Lab 4.2: Spatio-Temporal Basis Functions with FRK

In this Lab we shall focus on modeling the maximum temperature in July 1993 from data in the NOAA data set using spatio-temporal basis functions. The packages we need are the following:

```
library("dplyr")
library("FRK")
library("ggplot2")
library("gstat")
library("RColorBrewer")
library("sp")
library("spacetime")
library("STRbook")
library("tidyr")
```

The package **FRK** implements a low-rank approach to spatial and spatio-temporal modeling known as *fixed rank kriging* (FRK). FRK considers the random-effects model

$$Y(\mathbf{s};t) = \mathbf{x}(\mathbf{s};t)'\boldsymbol{\beta} + \sum_{i=1}^{n_\alpha} \phi_i(\mathbf{s};t)\alpha_i + \nu(\mathbf{s};t), \tag{1}$$

where $\{\phi_i(\mathbf{s};t) : i = 1,\ldots,n_\alpha\}$ are specified basis functions evaluated at space-time location $(\mathbf{s};t)$, $\{\alpha_i : i = 1,\ldots,n_\alpha\}$ are random effects, and $\nu(\mathbf{s};t)$ captures small-scale spatio-temporal variation not absorbed by the summation term. The random effects $\boldsymbol{\alpha} \sim Gau(\mathbf{0},\mathbf{C}_\alpha)$, where $\boldsymbol{\alpha} \equiv (\alpha_1,\ldots,\alpha_{n_\alpha})'$. Assume we are interested in the Y-process at n_y spatio-temporal locations, which we denote by the vector $\mathbf{Y}$. The process model then becomes

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\Phi}\boldsymbol{\alpha} + \boldsymbol{\nu}, \tag{2}$$

where ith column of the $n_y \times n_\alpha$ matrix $\boldsymbol{\Phi}$ corresponds to the ith basis function, $\phi_i(\cdot;\cdot)$ at all of the spatio-temporal locations ordered as given in $\mathbf{Y}$, and the vector $\boldsymbol{\nu}$ also corresponds to the spatio-temporal ordering given in $\mathbf{Y}$ such that $\boldsymbol{\nu} \sim Gau(\mathbf{0},\mathbf{C}_\nu)$.

A key difference between **FRK** and other geostatistical packages is that, in **FRK**, modeling and prediction are carried out on a fine, regular discretization of the spatio-temporal domain. The small grid cells are known as *basic areal units* (BAUs), and their primary utility is to account for problems of change of support (varying measurement footprint), which we do not consider in this Lab.

For spatio-temporal modeling and prediction, **FRK** requires the user to provide the point-level data as objects of class `STIDF` . Hence, for this exercise, we use `STObj5` from Lab 3.1, which we reconstruct below (for completeness) from `STObj3`.

```
data("STObj3", package = "STRbook")            # load STObj3
STObj4 <- STObj3[, "1993-07-01::1993-07-31"] # subset time
STObj5 <- as(STObj4[, -14], "STIDF")           # omit t = 14
STObj5 <- subset(STObj5, !is.na(STObj5$z))     # remove NAs
```

The spatio-temporal BAUs are constructed using the function **auto_BAUs** which takes several arguments, as shown below and detailed using the in-line comments. For more details see **help**(auto_BAUs). Note that as cellsize we chose **c**(1,0.75,1) which indicates a BAU size of 1 degree longitude $\times$ 0.75 degrees latitude $\times$ 1 day – this choice ensures that the BAUs are similar to the prediction grid used in Lab 3.1. The argument convex is an "extension radius" used in domain construction via the package **INLA**. See the help file of **inla.nonconvex.hull** for details.

```
BAUs <- auto_BAUs(manifold = STplane(),    # ST field on the plane
                  type = "grid",           # gridded (not "hex")
                  data = STObj5,           # data
                  cellsize = c(1, 0.75, 1), # BAU cell size
                  convex = -0.12,          # hull extension
                  tunit = "days")          # time unit is "days"
```

The BAUs are of class STFDF since they are three-dimensional pixels arranged regularly in both space and in time. To plot the spatial BAUs overlaid with the data locations, we run

```
plot(as(BAUs[, 1], "SpatialPixels"))     # plot pixel BAUs
plot(SpatialPoints(STObj5),
     add = TRUE, col = "red")            # plot data points
```

This generates the left panel of Figure 1. The BAUs, which we will also use as our prediction grid, overlap all the data points. The user has other options in BAU construction; for example, the following code generates *hexagonal* BAUs using a convex hull for a boundary.

```
BAUs_hex <- auto_BAUs(manifold = STplane(), # model on the plane
                  type = "hex",             # hex (not "grid")
                  data = STObj5,            # data
                  cellsize = c(1, 0.75, 1), # BAU cell size
                  nonconvex_hull = FALSE,   # convex hull
                  tunit = "days")           # time unit is "days"
```

Plotting proceeds in a similar fashion, except that the first line in the code chunk above now becomes

