will be of length (`n2` $\times$ `n1`) and with each element associate an integration weight equal to the area of the Voronoi tessellation associated with the element:

```
### Mesh integration points
###-----------------------
mesh_locs <- mesh$loc[,1:2]
h <- matrix(0,n1*n2,2)
areas <- rep(0,n1*n2)
for(i in 1:n2) {
  h[((i-1)*n1+1):(i*n1),] <- t(t(mesh_locs) - mesh_locs[i,])
  areas[((i-1)*n1+1):(i*n1)] <- Mesh["area_tess"]
}
h1_double <- as.double(h[,1])
h2_double <- as.double(h[,2])
```

The displacements (`h1`,`h2`) and the areas `areas` can then be used to construct the matrix `B` using the function `bisquare_B`.

```
###################
### Model specifics
###################
if (model == 1) B <- matrix(0,n2,n1)
if (model == 2) B <- diag(n1)
if (model == 3) B <- bisquare_B(h1 = h1_double,
                                h2 = h2_double,
                                delta=c(0,0),r=0.3,
                                n1 = n1,n2=n2,areas = areas)
```

Organising the observations

In order to map the process to the observations we construct an incidence matrix, which contains a `1` wherever the observation coincides with a vertex on the triangulation and a `0` otherwise. The dimension of this incidence matrix is (`m1` + `m2`) $\times$ (`n1` + `n2`), where `m1`, `m2`, are the number of observations in Z_1, Z_2, respectively. Since in this problem we have collocated observations, we find the incidence matrix for one of the observations, Z_1, and then form the whole incidence matrix by simply carrying out `bdiag` (block diagonal) of the first matrix with itself. We find the points with which the observation locations coincide by using the function `left_join`, which returns an `NA` if no observation coincides with the vertex.

```
obs_locs <- as.matrix(temps[c("lon","lat")])               ## observation locations
mesh_locs <- data.frame(x=mesh$loc[,1],y=mesh$loc[,2])      ## mesh locations
idx <- which(!(is.na(left_join(mesh_locs,obs1)$z)))         ## index of coincidence
C1 <- sparseMatrix(i=1:nrow(obs1),j=idx,x=1,dims=c(m1,n1))  ## incidence matrix of Z1
C <- bdiag(C1,C1)       ## incidence matrix
```

MCMC initialisation and prior specification

This is the computational and memory intensive part of the program. Since this is only a vignette we will only generate 10 samples. For the simulation study in Section 5, we used 50000 samples and thinned by keeping every 100. To carry out the full analysis, please set `long_analysis = 1` above.

```
### MCMC initialisations
###----------------------
if(long_analysis){
    N = 50000
    thin = 100
    burn_in = 1000
} else {
    N = 10
    thin = 1
    burn_in = 2
}
```

The marginal precisions were initialised to $\sigma_{11}^{-2} = 0.25$ and $\sigma_{2|1}^{-2} = 5$. The scales were initialised to $\kappa_1 = \kappa_{2|1} = 2$, $\rho = 0.1$, $A = 0$, $r = 0.3$ and $\Delta = (0,0)'$. Recall that we use transformations for some of these parameters. Therefore, for example, we initialise a variable `log_kappa1_samp` as $\log(2)$. Then, throughout, $\kappa_1 = \exp($`log_kappa1_samp`$)$.

```
Y_samp <- matrix(0,n,N/thin)
prec0_samp <- rep(1,N)
prec1_samp <- rep(0.25,N)
prec2_samp <- rep(5,N)

### kappa_1: init sample at log(2), kappa = exp(gamma)
log_kappa1_samp  <- matrix(log(2),1,N)

### kappa_{2|1}: init sample at log(2), kappa = exp(gamma)
log_kappa2_samp  <- matrix(log(2),1,N)

### rho_\epsilon: init sample at rho = 0.1, rho = 2*pnorm(gamma) - 1
rho_trans_samp <- rep(qnorm((0.1+1)/2),N)

### A: init sample at A = 0
A_samp <- rep(0,N)

### r: init sample at r = 0.3, r = exp(gamma)
log_r_samp <- rep(log(0.3),N)

### Delta_1 and Delta_2: init sample at Delta = (0,0)
d1_samp <- rep(0,N)
d2_samp <- rep(0,N)

### Sampling the Deviance
Deviance_samp <- rep(0,N)
```

We next specify the prior distributions over all unknown parameters. These are detailed in Table 2, Appendix 3 of Cressie & Zammit-Mangion (2015). In order to tune the prior distribution over the precision parameters we use a function `Findalphabeta_gamma` which, given the 5th and 95th percentiles for the desired standard deviations, finds appropriate values for α, β (shape and rate parameters) needed to define the Gamma distribution. Type `help(Findalphabeta_gamma)` for more details.

A new function appearing below is `makeQY`. This has an interface identical to `makeSY` but instead returns the precision matrix Σ^{-1}. This is useful since working with the precision matrices is slightly more computationally efficient than working with covariance matrices (even if the precision matrix is dense).

```r
###################################
### Parameter prior specification
###################################

### Y1,Y2 mean and precision
muY = rep(0,n)
QY <- makeQY(r = Dvec,
             var1 = 4,
             var2 = 0.2,
             kappa1 = 2,
             kappa2 = 2,
             B = 0*B)
QY_chol <- chol(QY)

### observation error precision
hyp_prec0 <- optim(par=c(1,1),Findalphabeta_gamma, p5=0.5, p95=2)
alpha0 <- hyp_prec0$par[1]
beta0 <- hyp_prec0$par[2]
Qo_base <-diag(m)
chol_Qo_base <-chol(Qo_base)

### log(kappa_1)
mu_log_kappa1 = log(2)
prec_log_kappa1 = 1

### log(kappa_{2|1}
mu_log_kappa2 = log(2)
prec_log_kappa2 = 1

### gamma, where rho = 2*pnorm(theta) - 1
mu_rho_trans = qnorm((1)/2)
prec_rho_trans = 1

### precision(Y1) and precision(Y2)
hyp_prec1 <- optim(par=c(1,1),Findalphabeta_gamma, p5=1, p95=10)
alpha1 <- alpha2 <-  hyp_prec1$par[1]
beta1 <- beta2 <- hyp_prec1$par[2]

### A (amplitude of bisquare function)
mu_A = 0
prec_A = 1/(0.2^2)

### Delta (shift parameter of bisquare function)
mu_delta1 = 0
prec_delta1 = 1/(0.5^2)
mu_delta2 = 0
prec_delta2 = 1/(0.5^2)

### r (aperture parameter of bisquare function)
mu_log_r = 1
prec_log_r = 1/(0.5^2)
```

