Bivariate conditional spatial models: Case study in Section 5

Noel Cressie and Andrew Zammit-Mangion

Setting up

In this document we show the *R Software* (R Core Team, 2014) code used to reproduce the results shown in Section 5 of Cressie & Zammit-Mangion (2016). The code for the application study in Section 3.2 is available in a separate document. Both these documents are available as vignettes in the package `bicon` which can be installed by first installing and loading the package `devtools` and then running

```
install_github("andrewzm/bicon")
```

The package will take some time to download since all the results of Section 5 are dispatched with the package.

As part of this example we will be needing the `INLA` package for mesh construction, and installation instructions for this can be found on the R-INLA homepage. Once this is installed we can load the required packages. As with the other document (simulation example in Section 3.2), we will also need `dplyr`, `tidyr`, and `Matrix` for core operations and `ggplot2`, `gridExtra`, `grid`, `extrafont` for plotting purposes. In addition, in this vignette we will need `maptools`, `mapproj`, and `RandomFields` (that provides the data used in this problem), and the `verification` package that contains a handy routing for computing CRPSs. For parallel operations we will be requiring `foreach` and `doParallel` and possibly `doMPI` if an MPI backend is available.

```
library(INLA)
library(dplyr)
library(tidyr)
library(Matrix)
library(ggplot2)
library(gridExtra)
library(grid)
library(extrafont)
library(maptools)
library(mapproj)
library(RandomFields)
library(verification)
library(foreach)
library(doParallel)
```

Finally, we will also need the package `bicon` mentioned above.

```
library(bicon)
```

As detailed in Cressie & Zammit-Mangion (2016), in this example we consider four models that vary only through the interaction function $b_o(h)$. The models are

$$
\begin{array}{ll}
\text{Model 1 (independent Matérns):} & b_o(h) \equiv 0, \\
\text{Model 2 (pointwise dependence):} & b_o(h) \equiv A\delta(h), \\
\text{Model 3 (diffused dependence):} & \text{Model 4 with } \Delta = 0 \\
\text{Model 4 (asymmetric dependence):} & b_o(h) \equiv \begin{cases} A\{1-(\|h-\Delta\|/r)^2\}^2, & \|h-\Delta\| \le r \\ 0, & \text{otherwise,} \end{cases}
\end{array}
$$

where $\Delta = (\Delta_1, \Delta_2)^\top$ is a shift parameter vector that captures asymmetry, r is the aperture parameter, and A is a scaling parameter. In Models 3 and 4, $b_o(h)$ is a shifted bisquare function in $\mathbb{R}^2$. The covariance functions $C_{11}(\cdot)$ and $C_{2|1}(\cdot)$ are Matérn covariance functions. For each model we also consider a *reversed* dependence, where we switch Y_2 and Y_1. This gives us a total of eight models to fit and compare.

First we set program options, indicating which parts of the program we want to run and which parts we want to skip, loading results from cache instead. The flag `LK_analysis` indicates whether we want to carry out a standard likelihood fit of all 8 models using the entire dataset (requires about 30 minutes of computing time) and the flag `LOO_analysis` indicates whether we want to run the leave-one-out cross-validation study (without re-fitting). In this vignette we set both flags to zero; the results are instead loaded from cache. If you wish to run parts of the program from scratch, please omit or comment out any of the `save` commands encountered below.

```
### Model choice
model_names <- c("independent","pointwise","moving_average_delta0","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)
LK_analysis <- 0                            ## Carry out likelihood analysis
LOO_analysis <- 0                           ## Carry out LOO analysis
Shifted_Pars_estimation <- 0                ## Fit shifted parimonious Matern
RF_estimation <- 0                          ## Carry out LOO with RFields
useMPI <- 0                                 ## MPI backend available?
```

The data

The data were made available through the package `RandomFields` and was studied at length in the paper of Gneiting, Kleiber, & Schlather (2010). We first load the data.

```
data(weather,package = "RandomFields")
weather <- weather %>% data.frame()
print(head(weather))
```

```
##   pressure temperature      lon    lat
## 1 200.4844  0.60537720 -131.000 46.000
## 2 384.8516 -0.02233887 -124.400 41.900
## 3 156.8984 -0.26644897 -124.500 46.100
## 4 248.4297 -1.30670166 -124.700 47.300
## 5 253.2266  0.14398193 -124.500 44.600
## 6 159.2031 -0.27954102 -124.985 49.907
```

The `weather` table contains four fields, with latitude, longitude, pressure forecasting errors and temperature forecasting errors for December 13, 2003 at 4 p.m. in the North American Pacific Northwest. Since pressure and temperature have different units, we find a scaling factor by taking the ratio of the variances of the two variates, and computing its square root. We will use this factor to scale the pressure variable.

```
P_scale <- (var(weather$pressure)/var(weather$temperature)) %>%
            sqrt() %>%
            as.numeric()
```

From this data frame we extract Z_1 and Z_2 and concatenate them into one long vector Z through a function `form_Z`. The vectors Z_1 and Z_2 are inverted if the model being analysed is greater than 4 (reversed model). 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.

```
form_Z <- function(model_num,scale=T) {
  Z1 <- matrix(weather$temperature)
  Z2 <- matrix(weather$pressure)
  if(scale) Z2 <- Z2 / P_scale # Change pressure to have similar scale
  if(model_num > 4) {          # Switch Z1 and Z2
    temp <- Z1
    Z1 <- Z2
    Z2 <- temp
  }
  Z <- rbind(Z1,Z2)            # Concatenate
}

m1 <- m2 <- nrow(weather)    # Number of observations of Y1 and Y2
m <- m1 + m2                 # Total number of observations
I_m1 <- Diagonal(m1)         # Identity matrix of size m1 x m1
```

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 in the package `bicon`, `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= weather[c("lon","lat")],
                     cutoff=0,
                     max.edge=0.75,offset = 4) ## Fine mesh
mesh_locs <- mesh$loc[,1:2]

## Compute distances as in Gneiting (2010) -- great circle distances
D <- as.matrix(RFearth2dist(as.matrix(mesh_locs)))
Dvec <- as.double(c(D))
Dobs <- as.matrix(RFearth2dist(as.matrix(weather[c("lon","lat")])))
Dobsvec <- c(Dobs)

## Cast into custom Mesh object
Mesh <- initFEbasis(p = mesh_locs,
                    t = mesh$graph$tv,
                    K = mesh$graph$vv)
```

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

```
### Mesh sizes
###-----------
n1 <- nrow(mesh_locs)
n2 <- nrow(mesh_locs)
n <- n1 + n2
```

As in the first document (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 integration weight.

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
###-----------------------
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` will be used to construct the matrix `B` using the function `bisquare_B`.

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 co-located observations, we find the incidence matrix for one of the observations, Z_1, and then form the whole incidence matrix by simply constructing a block diagonal matrix (using `bdiag`). 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.

```
mesh_locs <- data.frame(lon=mesh_locs[,1],lat=mesh_locs[,2])      ## mesh locations
idx <- which(!(is.na(left_join(mesh_locs,weather)$temperature)))  ## index of coincidence
C1 <- sparseMatrix(i=1:m1,j=idx,x=1,dims=c(m1,n1))  ## incidence matrix of Z1
C <- bdiag(C1,C1)                                   ## incidence matrix
```

Maximum likelihood estimation

Since the optimisation algorithm requires a parameter vector of the same length (irrespective of the model number) we first define a function `append_theta` that takes the parameter vector associated with the model in question and appends it so it is of the required size (in this case of length 11).

```
append_theta <- function(theta,model_num) {
  if(model_num %in% c(1,5)) {
    theta <- c(theta,rep(0,4))
    theta[10] <- 0.001
  } else if(model_num %in% c(2,6)) {
    theta <- c(theta,rep(0,3))
    theta[10] <- 0.001
  } else if(model_num %in% c(3,7)) {
    theta <- c(theta,rep(0,2))
  }
```

```
  theta
}
```

Next, we require a function that, given the parameter vector `theta` and the model number `model_num`, returns the required matrices and vectors used in fitting. These are the matrices

$$\texttt{SY} = \begin{bmatrix} \Sigma_{11} & \Sigma_{11}B^{\mathrm{T}} \\ B\Sigma_{11} & \Sigma_{2|1} + B\Sigma_{11}B^{\mathrm{T}} \end{bmatrix}, \qquad \texttt{So} = \begin{bmatrix} \tau_1^2 I_m & 0 \\ 0 & \tau_2^2 I_m \end{bmatrix}. \tag{1}$$

We then add these two together to obtain the matrix $\mathrm{cov}((Y_1^{\mathrm{T}}, Y_2^{\mathrm{T}})^{\mathrm{T}})$ which, recall that for this example is identical to $\mathrm{cov}((Z_1^{\mathrm{T}}, Z_2^{\mathrm{T}})^{\mathrm{T}})$ since the data is equal to the process at the observed locations. If `whole_mesh` is `TRUE`, then the process covariance matrix is evaluated over the entire mesh (used for co-kriging at unobserved locations).