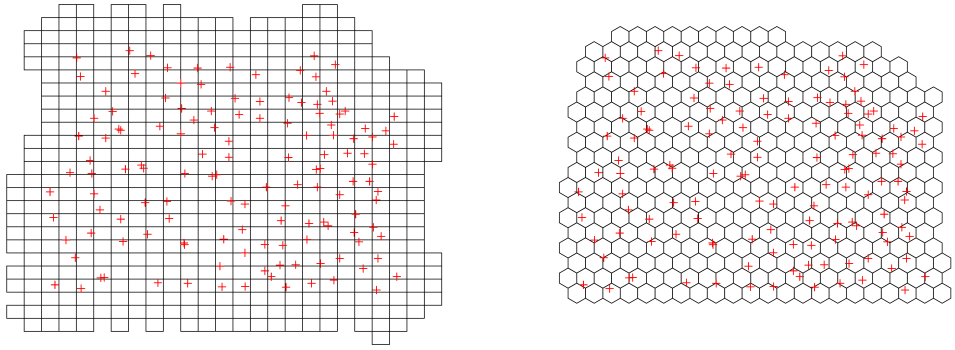

Figure 1: BAUs constructed for modeling and predicting maximum temperature from data in the NOAA data set. Left: Gridded BAUs arranged within a non-convex hull enclosing the data. Right: Hexagonal BAUs arranged within a convex hull enclosing the data.

```
plot(as(BAUs_hex[, 1], "SpatialPolygons"))
```

This allows for the fact the the BAUs are now (hexagonal) polygons and not rectangular pixels. The resulting plot is shown in the right panel of Figure 1.

Next we construct the basis functions $\{\phi_i(\mathbf{s};t) : i = 1,\ldots,n_\alpha\}$. In **FRK**, these are constructed by taking the tensor product of spatial basis functions with temporal basis functions. Specifically, consider a set of r_s spatial basis functions $\{\phi_p(\mathbf{s}) : p = 1,\ldots,r_s\}$, and a set of r_t temporal basis functions $\{\psi_q(t) : q = 1,\ldots,r_t\}$. Then we construct the set of spatio-temporal basis functions as $\{\phi_{st,u}(s,t) : u = 1,\ldots,r_s r_t\} = \{\phi_p(\mathbf{s})\psi_q(t) : p = 1,\ldots,r_s;\ q = 1,\ldots,r_t\}$.

In principle any basis function can be used in **FRK**, although these would need to be defined by the user. Examples of basis functions are given in Figure 2 The generic basis function that **FRK** uses by default is the bisquare function (see Figure 2) given by

$$b(\mathbf{s},\mathbf{v}) \equiv \begin{cases} \{1-(\|\mathbf{v}-\mathbf{s}\|/r)^2\}^2, & \|\mathbf{v}-\mathbf{s}\| \le r, \\ 0, & \text{otherwise}, \end{cases}$$

where r is the aperture parameter. Basis functions can be either regularly placed, or irregularly placed, and they are often multiresolutional. We choose two resolutions below, yielding $r_s = 94$ spatial basis functions in total, and place them irregularly in the domain. (Note that r_s and the bisquare apertures are determined automatically by `auto_basis`.)

```
G_spatial <- auto_basis(manifold = plane(),      # fns on plane
                        data = as(STObj5, "Spatial"), # project
                        nres = 2,                     # 2 res.
                        type = "bisquare",            # bisquare.
```

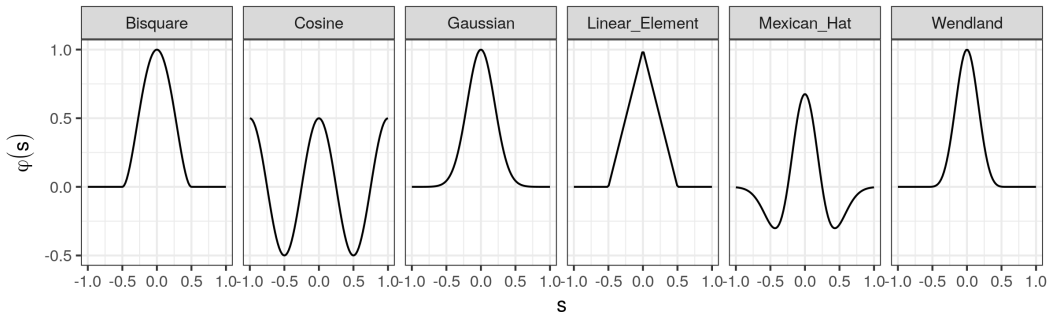

Figure 2: Spatial basis functions commonly employed in spatio-temporal modeling depicted in one-dimensional space. From left to right: The bisquare function, cosine function, Gaussian function, linear element, Mexican-hat wavelet, and the first-order Wendland function.

```
                          regular = 0)                    # irregular
```