The sampler

Before we start the MCMC routine, we need to initialise the slice sampler for those variables which have a conditional distribution that is not available in closed form. For the scale parameters and observation error correlation parameter we use univariate slice samplers, while for the parameters associated with the bisquare function we use a tri-variate slice sampler.

```
########################
### Gibbs-slice sampler
########################

slice_log_kappa1 <- slice::slice(1)
slice_log_kappa2 <- slice::slice(1)
slice_rho_trans <- slice::slice(1)
slice_bisquare <- slice::slice(3)
```

We can now start the Gibbs sampler, for which we will be requiring the conditional distributions. Denote $\theta = (\sigma^2_\varepsilon, \rho_\varepsilon, \sigma^2_{11}, \sigma^2_{2|1}, \kappa_{11}, \kappa_{2|1}, A, r)'$ as the collection of unknown parameters and $\tilde{\theta} = (\sigma^{-2}_\varepsilon, \tilde{\rho}_\varepsilon, \sigma^{-2}_{11}, \sigma^{-2}_{2|1}, \tilde{\kappa}_{11}, \tilde{\kappa}_{2|1}, A, \tilde{r})'$ as the transformed parameters we actually sample from. Recall from Cressie & Zammit-Mangion (2015) that $\rho_\varepsilon = 2\Phi(\tilde{\rho}_\varepsilon) - 1$, $r = \exp(\tilde{r})$ and $\kappa_i = \exp(\tilde{\kappa}_i)$. Then

$$\begin{aligned}
p(Y|Z,\tilde{\theta}) &\propto p(Z|Y,\tilde{\theta})p(Y|\tilde{\theta}) && (1)\\
p(\sigma^{-2}_\varepsilon|Y,Z,\tilde{\theta}^{/\sigma^{-2}_\varepsilon}) &\propto p(Z|Y,\tilde{\theta})p(\sigma^{-2}_\varepsilon) && (2)\\
p(\tilde{\rho}_\varepsilon|Y,Z,\tilde{\theta}^{/\tilde{\rho}_\varepsilon}) &\propto p(Z|Y,\tilde{\theta})p(\tilde{\rho}_\varepsilon) && (3)\\
p(\sigma^{-2}_{11}|Y,Z,\tilde{\theta}^{/\sigma^{-2}_{11}}) &\propto p(Y_1|\tilde{\theta})p(\sigma^{-2}_{11}) && (4)\\
p(\sigma^{-2}_{2|1}|Y,Z,\tilde{\theta}^{/\sigma^{-2}_{2|1}}) &\propto p(Y_2|Y_1,\tilde{\theta})p(\sigma^{-2}_{2|1}) && (5)\\
p(\tilde{\kappa}_{11}|Y,Z,\tilde{\theta}^{/\tilde{\kappa}_{11}}) &\propto p(Y_1|\tilde{\theta})p(\tilde{\kappa}_{11}) && (6)\\
p(\tilde{\kappa}_{2|1}|Y,Z,\tilde{\theta}^{/\tilde{\kappa}_{2|1}}) &\propto p(Y_2|Y_1,\tilde{\theta})p(\tilde{\kappa}_{2|1}) && (7)\\
p(A|Y,Z,\tilde{\theta}^{/A}) &\propto p(Y_2|Y_1,\tilde{\theta})p(A) && (8)\\
p(\Delta,\tilde{r}|Y,Z,\tilde{\theta}^{/(\Delta,\tilde{r})}) &\propto p(Y_2|Y_1,\tilde{\theta})p(\Delta)p(\tilde{r}) && (9)
\end{aligned}$$

We do not discuss the algorithm below in detail, as it simply carries out the updates according to the conditional distributions listed above. However, there are a couple of points are worth mentioning:

- Where appropriate, we sample directly using functons such as `rgamma`. When the conditional distribution is not available in closed form we use the slice sampler. When doing slice sampling, all we need to do is supply the sampler with the logarithm of the conditional distribution and a `learn` parameter, which indicates for how many samples we are adapting the interval width in the slice sampler. This is set equal to the burn-in in each case.
- To sample the precision and scale parameters, we need to construct the covariance and precision matrices corresponding to $C_{11}(\cdot)$ and $C_{2|1}(\cdot)$. For this we use the functions `makeS` and `makeQ` respectively.

```
Ycount=1
I_m1 <- Imat(m1)
set.seed(1) ## Fix seed
for(i in 2:N) {
  ## Sample Y
```

```
Qo_new <- prec0_samp[i-1]*Qo_base
QY_new <- QY +  prec0_samp[i-1] * as(crossprod(chol_Qo_base %*% C),"matrix")
R <- chol(QY_new)
muY_new <- matrix(backsolve(R,backsolve(R,t(C) %*% Qo_new %*% Z,transpose=T)),ncol=1)
Yi <- (muY_new + backsolve(R,rnorm(n = n)))[,1]

Y1i <- Yi[1:n1]
Y2i <- Yi[-(1:n1)]
if (i %% thin == 0) {
   Y_samp[,Ycount] <- Yi
   Ycount <- Ycount + 1
}

### Observation model parameters

## Sample prec(epsilon)
alpha0_new <- alpha0 - 1 + m/2
beta0_new  <- beta0 + 0.5*crossprod(chol_Qo_base %*% (Z - C %*%Yi))
prec0_samp[i] <- rgamma(n = 1,shape = alpha0_new,rate = as.numeric(beta0_new))

## Sample rho
ZCY <- (Z - C %*%Yi)
logf_a <- function(x) {
  a <- 2*(pnorm(x)) - 1
  Qo <- prec0_samp[i] * kronecker(solve(matrix(c(1,a,a,1),2,2)),I_m1)
  chol_Qo <- chol(Qo)
  as.numeric(0.5*(2 * sum(log(diag(chol_Qo)))) -
               0.5*crossprod(chol_Qo %*% ZCY) -
               0.5*(x - mu_rho_trans)^2*prec_rho_trans)
}

rho_trans_samp[i] <-  slice_rho_trans(rho_trans_samp[i-1] , logf_a, learn = i <= burn_in)
Qo_base <- kronecker(solve(matrix(c(1,2*pnorm(rho_trans_samp[i]) - 1,
                                    2*pnorm(rho_trans_samp[i]) - 1,1),2,2)),I_m1)
chol_Qo_base <- chol(Qo_base)

### Precision parameters

## Sample precision(Y1)
Q11base <- makeQ(r = Dvec,var = 1, kappa = exp(log_kappa1_samp[1,i-1]))
alpha1_new <- alpha1 - 1 + n1/2
beta1_new  <- beta1 + 0.5*(t(Y1i) %*% Q11base %*% Y1i)[,1]
prec1_samp[i] <- rgamma(n = 1,shape = alpha1_new,rate = beta1_new)

## Sample precision(Y2)
Q2_1base <- makeQ(r = Dvec, var=1,kappa= exp(log_kappa2_samp[1,i-1]))
alpha2_new <- alpha2 - 1 + n2/2
YB <- Y2i - A_samp[i-1]*B %*% Y1i
beta2_new  <- beta2 + 0.5*(t(YB) %*% Q2_1base %*% YB)[,1]
prec2_samp[i] <- rgamma(n = 1,shape = alpha2_new,rate = beta2_new)

### Interaction parameters
```

```
## Sample A
if(model %in% 2:3) {
  Q2_1 <- prec2_samp[i]*Q2_1base
  Y1_sub <- B %*% Y1i
  precA_new <- t(Y1_sub) %*% Q2_1 %*% Y1_sub + prec_A
  muA_new <- solve(precA_new,t(Y1_sub) %*% Q2_1 %*% Y2i + prec_A*mu_A)
  A_samp[i] <- as.numeric(muA_new + solve(chol(precA_new),rnorm(n = 1)))
}

## Sample delta,r
if(model == 3) {
  Q2_1 <- prec2_samp[i]*Q2_1base
  logf_B <- function(x) {
    BB <- bisquare_B(h1_double,h2_double,
                     delta=x[1:2],r=exp(x[3]),
                     n1 = n1,n2=n2,areas = areas)
    (-0.5*t(Y2i - A_samp[i]*BB %*% Y1i) %*% Q2_1 %*% (Y2i - A_samp[i]*BB %*% Y1i) -
     0.5*(x[1] - mu_delta1)^2*prec_delta1 -
     0.5*(x[2] - mu_delta2)^2*prec_delta2 -
     0.5*(x[3] - mu_log_r)^2*prec_log_r)[,1]

  }
  all_samp <-  slice_bisquare(c(d1_samp[i-1],d2_samp[i-1],log_r_samp[i-1]) ,
                                   logf_B, learn = i <= burn_in)

  d1_samp[i] <- all_samp[1]
  d2_samp[i] <- all_samp[2]
  log_r_samp[i] <- all_samp[3]
  B <- bisquare_B(h1_double,h2_double,
                  delta=all_samp[1:2],r=exp(all_samp[3]),
                  n1 = n1,n2=n2,areas = areas)
}

### Scale parameters

## Sample scale1
logf_k1 <- function(x) {
  SY <- makeS(r = Dvec, var = 1/prec1_samp[i], kappa = exp(x))
  R <- chol(SY)
  (-0.5*logdet(R) -
     0.5*crossprod(forwardsolve(t(R),Y1i)) -
     0.5*(x[1] - mu_log_kappa1)^2*prec_log_kappa1)[,1]
}
log_kappa1_samp[1,i] <- slice_log_kappa1(log_kappa1_samp[1,(i-1)] ,
                                    logf_k1, learn = i <= burn_in)

## Sample scale2
YB <- Y2i - A_samp[i]*B %*% Y1i
logf_k2 <- function(x) {
  SY <- makeS(r = Dvec, var = 1/prec2_samp[i], kappa = exp(x))
  R <- chol(SY)
  (-0.5*logdet(R) -
     0.5*crossprod(forwardsolve(t(R),YB)) -
```

```
      0.5*(x - mu_log_kappa2)^2*prec_log_kappa2)[,1]
  }
  log_kappa2_samp[1,i] <- slice_log_kappa1(log_kappa2_samp[1,(i-1)] ,
                                           logf_k2, learn = i <= burn_in)

  ### Form prevision matrix from updated parameters
  QY <- makeQY(r = Dvec,
               1/prec1_samp[i],
               1/prec2_samp[i],
               exp(log_kappa1_samp[1,i]),
               exp(log_kappa2_samp[1,i]),
               A_samp[i]*B)

  ### Compute deviance
  Deviance_samp[i] <-
      as.numeric(-2*( -0.5*m*log(2*pi) +
      0.5*logdet(L = sqrt(prec0_samp[i])*chol_Qo_base) -
      0.5*prec0_samp[i] * t(ZCY) %*% Qo_base %*% ZCY))

  if(!(i%%1)) cat(paste0("Iteration ",i),sep="\n")

}
```

```
## Iteration 2
## Iteration 3
## Iteration 4
## Iteration 5
## Iteration 6
## Iteration 7
## Iteration 8
## Iteration 9
## Iteration 10
```

Computing the DIC

The discrepancy between data and model can be be assessed using the *deviance*

$$D(Z, Y, \theta) \equiv -2 \log p(Z \mid Y, \theta). \tag{10}$$

Since the deviance is a function of the unknowns Y and θ we use the average deviance from the posterior realisations of (Y, θ)

$$\widehat{D}(Z) \equiv \frac{1}{N} \sum_{i=1}^{N} D(Z, Y^{(i)}, \theta^{(i)}). \tag{11}$$

where N is the number of samples.
The average deviance alone is not sufficient when comparing across models with differing complexities. In deviance-based strategies, a penalty for model complexity p_D is obtained by computing the difference between $\widehat{D}(Z)$ and the deviance at the posterior means of Y, θ, $D(Z; \widehat{Y}, \widehat{\theta})$. Then the *Deviance Information Criterion* (DIC) is simply the addition of the average deviance and the penalty (Spiegelhalter, Best, Carlin,