```
construct_mats <- function(theta,model_num,whole_mesh=F) {

  nu1 <- theta[7]
  nu2 <- theta[8]

  B <- theta[9]*Diagonal(n1) # Automatically zero if Model 1

  if(model_num %in% c(3,4,7,8)) {
    B <- theta[9]*bisquare_B(h1_double,h2_double,
                             delta=theta[11:12], # Zero for Model with no shift
                             r=theta[10],
                             n1 = n1,
                             n2 = n2,
                             areas = areas)
  }
  C1B <- C1 %*% B

  ## Form matrices (scaled pressure)
  S11 <- makeS(r = Dobsvec,var = theta[3],
               kappa = theta[5],nu = nu1)
  S2_1 <- makeS(r = Dobsvec,var = theta[4],
                kappa = theta[6],nu = nu2)
  if(model_num %in% c(3,4,7,8) | whole_mesh==TRUE) {
    S11_big <- makeS(r = Dvec,var = theta[3],
                     kappa = theta[5],nu = nu1)
    S21 <- C1B %*% (S11_big %*% t(C1))
    S12 <- t(S21)
    S22 <- S2_1 + forceSymmetric(C1B %*% forceSymmetric(S11_big) %*% t(C1B))
  } else {
    S21 <- S12 <- theta[9]*S11
    S22 <- S2_1 + theta[9]^2 * S11
  }

  if(whole_mesh) {
    S11 <- S11_big
    S2_1 <- makeS(r = Dvec,var = theta[4],
                  kappa = theta[6],nu =nu2)
    S21 <- B %*% S11_big
    S12 <- t(S21)
```

```
    S22 <- S2_1 + Matrix::crossprod(chol(S11_big) %*% t(B))
  }

  ## Form matrices (Unscaled pressure)
  S11_true <- ifelse(model_num > 4,P_scale^2,1) * S11
  S12_true <- P_scale * S12
  S21_true <- P_scale * S21
  S22_true <-  ifelse(model_num < 5, P_scale^2,1) *S22
  SY_true <- rBind(cBind(S11_true,S12_true),
                   cBind(S21_true,S22_true)) %>% as("dgeMatrix")
  So_true <- bdiag(ifelse(model_num < 5,1,P_scale^2) * theta[1]*I_m1,
                   ifelse(model_num < 5,P_scale^2,1) * theta[2]*I_m1)
  if(whole_mesh) So_true <- t(C) %*% So_true %*% C

  list(SY = SY_true, So = So_true, Z = form_Z(model_num,scale=F))
}
```

Now we're in place to define the log-likelihood function. This is the usual Gaussian log-likelihood function. In the function we allow the dropping of certain observations for cross-validation purposes. The indices of the observations we wish to drop are stored in the parameter `i`. If `i = NULL` then no observation are dropped. This argument is useful for cross-validation.

```
loglik_Model <- function(theta,model_num,i=NULL) {
  # theta1:  sigma2e1
  # theta2:  sigma2e2
  # theta3:  sigma211
  # theta4:  sigma22_1
  # theta5:  kappa11
  # theta6:  kappa2_1
  # theta7:  nu11
  # theta8:  nu2_1
  # theta9:  A
  # theta10: r
  # theta11: d1
  # theta12: d2

  theta <- append_theta(theta,model_num)

  ## Hard constraints on parameters
  if(theta[1] <= 0 |  theta[2] <= 0 | theta[3] <= 0 |
     theta[4] <= 0 | theta[5] <= 0.001 | theta[6] <= 0.001 |
     theta[7] <= 0.05 | theta[8] <= 0.05 | theta[10] < 0.0005) {
       return(Inf)
     } else {

       ## Construct matrices
       X <- construct_mats(theta,model_num)

       ## Drop observations if required for CV
       if(is.null(i)) {
         SY <- X$SY
         So <- X$So
         Z <- X$Z
```

```
      } else {
        SY <- X$SY[-i,-i]
        So <- X$So[-i,-i]
        Z <- X$Z[-i,,drop=F]
      }

      ## Evaluate log-likelihood function
      cholYo <- chol(SY + So)
      loglik <-
        -(-0.5 * logdet(cholYo) -
            0.5 * t(Z) %*% chol2inv(cholYo) %*% Z -
            0.5 * nrow(Z)*log(2*pi)) %>% as.numeric()

      return(loglik)
    }
}
```

For optimising we will use the `R` function `optim` (BFGS). We allow for 3000 maximum iterations and set `trace=6` for detailed output. We choose to not compute the Hessian since this is not required in our analysis. Recall that the parameter `i` here contains the indices of the observations we do not wish to include in the fit. If `i = NULL` then all observations are inlcuded. The function `optim_loglik` is called for each model in the program later on.

```
optim_loglik <- function(par,model_num, i = NULL){
  optim(par=par,
        fn = loglik_Model,
        model_num=model_num,
        i=i,
        hessian=FALSE,
        control=list(trace=6,
                     pgtol=0,
                     parscale=rep(0.1,length(par)),
                     maxit=3000))

}
```

The last function we need to define is one that fits all the models, possibly with a set of observations in `i` removed. Below, in the function `fit_all_models`, we first fit Model 1 using realistic starting values and store the results in `fit.Model1`, and then fit the reversed version (with pressure as Y_1) and store that in `fit.Model1_rev`. Model 2 is then fit using the estimates of Model 1 as starting values. Model 3 uses the maximum likelihood estimates of Model 2, and so on. The reversed version of Model 2 uses the results of the reversed version of Model 1 as starting values and so on.

```
fit_all_models <- function(i) {
  fit.Model1     <- optim_loglik(par=c(0.01,1,5,15,0.01,0.01,0.6,1.5),model_num=1, i = i)
  fit.Model1_rev <- optim_loglik(par=c(1,0.01,15,5,0.01,0.01,1.5,0.6),model_num=5, i = i)
  fit.Model2     <- optim_loglik(par=c(fit.Model1$par,-0.2),    model_num=2, i = i)
  fit.Model2_rev <- optim_loglik(par=c(fit.Model1_rev$par,-0.2),model_num=6, i = i)
  fit.Model3     <- optim_loglik(par=c(fit.Model2$par,0.1),     model_num=3, i = i)
  fit.Model3_rev <- optim_loglik(par=c(fit.Model2_rev$par,0.1), model_num=7, i = i)
  fit.Model4     <- optim_loglik(par=c(fit.Model3$par,0,0),     model_num=4, i = i)
  fit.Model4_rev <- optim_loglik(par=c(fit.Model3_rev$par,0,0), model_num=8, i = i)
```

```
  list(Model1 = fit.Model1,
       Model2 = fit.Model2,
       Model3 = fit.Model3,
       Model4 = fit.Model4,
       Model5 = fit.Model1_rev,
       Model6 = fit.Model2_rev,
       Model7 = fit.Model3_rev,
       Model8 = fit.Model4_rev)
}
```

With all functions in place we now call `fit_all_data <- fit_all_models(NULL)`. All this does is fit all the models using all the observations (since `i = NULL`). If `LK_analysis = 1` then this is done from scratch (takes about 30 minutes), otherwise the data is loaded from cache.

```
## First we carry out the analysis with all data
if(LK_analysis) {
  fit_all_data <- fit_all_models(NULL)
  save(fit_all_data, file=paste0("../inst/extdata/temp_pressure/LK_fits.rda"))
} else {
  load(system.file("extdata/temp_pressure","LK_fits.rda", package = "bicon"))
}
```

The log-likelihoods and AICs given by our fit are given in the table below. Note that Model 5 is Model 1 reversed (i.e., with pressure as Y_1), Model 6 is Model 2 reversed, and so on.

```
print("Log-likelihood for all models trained with complete dataset")
```

```
## [1] "Log-likelihood for all models trained with complete dataset"
```

```
sapply(fit_all_data,function(x) x$value) ## Negative LL
```

```
##   Model1   Model2   Model3   Model4   Model5   Model6   Model7   Model8
## 1276.770 1269.922 1264.901 1258.212 1276.770 1266.826 1268.983 1268.486
```

```
sapply(fit_all_data,function(x) x$value)*2 + 2*c(8,9,10,12) ## AIC
```

```
##   Model1   Model2   Model3   Model4   Model5   Model6   Model7   Model8
## 2569.541 2557.844 2549.803 2540.425 2569.541 2551.651 2557.967 2560.972
```

The parameters are listed below (the output is in LaTeX for direct use in paper). Note that since `P_scale` was used to put pressure on the same scale as temperature, we scale the fitted marginal standard deviation of the pressure fields so that they are on the original scale.