Temporal basis functions also need to be defined. We use the function **local_basis** below to construct a regular sequence of $r_t = 20$ bisquare basis functions between day 1 and day 31 of the month. Each of these bisquare basis functions is assigned an aperture of 2 days; that is, the support of each bisquare function is 4 days. The temporal grid is defined through

```
t_grid <- matrix(seq(1, 31, length = 20))
```

The basis functions are constructed using the following commands.

```
G_temporal <- local_basis(manifold = real_line(),   # fns on R1
                          type = "bisquare",        # bisquare
                          loc = t_grid,             # centroids
                          scale = rep(2, 20))       # aperture par.
```

Finally, we construct the $r_s r_t = 1880$ spatio-temporal basis functions by taking the tensor product of the spatial and the temporal ones, using the function **TensorP**.

```
G <- TensorP(G_spatial, G_temporal)       # take the tensor product
```

The basis functions `G_spatial` and `G_temporal` can be visualized using the plotting function **show_basis**; see Figure 3. While the basis functions are of tensor-product form, the resulting S-T covariance function obtained from the spatio-temporal random effects model is not separable in space and time.

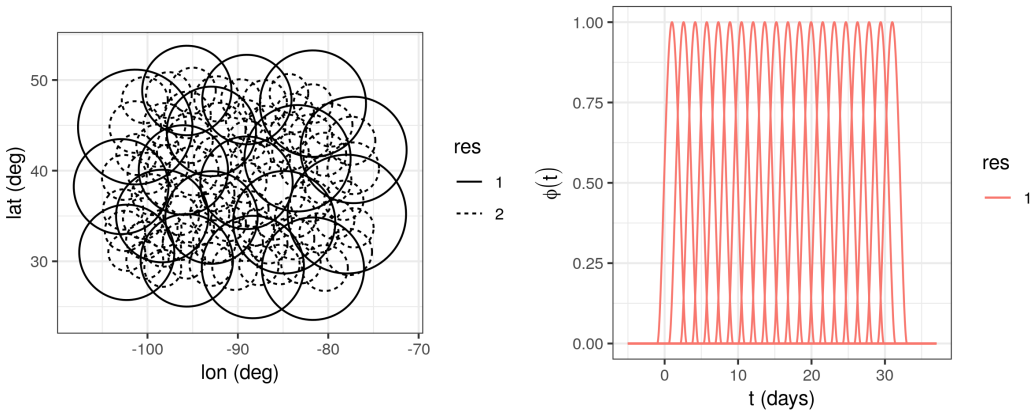

Figure 3: Spatial and temporal basis functions used to construct the spatio-temporal basis functions. Left: Locations of spatial basis functions (circles denote spatial support). Right: Temporal basis functions.

In **FRK**, the fine-scale variation term at the BAU level, (2), is assumed to be Gaussian with covariance matrix proportional to $\mathrm{diag}(\{\sigma^2_{\nu,i}\})$, where $\{\sigma^2_{\nu,i} : i = 1, \ldots, n_y\}$ are pre-specified at the BAU level (the constant of proportionality is then estimated by **FRK**). Typically, these are related to some geographically related quantity such as surface roughness. In our case, we simply set $\sigma^2_{\nu,i} = 1$ for all i.

```
BAUs$fs = 1
```

The fine-scale variance at the BAU level is confounded with the measurement-error variance. In some cases, the measurement-error variance is known; when it is not (as in this case), one can carry out a simple analysis to estimate the value of the semivariogram at the origin. In this case, we simply assume that the nugget effect estimated when fitting the separable covariance function in Lab 4.1 is the measurement-error variance – any residual nugget component is then assumed to be the fine-scale variance introduced as a consequence of the low-rank approximation to the process. The measurement-error variance is specified in the `std` field in the data `ST` object.

```
STObj5$std <- sqrt(0.049)
```

The response variable and covariates are identified through a standard `R` formula. In this case we use latitude as a covariate and set

```
f <- z ~ lat + 1
```

We are now ready to call the main function **FRK**, which estimates all the unknown parameters in the models, including the covariance matrix of the basis-function coefficients and the fine-scale variance. We need to supply the formula, the data, the basis functions, the BAUs, and any other parameters configuring the expectation-maximization (EM) algorithm used for finding the maximum likelihood estimates. To reduce processing time, we have set the number of EM-algorithm steps to 3. Convergence of the EM algorithm can be assessed visually by setting `print_lik = TRUE` below.

```
S <- FRK(f = f,                 # formula
         data = list(STObj5), # (list of) data
         basis = G,             # basis functions
         BAUs = BAUs,           # BAUs
         n_EM = 3,              # max. no. of EM iterations
         tol = 0.01)            # tol. on change in log-likelihood
```

Once the model is fitted, prediction proceeds via the function **predict**. If the argument `newdata` is not specified, then prediction is done at all the BAUs.

```
grid_BAUs <- predict(S)
```

The resulting object, `grid_BAUs`, is also of class `STFDF`, and plotting proceeds as per Lab 4.1 using the **stplot** function. Plotting of the FRK predictions and prediction standard errors is left as an exercise.