& Van Der Linde, 2002). This results in the following formula:

$$DIC \equiv 2\widehat{D}(Z) - D(Z; \widehat{Y}, \widehat{\theta}). \tag{12}$$

We find the mean deviance and the DIC from the samples as follows.

```r
MCMC_sub <- seq(burn_in,N,by=thin)
Qo_base <- kronecker(solve(matrix(c(1,2*pnorm(mean(rho_trans_samp[MCMC_sub]))-1,
                        2*pnorm(mean(rho_trans_samp[MCMC_sub]))-1,1),2,2)),I_m1)
Deviance_mean <- as.numeric(-2*( -0.5*m*log(2*pi) +
                           0.5*logdet(L = chol(mean(prec0_samp[MCMC_sub])*Qo_base)) -
                           0.5*crossprod(chol(mean(prec0_samp[MCMC_sub])*Qo_base) %*%
                                         (Z - C %*%apply(Y_samp,1,mean)))))
DIC <- 2*mean(Deviance_samp[MCMC_sub]) - Deviance_mean
```

Obviously, with such few samples the results are not of much use. Instead, we saved the results using `N`=50000, `burn_in` = 1000 and `thin` =100 using the following piece of code

```r
### Should only be run with path set as vignette source directory
if (long_analysis)
save(model_name,
     MCMC_sub,
     Y_samp,
     prec0_samp,
     rho_trans_samp,
     prec1_samp,
     prec2_samp,
     log_kappa1_samp,
     log_kappa2_samp,
     A_samp,
     log_r_samp,
     d1_samp,
     d2_samp,
     Deviance_samp,
     Deviance_mean,
     DIC,
     file=paste0("../inst/extdata/",model_name,".rda"))
```

and now re-load these results using

```r
if(!long_analysis) {
  if(model == 1)  load(system.file("extdata","independent.rda", package = "bicon"))
  if(model == 2)  load(system.file("extdata","pointwise.rda", package = "bicon"))
  if(model == 3)  load(system.file("extdata","moving_average.rda", package = "bicon"))
}
```

If we wish to write out the results to a text file we can use the following code (not evaluated)

```r
fileConn<-file(paste0("../paper/",model_name,".txt"))
print_row <- function(x) {
  fline <- paste0(round(median(x[MCMC_sub]),digits=2)," [",
                  round(quantile(x[MCMC_sub],c(0.25)),digits=2),", ",
                  round(quantile(x[MCMC_sub],c(0.75)),digits=2),"]")

}
writeLines(c(print_row(1/prec0_samp),
             print_row(2*pnorm(rho_trans_samp)-1),
             print_row(1/prec1_samp),
             print_row(1/prec2_samp),
             print_row(exp(log_kappa1_samp[1,])),
             print_row(exp(log_kappa2_samp[1,])),
             print_row(A_samp),
             print_row(exp(log_r_samp)),
             print_row(d1_samp),
             print_row(d2_samp),
             print_row(Deviance_samp),
             paste0(round(mean(Deviance_samp[MCMC_sub]),digits=2)),
             paste0(round(mean(Deviance_samp[MCMC_sub]) - Deviance_mean,digits=2)),
             paste0(round(DIC,digits=2))),
           fileConn)
close(fileConn)
```

Plotting

As in the other vignette (simulation example of Section 3.2), we only briefly describe the code used to plot the results. The first plot shows the domain of interest around the state of Colorado. The shapefile required is available in the `extdata` folder of the package and is accessible through

```r
shapefile_path <- system.file("extdata", "cb_2013_us_state_5m.shp", package = "bicon")
States <- maptools::readShapeSpatial(shapefile_path)
```

We now extract these data into a data frame, and add coordinates for the labels in a data frame `State_summaries`. We then find the convex hull of our triangulation (domain boundary) using `chull`.

```r
### DOMAIN PLOT
States_fort <- fortify(States) %>% mutate( id= as.numeric(id))
State_names <- data.frame(Name = States$NAME, id = 0:(length(States$NAME)-1))

States_fort <- left_join(States_fort,State_names)
State_summaries <- group_by(States_fort,Name) %>%
  summarise(long = mean(long), lat = mean(lat))
State_summaries[4,c("long","lat")] <- c(-112,35.22)
State_summaries[46,c("long","lat")] <- c(-101.09198,  43.80577)
State_summaries[32,c("long","lat")] <- c(-118.75965,  37.96324)
State_summaries[31,c("long","lat")] <- c(-99.79460,  41.49899)
State_summaries[20,c("long","lat")] <- c(-99.10218,  38.75878)
State_summaries[35,c("long","lat")] <- c(-105.71772,  34.71201)
State_summaries[40,c("long","lat")] <- c(-97.32183,  35.72782)

conv_hull <- Mesh[c("x","y")][chull(Mesh[c("x","y")]),]
conv_hull <- rbind(conv_hull,conv_hull[1,])
```

We can now plot the domain of interest overlaying the state boundaries.

```
Stateplot <- LinePlotTheme() +
  geom_path(data=States_fort,aes(long,lat,group=group,label=Name),linetype="dotted") +
  geom_text(data=State_summaries,aes(long,lat,label=Name)) +
  coord_fixed(,xlim=c(-115,-95),ylim=c(33,45)) +
  geom_path(data=conv_hull,aes(x,y),size=1,,colour="black") +
  theme(plot.margin = grid::unit(c(2, 2, 2, 2),units="mm")) + xlab("lon")
if(print_figs)
    ggsave(Stateplot,
           filename = file.path(img_path,"Stateplot.eps"),
           width=7,height=4,family="Arial")
if(show_figs)
    print(Stateplot,family="Arial")
```