```
print("Estimated parameters for all models")
par_est <- plyr::rbind.fill(sapply(fit_all_data,function(x) data.frame(t(x$par))))
par_est1 <- par_est[1:4,]
par_est1[,c(1,3)] <- sqrt(par_est1[,c(1,3)])
par_est1[,c(2,4)] <- sqrt(par_est1[,c(2,4)]) * P_scale
par_est1[,9] <- par_est1[,9] * P_scale
```

```
colnames(par_est1) <- c("$\\sigma_1$","$\\sigma_2$","$\\sigma_{11}$",
                        "$\\sigma_{2|1}$","$\\kappa_{11}$","$\\kappa_{2|1}$",
                        "$\\nu_{11}$","$\\nu_{2|1}$",
                        "$A$","$r$", "$\\Delta_1$","$\\Delta_2$")
rownames(par_est1) <- c("Model 1","Model 2","Model 3","Model 4")
print(xtable::xtable(par_est1,digits=c(rep(2,5),3,3,rep(2,6))),
      sanitize.text.function=function(x){x},
      hline.after=NULL)
```

Prediction

We predict the temperature and pressure fields at the unobserved locations using cokriging. Since we assume zero mean, this is simple cokriging; the predictive mean and variance can thus be obtained by simple conditioning with a joint multivariate Gaussian distribution. If `i = NULL` then the data is used to predict at all (observed and unobserved) locations. Otherwise prediction is only carried out at the locations in i with the observations in i removed. Note that when `i` is specified it is assumed that only the covariance matrices associated with the observation locations are supplied. This enables us to use the same function for cross-validation (see below).

```
cokrige <- function(X,i=NULL) {
  SS <- X$SY + X$So

  if(is.null(i)) {
    Z <- X$Z
    Q <- chol2inv(chol(C%*% SS %*% t(C))) %>% as("dgeMatrix")
    mu_pred  <- SS%*% t(C) %*% Q %*% Z %>% as.numeric()
    var_pred <- diag(SS - SS %*% t(C) %*% Q %*% C %*% SS) %>% as.numeric()
    data.frame(mu_pred = mu_pred,
               var_pred = var_pred)
  } else {
    cholSS <- chol(SS[-i,-i]) ## this was SS[-i,-i]
    SSinv <- chol2inv(cholSS) %>% as("dgeMatrix")

    mu_pred  <- SS[i,-i] %*% SSinv %*%  X$Z[-i,,drop=FALSE] %>% as.numeric()
    var_pred <- diag(SS[i,i] - SS[i,-i] %*% SSinv %*% SS[-i,i]) %>% as.numeric()
    data.frame(mu_pred = mu_pred,
               var_pred = var_pred,
               Z = X$Z[i,],
               i=i)
  }

}
```

Below we predict at all the mesh locations using Model 1 and Model 4. First we construct the required matrices and store them in `X1` and `X4`. Then we carry out cokriging and add the mean predictions to the mesh.

```
X1 <- construct_mats(theta = append_theta(fit_all_data[[1]]$par,model_num = 1),
                     model_num = 1,whole_mesh = T)
X4 <- construct_mats(theta = append_theta(fit_all_data[[4]]$par,model_num = 4),
                     model_num = 4,whole_mesh = T)
ALL1 <- cokrige(X=X1,i=NULL)
```

```
ALL4 <- cokrige(X=X4,i=NULL)

Mesh["y1_Model1"] <- ALL1$mu_pred[(1:n1)]
Mesh["y1_Model4"] <- ALL4$mu_pred[(1:n1)]
Mesh["y2_Model1"] <- ALL1$mu_pred[-(1:n1)]
Mesh["y2_Model4"] <- ALL4$mu_pred[-(1:n1)]
```

Leave-one-out cross-validation

Unlike Gneiting et al. (2010), here we carry out leave-one-out cross validation (LOOCV) without re-fitting the model each time. If we have an MPI cluster available we carry out the LOOCV over MPI, otherwise we parallelise using the machine's multiple cores. The two loops below iterate over the observations and models.

```
## Now we do a LOO analysis
if(LOO_analysis) {

  if(useMPI) {
    library(doMPI)
    cl <- startMPIcluster(count=80)
    registerDoMPI(cl)
  } else {
    library(doParallel)
    cl <- makePSOCKcluster(4,outfile="cores_output.txt")
    registerDoParallel(cl)
  }

  ## Loop over each observation location
  pred <- foreach(i = 1:m1,.combine = "rbind",
                  .packages = c("Matrix","bicon","dplyr","foreach")) %dopar% {

    fit.Model <- fit_all_data

    ## Loop over each model (not parallelised)
    pred <- foreach(j = seq_along(fit.Model),.combine = "rbind") %do% {

      ## Construct matrices
      X <- construct_mats(theta = append_theta(fit.Model[[j]]$par,model_num = j),
                          model_num = j)

      ## Cokrige, leaving out the ith observation (for both temperature and pressure)
      cbind(cokrige(X=X,i=c(i,i+m1)),
            model_num = j)
    }
    pred
  }

  if(useMPI) {
    closeCluster(cl)
  } else {
    stopCluster(cl)
  }
```

```
  ### Should only be run with path set as vignette source directory
  save(pred, file=paste0("../inst/extdata/temp_pressure/all_predictions.rda"))
}
```

If we decided not to run the LOOCV, we load the results from cache.

```
if(!LOO_analysis) {
  load(system.file("extdata/temp_pressure/all_predictions.rda", package = "bicon"))
}
```

The shifted parsimonious Matérn model

In this section we repeat the above analysis for the *shifted* parsimonious Matérn model, obtained by applying the method of Li & Zhang (2011) to the standard parsimonious model. First we define a function that constructs the matrices based on the usual parameters.

```
sh_pars_mats <- function(theta) {
    ## Now create the shifted locations for the cross-covariances
    new_locs <- weather[,3:4] + matrix(theta[1:2],ncol=2,nrow=nrow(weather),byrow=TRUE)

    X <- rbind(new_locs,weather[,3:4])
    tot_D <- as.matrix(RFearth2dist(as.matrix(X)))
    D12 <- tot_D[-(1:m1),1:m1]
    D21 <- t(D12)

    sigma2_21 <- sqrt(theta[9] * theta[10])*theta[6]
    SY11 <- makeS(r=Dobsvec,var= theta[9],kappa = theta[3],nu=theta[4])
    SY22 <- makeS(r=Dobsvec,var= theta[10],kappa = theta[3],nu=theta[5])
    SY12 <- makeS(r=c(D21),var=sigma2_21, kappa = theta[3], nu = (theta[4] + theta[5])/2)
    SY21 <- t(SY12)
    So11 <- theta[7]^2 * diag(m1)
    So22 <- theta[8]^2 * diag(m1)
    SY <- cbind(rbind(SY11,SY12),rbind(SY21,SY22))
    So <- bdiag(So11,So22)
    list(SY = SY, So = So,Z = form_Z(1L,scale=F))
}
```

Next, we define the likelihood funciton; note that the parameter definitions for the parsimonious Matérn are different than for the conditional approach:

```
loglik_sh_pars_model <- function(theta,shift=FALSE) {
  # theta1:  delta1
  # theta2:  delta2
  # theta3:  kappa
  # theta4:  nu1
  # theta5:  nu2
  # theta6:  rho
  # theta7:  tau1
  # theta8:  tau2/100
  # theta9:  sigma2_11
  # theta10: sigma2_22/10000
  if(theta[3] < 0.00001 | theta[4] < 0.1 | theta[5] < 0.1 |
```

```
    abs(theta[6]) >= 1 | theta[7] <= 0 | theta[8] <= 0 |
    theta[9] <= 0 | theta[10] <= 0) {
      return(Inf)
    } else  if (abs(theta[6]) > sqrt(theta[4] * theta[5]) / 0.5*(theta[4] + theta[5]))
    {
      return(Inf)
    }  else {

       if(!shift) {
           theta[1] <- theta[2] <- 0
       }

      theta[8] <- theta[8] * 100    ## These were divided by 100 and 1000
      theta[10] <- theta[10] * 10000  ## in the initial call, respectively

      ## Get the data
      Z = form_Z(1L,scale=F)

      ## Get the matrices
      Matrices <- sh_pars_mats(theta)
      S <- Matrices$SY + Matrices$So
      cholS <- chol(S)

      loglik <-
        -as.numeric(-0.5 * determinant(S)$modulus -
                      0.5 * t(Z) %*% chol2inv(cholS) %*% Z -
                      0.5 * nrow(Z)*log(2*pi))

      return(loglik)
    }
}
```

Finally, we estimate the parameters:

```
## Set shift=TRUE below to estimate the shift parameters; otherwise they are fixed to zero
## and we get the same likelihood and parameter estimates of Gneiting (2010)
if(Shifted_Pars_estimation) {
    optim_est_sh_pars <- optim(c(0.5,-1,1/95.88,
                      0.6,1.6,-0.5,
                      0.019,69.66/100,
                      6.95,
                      67191/10000),
                    loglik_sh_pars_model,
                    control=list(trace=6,
                                 maxit=3000),
                    shift=TRUE)
    save(optim_est_sh_pars,file = "../inst/extdata/temp_pressure/Shifted_Pars_est_results.rda")
} else {
    load(system.file("extdata/temp_pressure","Shifted_Pars_est_results.rda", package = "bicon"))
}


print(paste0("Log-lik. of shifted parsimonious Matern is ",
```

```
            optim_est_sh_pars$value))
```

```
## [1] "Log-lik. of shifted parsimonious Matern is 1260.87294870203"
```

```
print(paste0("AIC of shifted parsimonious Matern is ",
            optim_est_sh_pars$value*2 + 20)) # 10 parameters
```

```
## [1] "AIC of shifted parsimonious Matern is 2541.74589740405"
```

For co-kriging (for cross-validation) we follow the same approach as earlier:

```
theta_lk <- optim_est_sh_pars$par
theta_lk[8] <- theta_lk[8] * 100    ## These were divided by 100 and 1000
theta_lk[10] <- theta_lk[10] * 10000  ## in the initial call, respectively
X <- sh_pars_mats(theta = theta_lk)

## Loop over each observation location
print("Running CV for shifted parsimonious Matern...")
```

```
## [1] "Running CV for shifted parsimonious Matern..."
```

```
pred_sh_pars <- foreach(i = 1:m1,.combine = "rbind") %do% {
                X <- sh_pars_mats(theta = theta_lk)
                cbind(cokrige(X=X,i=c(i,i+m1)),
                              model_num = 1)}
```

Curing of results

The following code is only documented in-line since it just involves data manipulation for obtaining the results shown in the paper. For verification we find the mean absolute error (MAE), the root mean-squared prediction error (RMSPE) and the continuous probability rank score (CRPS) as described by Gneiting, Raftery, Westveld III, Anton, & Goldman (2005).