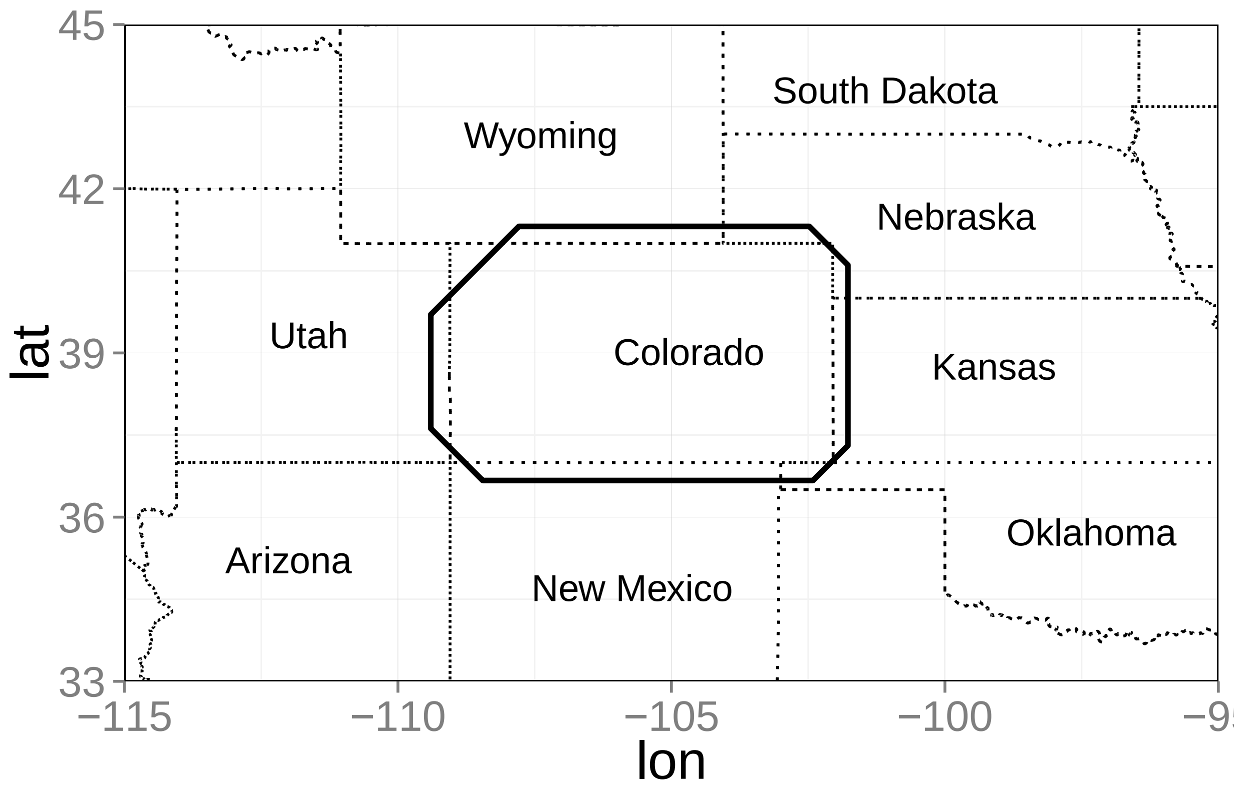

Figure 1: State boundaries in a region of the USA (dashed lines), with the domain of interest enclosed by a bounding polygon (solid line).

The following code plots the triangulation together with the observation locations.

```
### MESH PLOT
meshplot <- function(g,include_obs=1L) {
  p <- plot(Mesh,g=g,plot_dots=F)
  p <- p +
    xlab('lon') + ylab('lat') +
    coord_fixed(xlim=c(-110,-101.5),ylim=c(36.5,41.5)) +
    geom_path(data=States_fort,aes(long,lat,group=group,label=Name),linetype="dotted") +
    geom_path(data=conv_hull,aes(x,y),size=1,,colour="black")
  if(include_obs) p <- p + geom_point(data = temps,aes(lon,lat),size=3)
  p
}
Colorado_Mesh  <- meshplot(LinePlotTheme())
if(print_figs)
    ggsave(Colorado_Mesh,
           filename = file.path(img_path,"meshplot.eps"),
           width=7,height=4,family="Arial")
if(show_figs)
    print(Colorado_Mesh,family="Arial")
```

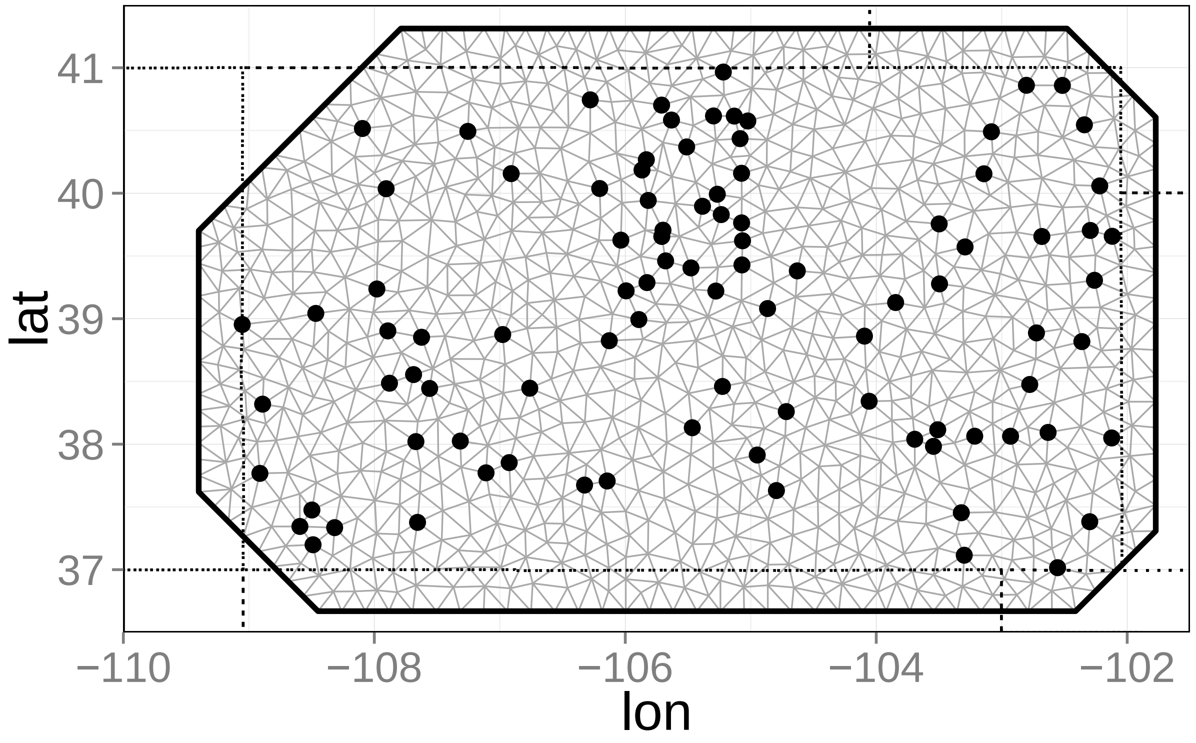

Figure 2: The irregular triangular grid used for discretising D. The observation locations given by D^O consist of the large dots and the discretized spatial domain D^L consists of the mesh nodes.