```
### Analyse results

## put data set into long format
weather_long <- mutate(weather,loc_num = 1:nrow(weather)) %>%
  mutate(sum_D = apply(Dobs,1,function(x) sum(sort(x)[1:2]))) %>%
  gather(process,z,temperature,pressure,convert = TRUE)

## Utility wrapper around the crps function
crps_wrapper <- function(Z,mu,sd) {
  crps(Z,cbind(mu,sd))$crps
}

sanitise_results <- function(pred) {
    ## put data set into long format
    pred2 <- mutate(pred,                                          # Take our LOOCV results
                process = ifelse((model_num < 5 & i <= m1) |  # Assign process name to row
                                 (model_num >=5 & i >  m1) ,
```

```
                                "temperature",
                                "pressure"),
                loc_num = ((i-1) %% m1)+1) %>%                # Assign loc ID
  left_join(weather_long) %>%                                 # Join up with data
  dplyr::select(-i,-z)                                        # Remove unwanted columns

    results <- pred2 %>%                                    # Take the predictions
    group_by(process,model_num) %>%                         # Group by process and model
    summarise(MAE = mean(abs(mu_pred - Z)),                 # Find MAE
              MAE_se = sd(mu_pred - Z)/sqrt((m1-1)),        # Find MAE standard error
              Bias = mean(mu_pred - Z),                     # Find mean bias
              Bias_se = sd(mu_pred - Z)/sqrt((m1-1)),       # Find bias standard error
              Bias_norm = mean((mu_pred - Z)/sqrt(var_pred)), # Find mean normalised bias
              RMSPE = sqrt(mean((mu_pred - Z)^2)),          # Find RMSPE
              CRPS = mean(crps_wrapper(Z,mu_pred,sqrt(var_pred))),   # Find CRPS
              CRPS_se = sd(crps_wrapper(Z,mu_pred,sqrt(var_pred)))/  # Find CRPS se
                sqrt(m1-1))

    results
}
results <- sanitise_results(pred)
print(results)
```

```
## Source: local data frame [16 x 10]
## Groups: process [?]
##
##        process model_num       MAE    MAE_se         Bias   Bias_se
##          <chr>     <int>     <dbl>     <dbl>        <dbl>     <dbl>
## 1     pressure         1 69.557498 9.8831976 -8.723436484 9.8831976
## 2     pressure         2 70.190275 9.9657301 -9.135887569 9.9657301
## 3     pressure         3 70.317490 9.8578626 -8.036973575 9.8578626
## 4     pressure         4 66.068990 9.2041738 -4.212511072 9.2041738
## 5     pressure         5 69.557498 9.8831976 -8.723436484 9.8831976
## 6     pressure         6 67.020147 9.8313416 -4.821590904 9.8313416
## 7     pressure         7 66.808978 9.7410402 -5.364313591 9.7410402
## 8     pressure         8 66.577363 9.6881394 -5.095728792 9.6881394
## 9  temperature         1  1.143941 0.1305158  0.040281435 0.1305158
## 10 temperature         2  1.143934 0.1305225  0.040190622 0.1305225
## 11 temperature         3  1.095204 0.1228428  0.037770689 0.1228428
## 12 temperature         4  1.080266 0.1176788  0.017517495 0.1176788
## 13 temperature         5  1.143941 0.1305158  0.040281435 0.1305158
## 14 temperature         6  1.118945 0.1272823 -0.003432577 0.1272823
## 15 temperature         7  1.104627 0.1255866 -0.024560013 0.1255866
## 16 temperature         8  1.101850 0.1255400 -0.023512694 0.1255400
## # ... with 4 more variables: Bias_norm <dbl>, RMSPE <dbl>, CRPS <dbl>,
## #   CRPS_se <dbl>
```

We next do the same for the shifted parsimonious Matérn model.

```
## Extract our results
sh_pars_results <- sanitise_results(pred_sh_pars) %>%   # Only consider models with Y1 temp.
                   dplyr::select(model_num,MAE,RMSPE,CRPS)
```

```
## Joining, by = c("process", "loc_num")

## Adding missing grouping variables: `process`
```

```
sh_pars_results$model_num <- "Shifted_Pars"
```

For completeness we also carry out LOOCV on the (unshifted) parsimonious and full Matérn models using the `RandomFields` package. This closely follows the approach illustrated in M. Schlather, Malinowski, Menck, Oesting, & Strokorb (2015).

```
Dist.mat <- as.vector(RFearth2dist(as.matrix(weather[, 3:4])))   # Compute distances
PT <- as.matrix(weather[, 1:2])                                  # Change data into matrix
## Note the below code only works with RandomFields v3.0.62
if(RF_estimation) {

  ## Parsimonious Matern model
  nug <- RMmatrix(M = matrix(nc = 2, c(NA, 0, 0, NA)), RMnugget())   # nugget model
  pars.model <- nug + RMbiwm(nudiag = c(NA, NA), scale = NA,         # parsimonious model
                             cdiag = c(NA, NA), rhored = NA)
  RFpars <- RFfit(pars.model, distances = Dist.mat, dim = 3, data = PT)  # fit model
  print(RFpars)
  CVresults.pars <- RFcrossvalidate(RFpars,              # carry out CV
                       x = as.matrix(weather[, 3:4]),
                       data = PT,
                       full = TRUE)                      # on all the data (no re-fitting)

  ## Full Matern model
  whole.model <- nug + RMbiwm(nudiag = c(NA, NA), nured = NA,
                              s = rep(NA, 3), cdiag = c(NA, NA), rhored = NA)
  RFwhole <- RFfit(whole.model, distances = Dist.mat, dim = 3, data = PT)
  CVresults.whole <- RFcrossvalidate(RFwhole,
                       x = as.matrix(weather[, 3:4]),
                       data = PT,
                       full = TRUE)

  ## cache results
  save(CVresults.pars,RFpars,CVresults.whole,RFwhole,
       file = "../inst/extdata/temp_pressure/RF_CV_results.rda")

} else {
  ## otherwise load
  load(system.file("extdata/temp_pressure","RF_CV_results.rda", package = "bicon"))
}
```

```
## Combine results into one long data frame
RFpred2 <- rbind(data.frame(Z = weather$pressure,
                            mu_pred = CVresults.pars$`user's model`$predicted[,1],
                            var_pred = CVresults.pars$`user's model`$krige.var[,1],
                            loc_num = 1:m1,
                            model_name = "Pars",
                            process = "pressure"),
                 data.frame(Z = weather$temperature,
                            mu_pred = CVresults.pars$`user's model`$predicted[,2],
```

```
                              var_pred = CVresults.pars$`user's model`$krige.var[,2],
                              loc_num = 1:m1,
                              model_name = "Pars",
                              process = "temperature"),
                    data.frame(Z = weather$pressure,
                              mu_pred = CVresults.whole$`user's model`$predicted[,1],
                              var_pred = CVresults.whole$`user's model`$krige.var[,1],
                              loc_num = 1:m1,
                              model_name = "Whole",
                              process = "pressure"),
                    data.frame(Z = weather$temperature,
                              mu_pred = CVresults.whole$`user's model`$predicted[,2],
                              var_pred = CVresults.whole$`user's model`$krige.var[,2],
                              loc_num = 1:m1,
                              model_name = "Whole",
                              process = "temperature")) %>%
            left_join(weather_long) %>%
            dplyr::select(-z)

## Get out the diagnostics
RFresults <- RFpred2 %>%
  group_by(process,model_name) %>%  # Group by process and model name
  summarise(MAE = mean(abs(mu_pred - Z)),
            MAE_se = sd(mu_pred - Z)/sqrt((m1-1)),
            Bias = mean(mu_pred - Z),
            Bias_se = sd(mu_pred - Z)/sqrt((m1-1)),
            Bias_norm = mean((mu_pred - Z)/sqrt(var_pred)),
            RMSPE = sqrt(mean((mu_pred - Z)^2)),
            CRPS = mean(crps_wrapper(Z,mu_pred,sqrt(var_pred))),
            CRPS_se = sd(crps_wrapper(Z,mu_pred,sqrt(var_pred)))/sqrt(m1-1))