To combine the figures into a single panel we use

```
g <- arrangeGrob(Stateplot,Colorado_Mesh,ncol=2)
if(print_figs) ggsave(g,
                      filename = file.path(img_path,"Fig2.eps"),
                      width=14,height=4,family="Arial")
```

Next, we plot the estimated fields together with some major cities in the state of Colorado. For the cities, we need the shapefile `ci08au12.shp`, also available in `extdata`. This can be loaded and subsetted using:

```
shapefile_path <- system.file("extdata", "ci08au12.shp", package = "bicon")
Cities <- maptools::readShapeSpatial(shapefile_path) %>%
          as.data.frame() %>%
          subset(STATE == "COLORADO" &
                 NAME %in% c("DENVER","COLORADO SPRINGS","PUEBLO","FORT COLLINS"))
```

Now we do some pre-processing and print the required plots

```r
Cities$NAME <- as.character(Cities$NAME)
Cities$n <- sapply(Cities$NAME,nchar)
Cities[which(Cities$NAME == "COLORADO SPRINGS"),]$NAME <- "COL. SPRINGS"
Denver <- subset(Cities,NAME == "DENVER")

mesh_cities <- function(g,pt_size=3,txt_size=6) {
  g + geom_point(data=Cities,aes(LON,LAT),size=pt_size,shape=17,colour="black") +
    geom_text(data=Cities,aes(LON,LAT,label=NAME),hjust=-0.07,vjust=0,size=txt_size)

}

Mesh["Y1"] <- apply(Y_samp,1,mean)[1:n1]
Mesh["Y2"] <- apply(Y_samp,1,mean)[-(1:n1)]
Mesh["diffY"] <- Mesh["Y2"] - Mesh["Y1"]

Y1plot <- plot_interp(Mesh,"Y1",ds=200,min=-7.9,max=7.9,leg_title="Y1 (degC)",reverse=T) %>%
  meshplot(include_obs=0L) %>%
  mesh_cities()

Y1plot2 <- LinePlotTheme() %>% ## BW version
  meshplot(include_obs=0L) %>%
  mesh_cities(pt_size=6,txt_size=10) +
  geom_point(data=Mesh@pars$vars,aes(x=x,y=y,size=abs(Y1),shape=Y1>0))+
  scale_size_continuous((name="Y2 (deg C)")) +
  scale_shape_manual(values = c(19,1),guide=F) + xlab("") +
  theme(text = element_text(size = 35))

Y2plot <- plot_interp(Mesh,"Y2",ds=200,min=-7.9,max=7.9,leg_title="Y2 (degC)",reverse=T) %>%
  meshplot(include_obs=0L)  %>%
  mesh_cities()

Y2plot2 <- LinePlotTheme() %>%   ## BW version
  meshplot(include_obs=0L) %>%
  mesh_cities(pt_size=6,txt_size=10) +
  geom_point(data=Mesh@pars$vars,aes(x=x,y=y,size=abs(Y2),shape=Y2>0))+
  scale_size_continuous(name="Y2 (deg C)") +
  scale_shape_manual(values = c(19,1),guide=F) + xlab("") +
  theme(text = element_text(size = 35))


if(print_figs) {
    ggsave(Y1plot %>%
           mesh_cities(pt_size=6,txt_size=10) + xlab(""),
           filename = file.path(img_path,"Y1est.png"),width=13,height=7)
    ggsave(Y2plot %>%
           mesh_cities(pt_size=6,txt_size=10) ,
           filename = file.path(img_path,"Y2est.png"),width=13,height=7)
}
```

```r
knitr::opts_chunk$set(dev = 'pdf')
if(show_figs) print(Y1plot %>%
                        mesh_cities(pt_size=6,txt_size=6)+
                        theme(text = element_text(size = 15)) +
                        theme(legend.key.height=unit(2,"line")),
                        width=13,height=7)
```

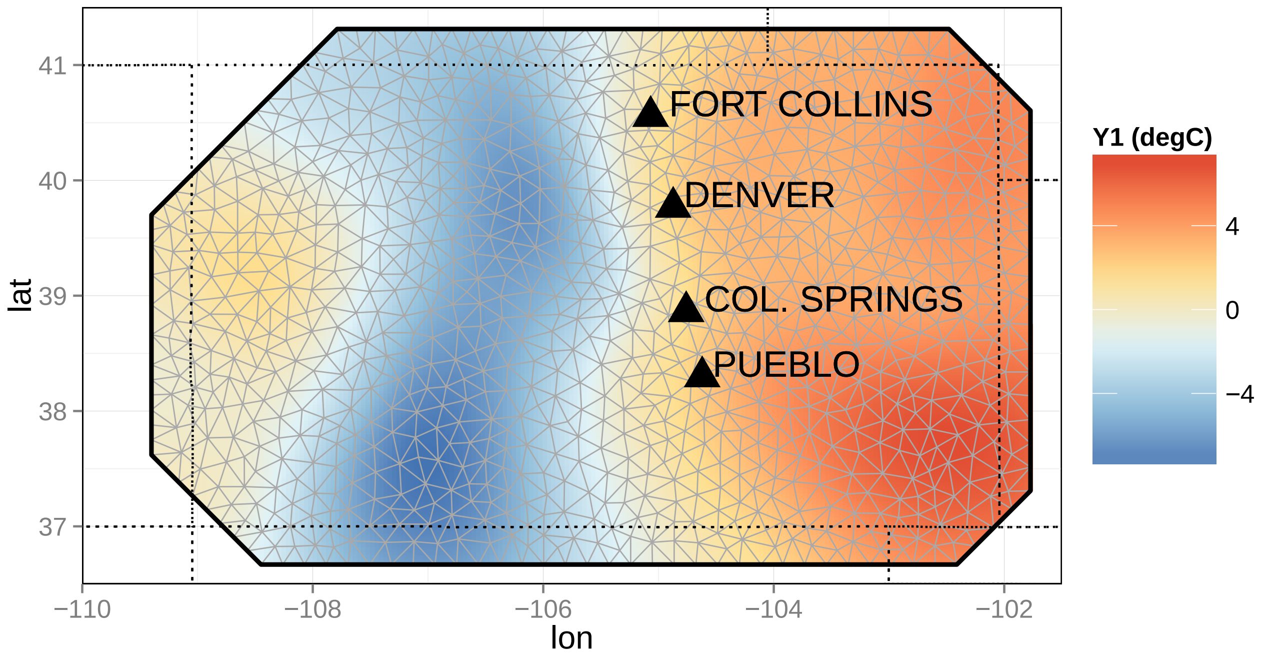

Figure 3: Interpolated map in degrees Celsius (degC) of $E(Y_1 \mid Z_1, Z_2)$, overlaid on the triangulation

```
if(show_figs)   print(Y2plot %>%
                          mesh_cities(pt_size=6,txt_size=6) +
                          theme(text = element_text(size = 15)) +
                          theme(legend.key.height=unit(2,"line")),
                          width=13,height=7)
```

```
### KERNEL VECTOR PLOT
if(model==3) {

  ## Kernel plots
  dh = 0.04
  d1_samp_prior <- rnorm(mu_delta1,sd=1/sqrt(prec_delta1),n=length(MCMC_sub))
  d2_samp_prior <- rnorm(mu_delta1,sd=1/sqrt(prec_delta2),n=length(MCMC_sub))
  log_r_samp_prior <- rnorm(mu_log_r,sd=1/sqrt(prec_log_r),n=length(MCMC_sub))
  A_samp_prior <- rnorm(mu_A,sd=1/sqrt(prec_A),n = length(MCMC_sub))

  dir_vector <- subset(Cities,NAME %in% c("DENVER","FORT COLLINS")) %>%
                select(LON,LAT) %>%
                summarise(x = diff(LON),y=diff(LAT))
  abs_dv <- sqrt(sum(dir_vector^2))
  hplot_post <- hplot_prior <- outer(seq(0,3,by=dh),t(dir_vector))[,,1] %>%
                               data.frame() %>%
                               mutate(he = sqrt(x^2 + y^2))
```