## Extract our results
select_results <- filter(results,model_num < 5) %>%          # Only consider models with Y1 temp.
                  dplyr::select(model_num,MAE,RMSPE,CRPS)

## Extract RandomFields results
select_resultsRF <- dplyr::select(RFresults,model_name,MAE,RMSPE,CRPS)

## Relabel columns
colnames(select_results) <- colnames(select_resultsRF) <- colnames(sh_pars_results) <-
    c("Process","Model","MAE","RMSPE","CRPS")


## Join our results with those from RandomFields and the shifted parsimonious Matern
all_results <- rbind.data.frame(select_results,select_resultsRF,sh_pars_results) %>%
    as.data.frame() %>% arrange(Process)

## Print the LaTeX table
print(xtable::xtable(all_results,digits=3),
      each = "column", max = c(F,NA,NA,T,NA),
      sanitize.text.function=function(x){x},
      hline.after=NULL,include.rownames=FALSE)
```

```
## % latex table generated in R 3.2.0 by xtable 1.8-2 package
## % Mon Sep 26 12:14:26 2016
## \begin{table}[ht]
## \centering
## \begin{tabular}{llrrr}
##  Process & Model & MAE & RMSPE & CRPS \\
##  pressure & 1 & 69.557 & 123.356 & 55.327 \\
##   pressure & 2 & 70.190 & 124.411 & 55.640 \\
##   pressure & 3 & 70.317 & 122.995 & 55.187 \\
##   pressure & 4 & 66.069 & 114.671 & 51.725 \\
##   pressure & Pars & 70.150 & 122.970 & 55.349 \\
##   pressure & Whole & 66.189 & 122.758 & 55.225 \\
##   pressure & Shifted_Pars & 67.009 & 114.978 & 52.479 \\
##   temperature & 1 & 1.144 & 1.625 & 0.813 \\
##   temperature & 2 & 1.144 & 1.626 & 0.814 \\
##   temperature & 3 & 1.095 & 1.530 & 0.780 \\
##   temperature & 4 & 1.080 & 1.465 & 0.767 \\
##   temperature & Pars & 1.110 & 1.562 & 0.790 \\
##   temperature & Whole & 1.109 & 1.576 & 0.792 \\
##   temperature & Shifted_Pars & 1.091 & 1.478 & 0.771 \\
##   \end{tabular}
## \end{table}
```

Plotting

The rest of the code is devoted to plotting. Since this is terse, we do not discuss it. It relies on knowledge of the packages `ggplot2`, `dplyr`, and `tidyr`, the latter needed for putting the data into an appropriate format.

```
# Load shapefiles
shape1 <- system.file("extdata", "cb_2013_us_state_5m.shp", package = "bicon")
shape2 <- system.file("extdata", "Canada_provinces.SHP", package = "bicon")
US_States <- maptools::readShapeSpatial(shape1)
Ca_States <- maptools::readShapeSpatial(shape2)

US_States_fort <- fortify(US_States) %>% mutate( id= as.numeric(id))
Ca_States_fort <- fortify(Ca_States) %>% mutate( id= as.numeric(id)+100)

US_State_names <- data.frame(Name = US_States$NAME, id = 0:(length(US_States$NAME)-1))
Ca_State_names <- data.frame(Name = Ca_States$NAME, id = 0:(length(Ca_States$NAME)-1) + 100)

US_States_fort <- left_join(US_States_fort,US_State_names,by="id")
Ca_States_fort <- left_join(Ca_States_fort,Ca_State_names,by="id")

All_States <- rbind(US_States_fort,Ca_States_fort) %>%
  filter(Name %in% c("British Columbia","Alberta",
                     "Washington","Oregon",
                     "Idaho","California",
                     "Nevada","Montana"))

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

Stateplot <- LinePlotTheme() +
```

```
  geom_path(data=All_States,aes(long,lat,group=group,label=Name),linetype="solid") +
  coord_fixed(xlim=c(-137,-110),ylim=c(35,58)) +
  geom_path(data=conv_hull,aes(x,y),size=1,colour="black",linetype="dashed") +
  theme(plot.margin = grid::unit(c(2, 2, 2, 2),units="mm")) + xlab("lon")

meshplot <- function(g,include_obs=1L) {
  p <- plot(Mesh,g=g,plot_dots=F)
  p <- p +
    xlab('lon') + ylab('lat') +
    coord_fixed(xlim=c(-137,-110),ylim=c(35,58)) +
    geom_path(data=All_States,aes(long,lat,group=group,label=Name),linetype="solid") +
    geom_path(data=conv_hull,aes(x,y),size=1,colour="black",linetype="dashed")
  if(include_obs) p <- p + geom_point(data = weather,aes(lon,lat),size=3,col="red")
  p
}

if(show_figs) {
  States_mesh <- meshplot(LinePlotTheme())
  print(States_mesh)
}
```

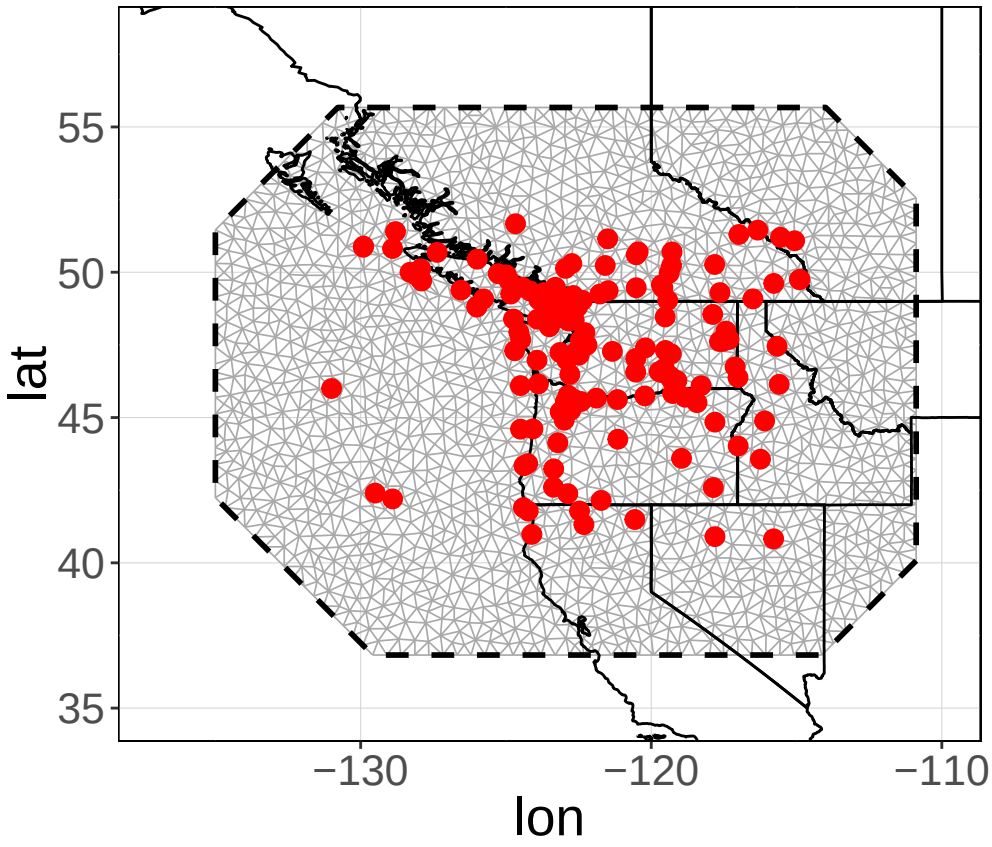

Fig. 1: State boundaries and province boundaries of a region of the U.S.A. and Canada (dark solid lines), with the domain of interest enclosed by a bounding polygon (dashed line). The irregular triangular grid used for discretizing D (light solid lines) and the observation locations given by D^O (dots) are also shown. The discretized spatial domain D^L consists of the vertices of the triangular grid.

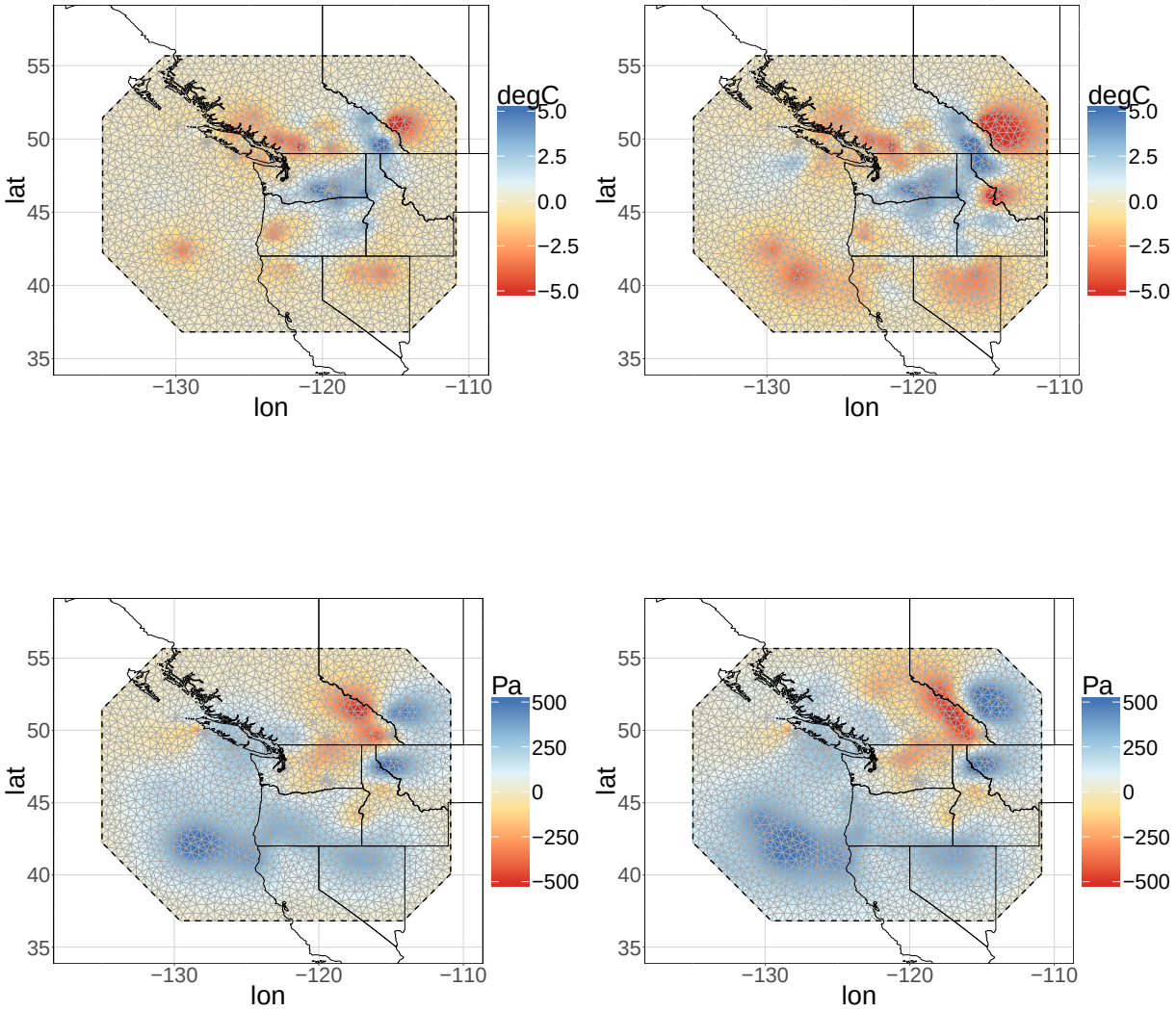

Fig. 2: The cokriged surface using maximum likelihood estimates for the parameters with Model 1 (left panels) and Model 4 (right panels) for the temperature (top panels) and pressure (bottom panels) error fields.

```
g1M1 <- plot_interp(Mesh,"y1_Model1",150,max=5,min=-5,leg_title="degC") %>%
  meshplot(include_obs = 0L)
g1M4 <- plot_interp(Mesh,"y1_Model4",150,max=5,min=-5,leg_title="degC") %>%
  meshplot(include_obs = 0L)

g2M1 <- plot_interp(Mesh,"y2_Model1",150,max=500,min=-500,leg_title="Pa") %>%
  meshplot(include_obs = 0L)
g2M4 <- plot_interp(Mesh,"y2_Model4",150,max=500,min=-500,leg_title="Pa") %>%
  meshplot(include_obs = 0L)
if(show_figs) {
  grid.arrange(g1M1,g1M4,g2M1,g2M4,ncol=2)
}
```

```
par1 <- fit_all_data[[4]]$par
par2 <- fit_all_data[[8]]$par
x <- seq(-3,3,length=100)
```

```
XY <- expand.grid(h1=x,h2=x)
XY$b1 <- bisquare_2d(h1=XY[,1],h2=XY[,2],delta=par1[11:12],r=par1[10],A=par1[9])
XY$b2 <- bisquare_2d(h1=XY[,1],h2=XY[,2],delta=par2[11:12],r=par2[10],A=par2[9])

axes <- geom_line(data=data.frame(x=c(0,0,-3,3),y=c(-3,3,0,0),grp=c(1,1,2,2)),
                  aes(x,y,group=grp),colour="black")
library(scales)
greys <- scale_fill_gradient(low="black",high="white")
bluered <- scale_fill_gradient2(low=muted("blue"),high=muted("red"))
g1 <- LinePlotTheme() + geom_tile(data=XY,aes(h1,h2,fill=b1)) +
  greys + axes + coord_fixed() +ggtitle("Model 4")
g2 <- LinePlotTheme() + geom_tile(data=XY,aes(h1,h2,fill=b2)) +
  axes + greys + coord_fixed() +ggtitle("Model 8")
if(show_figs) grid.arrange(g1,g2,nrow=1)
```

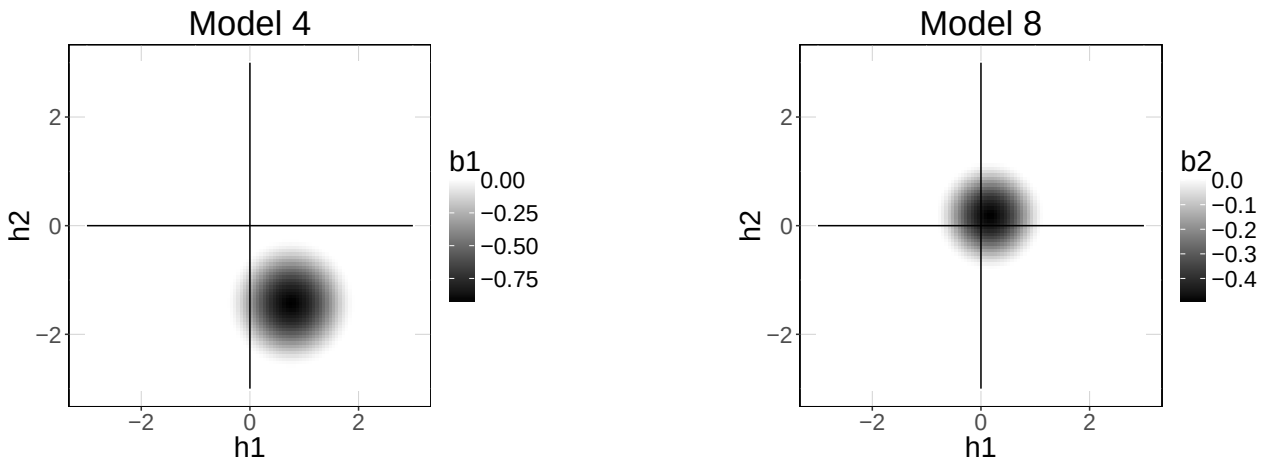

Fig. 3: Cross-covariance functions for Model 4 and Model 8 (that is, Model 4 with temperature and pressure reversed)

```
h1_grid<- seq(-4,4,by=0.2)
h2_grid <- seq(-4,4,by=0.2)
Disp <- expand.grid(h1 = h1_grid,h2 = h2_grid)
xo <- seq(-127,-113,by=0.2)
yo <- seq(40,55,by=0.2)

doMC::registerDoMC(6)
Disp$corr <- foreach(i = 1:nrow(Disp),.combine="c") %dopar% {
  Temp <- akima::interp(mesh_locs[,1]-Disp$h1[i],mesh_locs[,2]-Disp$h2[i],
                        Mesh["y1_Model1"],xo,yo)
  Pres <- akima::interp(mesh_locs[,1],mesh_locs[,2],
                        Mesh["y2_Model1"],xo,yo)
  cor(c(Pres$z),c(Temp$z),"na.or.complete")
}
axes <- geom_line(data=data.frame(x=c(0,0,-4,4),y=c(-4,4,0,0),grp=c(1,1,2,2)),
                  aes(x,y,group=grp),colour="black")
corr_plot <- LinePlotTheme() + geom_tile(data=Disp,aes(h1,h2,fill=corr)) + axes + bluered +
                           geom_point(data=data.frame(d1 = par1[11],d2 = par1[12]),
                                      aes(d1,d2),pch=9,size=5,colour="yellow") +
```