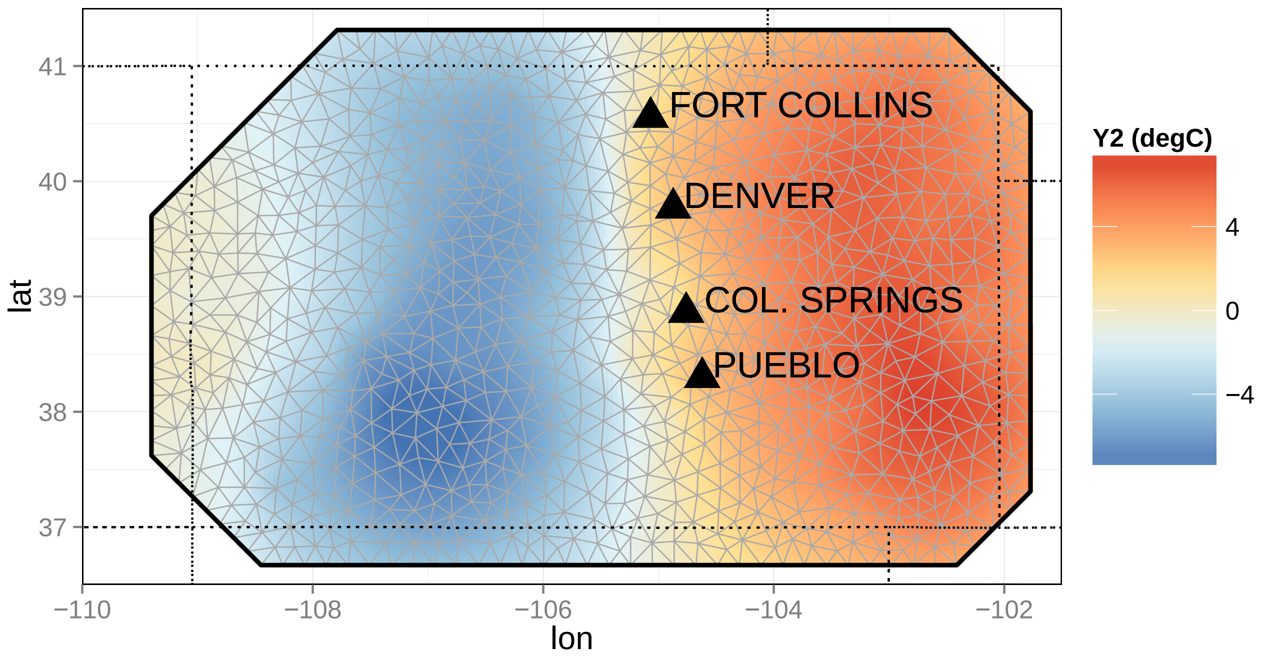

Figure 4: Interpolated map in degrees Celsius (degC) of $E(Y_2 \mid Z_1,Z_2)$, overlaid on the triangulation

```
for(i in MCMC_sub[-1])
  hplot_post[,paste0("MCMC.",i)] <- bisquare_2d(hplot_post$x,
                                          hplot_post$y,
                                          delta = c(d1_samp[i],
                                          d2_samp[i]),
                                          r = exp(log_r_samp[i]),
                                          A = A_samp[i])

for(i in 1:length(MCMC_sub))
  hplot_prior[,paste0("MCMC.",i)] <-  bisquare_2d(hplot_prior$x,
                                            hplot_prior$y,
                                            delta = c(d1_samp_prior[i],
                                            d2_samp_prior[i]),
                                            r = exp(log_r_samp_prior[i]),
                                            A = A_samp_prior[i])


process_h <- function(h) {
  select(h,-x,-y) %>%
    gather(MCMC,z,-he) %>%
    group_by(he) %>%
    summarise(lq = quantile(z,0.25),
              mq = quantile(z,0.5),
              uq = quantile(z,0.75))
}

hpost <- process_h(hplot_post)
hprior <- process_h(hplot_prior)

plot_h <- function(g,h,col="red",al=0.9) {
    g + geom_ribbon(data=h,aes(x = he, ymin = lq, ymax = uq),
                    fill=col,alpha=al) +
      geom_line(data=h,aes(x=he,y=lq),linetype=2,size=0.4) +
      geom_line(data=h,aes(x=he,y=mq),linetype=1,size=0.4) +
      geom_line(data=h,aes(x=he,y=uq),linetype=2,size=0.4)
  }
  g <- plot_h(LinePlotTheme(),hprior,col="light grey",al=1) %>%
  plot_h(hpost,col="black",al=0.5) +
    ylim(-0.15,0.4) +
    ylab(expression(paste(b[o],"(h",bold(e),")",sep = "")))+
    xlab("h") +
    coord_fixed(xlim=c(0,2.43),ratio = 5,ylim = c(-0.15,0.4)) +
    geom_line(data= data.frame(x = c(0,0,abs_dv,abs_dv),
                               y=c(-0.15,-0.14,-0.15,-0.14),
                               group=c(1,1,2,2)),
              aes(x,y,group=group),size=4) +
    theme(plot.margin=grid::unit(c(2.5,2.5,0,0),"mm"))
  gshow <- g + geom_text(data=data.frame(x=c(0.13,abs_dv+0.13),
                              y=c(-0.125,-0.125),
                              txt=c("DE","FC")),
              aes(x,y,label=txt),size=6)
  g <- g + geom_text(data=data.frame(x=c(0.13,abs_dv+0.13),
                              y=c(-0.125,-0.125),
                              txt=c("DE","FC")),
              aes(x,y,label=txt),size=20)

  if (print_figs) ggsave(g,filename=file.path(img_path,"Denver_FortCollins_vector.png"))
  if (show_figs) print(gshow)

  g_all <- arrangeGrob(arrangeGrob(Y1plot,Y2plot,ncol=1),g +
                         theme(text = element_text(size = 50)),ncol=2)
  if (print_figs) {
      cairo_ps(filename = file.path(img_path,"Fig3.eps"),
               width=28,height=16,family="Arial")
      print(g_all)
      dev.off()
  }
  g_all <- arrangeGrob(arrangeGrob(Y1plot2,Y2plot2,ncol=1),g +
                           theme(text = element_text(size = 50)),ncol=2)
  if (print_figs){
      cairo_ps(filename = file.path(img_path,"Fig3BW.eps"),
               width=28,height=16,family="Arial")
      print(g_all)
      dev.off()
  }

}
```

References

Cressie, N., & Zammit-Mangion, A. (2015). Multivariate spatial covariance models: A conditional approach. *Submitted*.

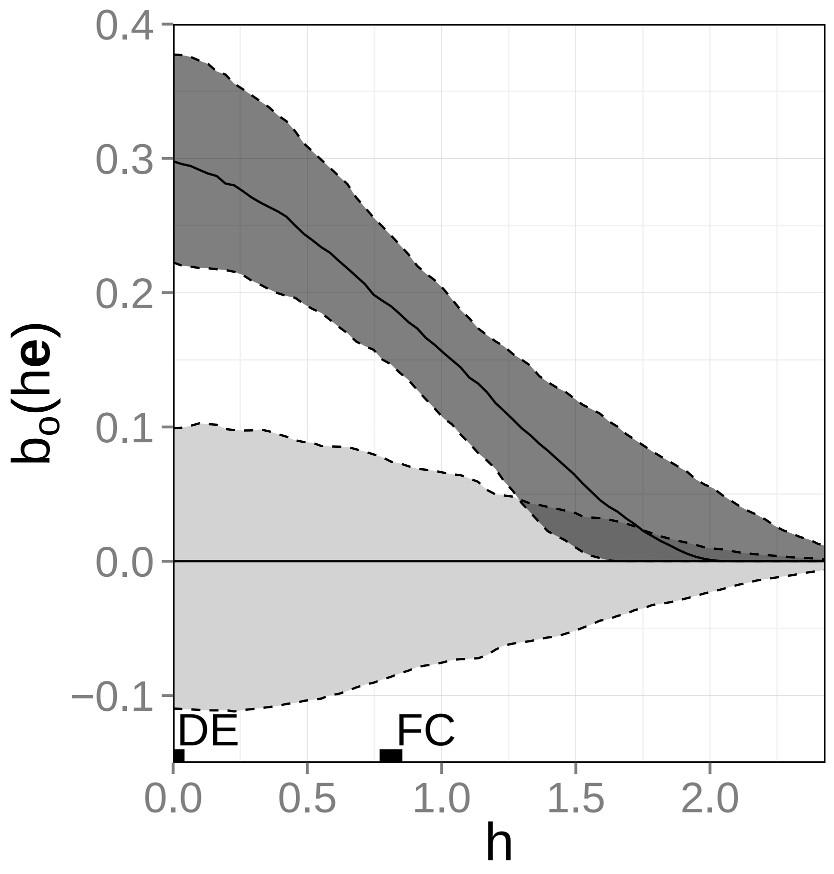

Figure 5: Prior (light grey) and posterior (dark grey) median (solid line) and inter-quartile ranges (enclosed by dashed lines) of the interaction function $b_o(h)$ of Model 3, along a unit vector e originating at Denver (DE) in the direction of Fort Collins (FC).

Genton, M. G., & Kleiber, W. (2015). Cross-covariance functions for multivariate geostatistics. *Statistical Science*, *30*, to appear.

Spiegelhalter, D. J., Best, N. G., Carlin, B. P., & Van Der Linde, A. (2002). Bayesian measures of model complexity and fit. *Journal of the Royal Statistical Society, Series B*, *64*, 583–639.