```
                        coord_fixed(xlim=c(-2.5,2.5),ylim=c(-2.5,2.5))

if(show_figs) print(corr_plot)
```

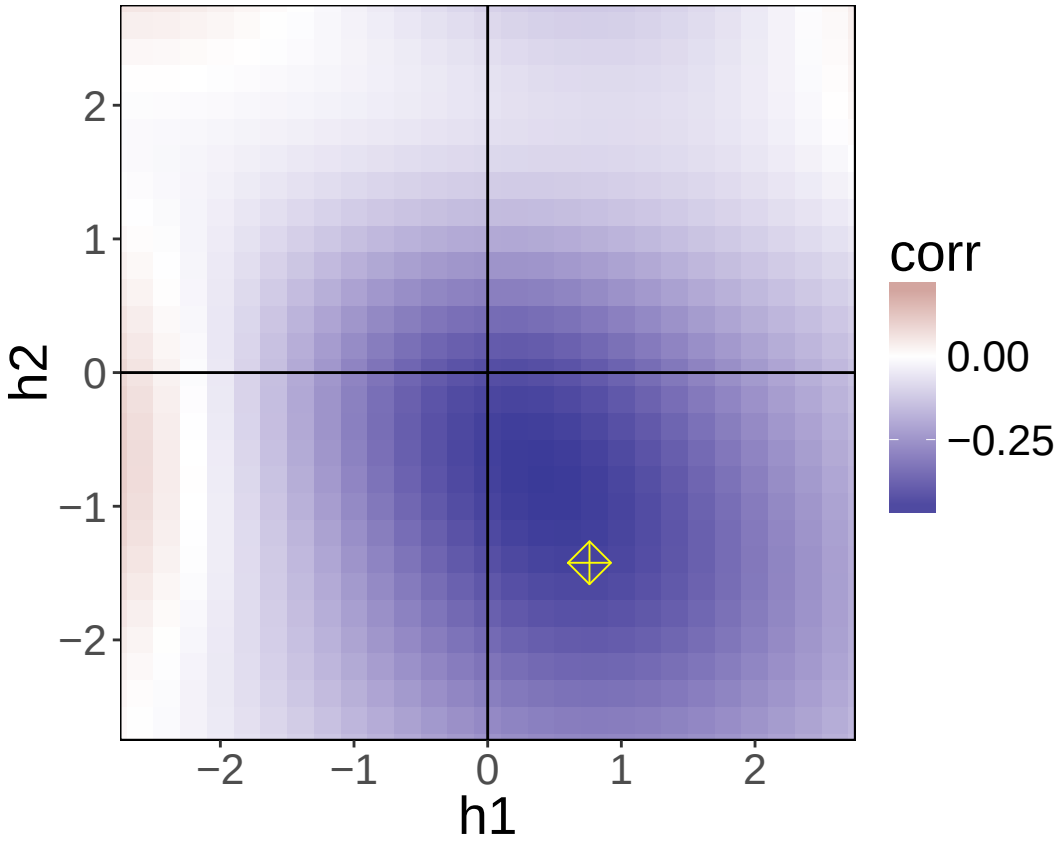

Fig. 4: Correlation between interpolated gridded maps of temperature and pressure as a function of displacement of the temperature field in latitude/longitude degrees, h. The yellow symbol indicates the maximum-likelihood estimate of the shift parameter with Model 4.

```
if(print_figs) ggsave(corr_plot,
                      filename = file.path(img_path,"T-P-corr.png"),
                      width=8,height=7,family="Arial")
```

```
delta <- 0.25
x_grid<- seq(-131,-115,by=delta)
y_grid <- seq(40,50,by=delta)
grid_locs <- expand.grid(x = x_grid, y = y_grid) %>% as.matrix()
D_grid <- as.matrix(RFearth2dist(grid_locs))
Dvec_grid <- as.double(c(D_grid))

n2_grid <- n1_grid <- nrow(grid_locs)
h_grid <- matrix(0,n1_grid^2,2)
areas_grid <- rep(delta^2,n1_grid^2)
for(i in 1:n2_grid) {
  h_grid[((i-1)*n1_grid+1):(i*n1_grid),] <- t(t(grid_locs) -grid_locs[i,])
}
```

```
theta = fit_all_data$Model4$par
B <- theta[9]*bisquare_B(h_grid[,1],h_grid[,2],
                         delta=theta[11:12], # Automatically zero for Model with no shift
                         r=theta[10],
                         n1 = n1_grid,
                         n2 = n2_grid,
                         areas = areas_grid)

  S11 <- makeS(r = Dvec_grid,var = theta[3],
               kappa = theta[5],nu = theta[7])
  S11 <- S11 + theta[1]^2*diag(nrow(S11))
  S2_1 <- makeS(r = Dvec_grid, var = theta[4],
                kappa = theta[6],nu = theta[8])
  S21 <- B %*% S11
  S12 <- t(S21)
  S22 <- S2_1 + Matrix::crossprod(chol(S11) %*% t(B))
  S22 <- S22 + theta[2]^2*diag(nrow(S22))

centre_node <- which(grid_locs[,1] == -123 & grid_locs[,2] == 45)
h_centre <- t(t(grid_locs) -c(-123,45))
H <- data.frame(h1 = h_centre[,1],
                h2 = h_centre[,2],
                S11 = (S11)[centre_node,],
                S12 = (S12)[centre_node,],
                S21 = (S21)[centre_node,],
                S22 = (S22)[centre_node,]) %>%
    gather(cov_mat,C,-h1,-h2) %>%
    mutate(Cgrp1 = ifelse(cov_mat %in% c("S11","S12"),"Y1","Y2"),
           Cgrp2 = ifelse(cov_mat %in% c("S11","S21"),"Y1","Y2")) %>%
    group_by(cov_mat) %>%
    mutate(corr = C / max(abs(C)))

axes <- geom_line(data=data.frame(x=c(0,0,-5,5),y=c(-5,5,0,0),grp=c(1,1,2,2)),
                  aes(x,y,group=grp),colour="black")
corr_fn_plot <- LinePlotTheme() + geom_tile(data=H,aes(h1,h2,fill=corr)) +
    geom_contour(data=H,aes(h1,h2,z=corr),binwidth=0.2,colour="black",lty="dashed") +
          facet_grid(Cgrp1~Cgrp2) + axes +
          scale_fill_gradient2(low="blue",high="red") +
  coord_fixed(xlim=c(-4,4),ylim=c(-4.2,4.2))  +
    theme(panel.margin = grid::unit(3, "lines"))

if(show_figs) print(corr_fn_plot)
```

```
if(print_figs) ggsave(corr_fn_plot,
                      filename = file.path(img_path,"T-P-cov.png"),
                      width=8,height=7,family="Arial")

if(print_figs) {
  g <- arrangeGrob(States_mesh,corr_fn_plot,ncol=2)
  ggsave(g,
         filename = file.path(img_path,"Fig2.eps"),
         width=14,height=6,family="Arial")
```

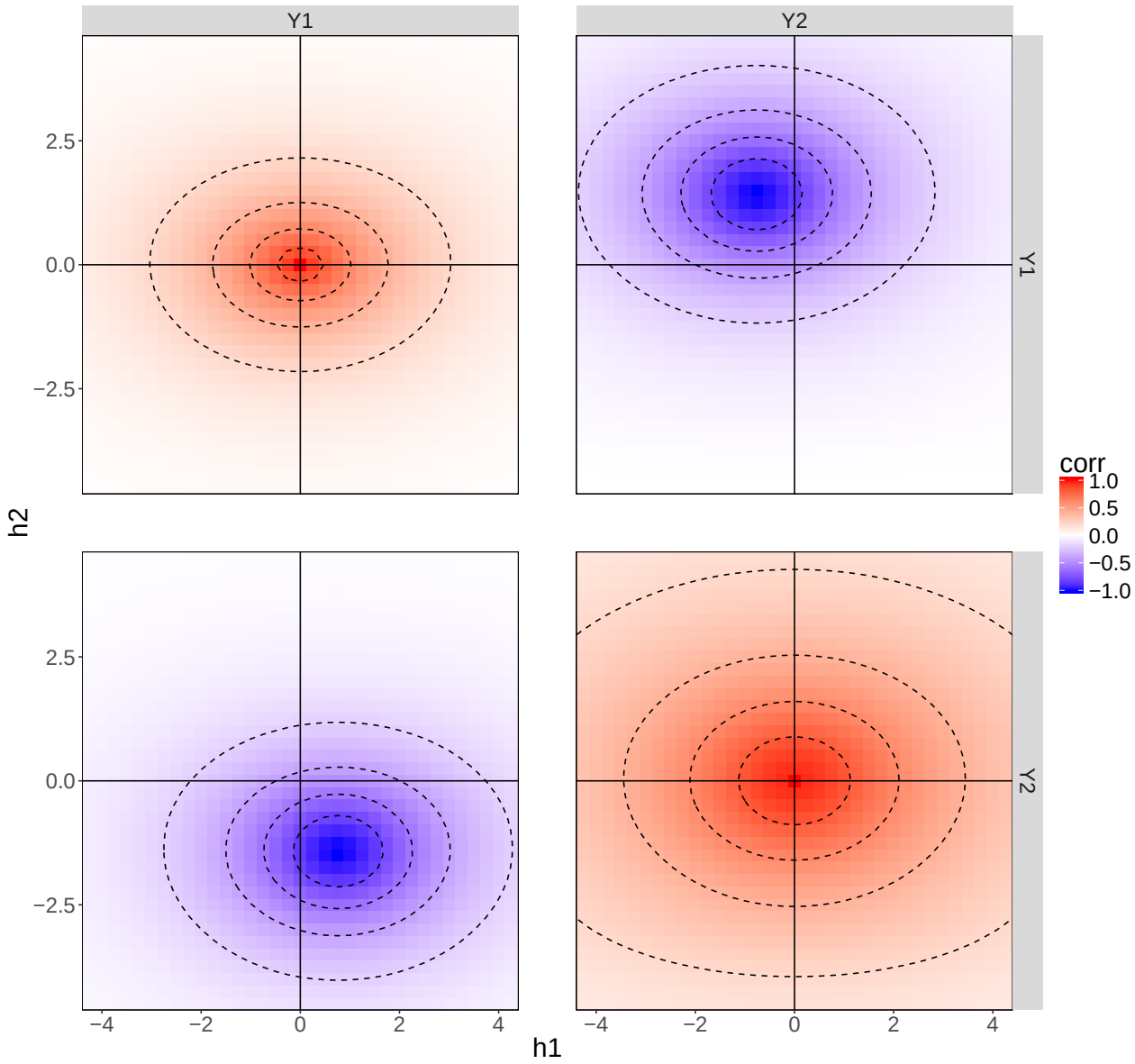

Fig. 5: The correlation and cross-correlation functions estimated using Model 4, depicted as a function of displacement h in degrees longitude/latitude, at the location $s = (-123°, 45°)$. Contour lines of correlation are in intervals of 0.2.

```
}
```

```
Mesh["M1T_errors"] <- sqrt(pmax(ALL1$var_pred,0))[1:n1]
Mesh["M4T_errors"] <- sqrt(pmax(ALL4$var_pred,0))[1:n1]
Mesh["diffT_errors"] <- Mesh["M1T_errors"] - Mesh["M4T_errors"]
Mesh["ratioT_errors"] <- Mesh["M1T_errors"]/pmax(Mesh["M4T_errors"],1e-9)
std_diff <- (plot_interp(Mesh,"diffT_errors",150) +
               scale_fill_gradient2(low=muted("yellow"),
                                    mid="white",
                                    high=muted("magenta"),
                                    guide = guide_legend(title="diff (deg. C)"))) %>%
              meshplot(include_obs = 1L)
```

```
## Scale for 'fill' is already present. Adding another scale for 'fill',
## which will replace the existing scale.
```

```
X <- data.frame(M1_errors = Mesh["M1T_errors"],
                M4_errors = Mesh["M4T_errors"])
std_diff_scatter <- LinePlotTheme() +
  geom_point(data=X,aes(M1_errors,M4_errors,fill = M1_errors - M4_errors),
             colour="black",size=4,shape=21) +
  scale_fill_gradient2(low=muted("green"),mid="white", high=muted("magenta"),
                       guide=guide_legend(title="diff (deg. C)")) +
  geom_line(data=data.frame(x=c(-5,5),y=c(-5,5)),aes(x,y),linetype="dashed") +
  xlab("Model 1 standard errors for Y1 (deg. C)") +
  ylab("Model 4 standard errors for Y1 (deg. C)") +
    coord_fixed(xlim=c(0,3.5),ylim=c(0,3.5),ratio=0.8) +
  theme(text = element_text(size = 30), axis.title.y = element_text(vjust=2));
if (show_figs) print(std_diff_scatter)
```

```
if(print_figs) {
  g <- arrangeGrob(g1M4,(std_diff_scatter +
                           theme(plot.margin = grid::unit(c(10, 10, 10, 10),
                                                          units="mm"))),
                   g2M4,std_diff,ncol=2)
  ggsave(g,
         filename = file.path(img_path,"Fig3.eps"),
         width=28,height=16,family="Arial")
}
```

Package versions

If you find that the code above is not reproducing the figures precisely, it is highly likely that this is due to some new, updated package implementing things differently. The package versions used to construct this document are listed below.

```
sessionInfo()
```

```
## R version 3.2.0 (2015-04-16)
## Platform: x86_64-pc-linux-gnu (64-bit)
```

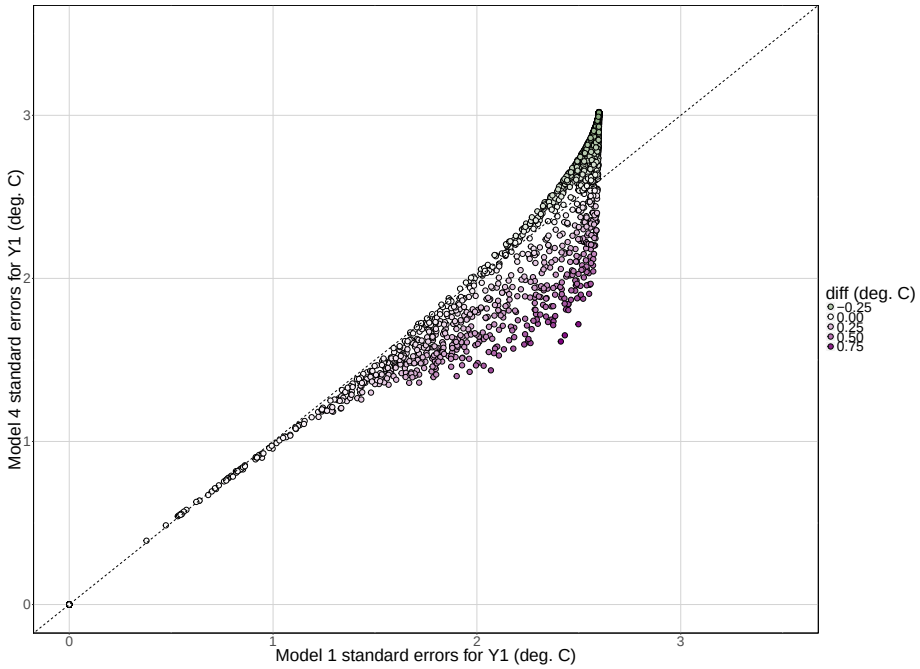

Fig. 6: A scatter plot of the kriging standard errors of Y_1 obtained with Model 4 against those obtained with Model 1 at each of the mesh vertices. The colour illustrates the difference between the two, with green denoting the higher standard error of Model 4 and purple the higher standard error of Model 1.

```
## Running under: Ubuntu 14.04.2 LTS
##
## locale:
##  [1] LC_CTYPE=en_AU.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_AU.UTF-8        LC_COLLATE=en_AU.UTF-8
##  [5] LC_MONETARY=en_AU.UTF-8    LC_MESSAGES=en_AU.UTF-8
##  [7] LC_PAPER=en_AU.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_AU.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
##  [1] parallel  grid      splines   stats     graphics  grDevices utils
##  [8] datasets  methods   base
##
## other attached packages:
##  [1] scales_0.4.0             bicon_0.1.0
##  [3] doParallel_1.0.10        iterators_1.0.8
##  [5] foreach_1.4.3            verification_1.42
##  [7] dtw_1.18-1               proxy_0.4-15
##  [9] CircStats_0.2-4          MASS_7.3-40
## [11] boot_1.3-16              fields_8.4-1
## [13] spam_1.4-0               RandomFields_3.1.1
## [15] RandomFieldsUtils_0.0.10 mapproj_1.2-4
## [17] maps_3.1.1               maptools_0.8-37
## [19] extrafont_0.17           gridExtra_2.0.0
## [21] ggplot2_2.1.0            tidyr_0.5.1
## [23] dplyr_0.5.0              INLA_0.0-1455098891
## [25] Matrix_1.2-0             sp_1.2-3
```

```
##
## loaded via a namespace (and not attached):
##  [1] network_1.13.0     akima_0.5-12       reshape2_1.4.1
##  [4] gpclib_1.5-5       lattice_0.20-31    colorspace_1.2-6
##  [7] htmltools_0.3.5    yaml_2.1.13        foreign_0.8-66
## [10] DBI_0.5-1          RColorBrewer_1.1-2 doMC_1.3.3
## [13] plyr_1.8.4         stringr_1.1.0      munsell_0.4.3
## [16] gtable_0.2.0       codetools_0.2-11   evaluate_0.9
## [19] labeling_0.3       knitr_1.13         Rttf2pt1_1.3.3
## [22] Rcpp_0.12.7        xtable_1.8-2       formatR_1.4
## [25] deldir_0.1-12      digest_0.6.10      stringi_1.1.1
## [28] tools_3.2.0        magrittr_1.5       lazyeval_0.2.0
## [31] tibble_1.2         extrafontdb_1.0    assertthat_0.1
## [34] rmarkdown_0.8.1    R6_2.1.3           compiler_3.2.0
```

References

Cressie, N., & Zammit-Mangion, A. (2016). Multivariate spatial covariance models: A conditional approach. *Biometrika, in press*.

Gneiting, T., Kleiber, W., & Schlather, M. (2010). Matérn cross-covariance functions for multivariate random fields. *Journal of the American Statistical Association*, *105*, 1167–1177.

Gneiting, T., Raftery, A. E., Westveld III, A. H., Anton, H., & Goldman, T. (2005). Calibrated probabilistic forecasting using ensemble model output statistics and minimum cRPS estimation. *Monthly Weather Review*, *133*(5), 1098–1118.

Li, B., & Zhang, H. (2011). An approach to modeling asymmetric multivariate spatial covariance structures. *Journal of Multivariate Analysis*, *102*, 1445–1453.

R Core Team. (2014). *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. Retrieved from http://www.R-project.org/.

Schlather, M., Malinowski, A., Menck, P. J., Oesting, M., & Strokorb, K. (2015). Analysis, simulation and prediction of multivariate random fields with package RandomFields. *Journal of Statistical Software*, *63*(8), 1–25.