Lab 4.3: Temporal Basis Functions with SpatioTemporal

In this Lab we model the maximum temperature in the NOAA data set (`Tmax`) using temporal basis functions and spatial random fields. Specifically, we use the model

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

where $\mathbf{x}(\mathbf{s};t)$ are the covariates; $\boldsymbol{\beta}$ are the regression coefficients; $\{\phi_i(t)\}$ are the temporal basis functions; $\{\alpha_i(\mathbf{s})\}$ are coefficients of the temporal basis functions, modeled as multivariate (spatial) random fields; and $\nu(\mathbf{s};t)$ is a spatially correlated, but temporally independent, random process.

Spatio-temporal modeling using temporal basis functions can be carried out using the package **SpatioTemporal**. For this Lab we require the following packages.

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

The space-time object used by **SpatioTemporal** is of class `STdata` and is created using the function `createSTdata`. This function takes the data either as a space-wide matrix with the row names containing the date and the column names the station ID, or as a data frame in long form. Here we use the latter. This data frame needs to have the station ID as characters in the field `ID`, the data in the field `obs`, and the date in the field `date`. A new data frame of this form can be easily created using the function `transmute` from the package **dplyr**.

```r
data("NOAA_df_1990", package = "STRbook")   # load NOAA data
NOAA_sub <- filter(NOAA_df_1990,        # filter data to only
                   year == 1993 &       # contain July 1993
                   month == 7 &
                   proc == "Tmax")      # and just max. temp.

NOAA_sub_for_STdata <- NOAA_sub %>%
                       transmute(ID = as.character(id),
                                 obs = z,
                                 date = date)
```

The covariates that will be used to model the spatially varying effects also need to be supplied as a data frame. In our case we only consider the station coordinates as covariates. The station coordinates are extracted from the maximum temperature data as follows.

```
covars <-  dplyr::select(NOAA_sub, id, lat, lon) %>%
           unique() %>%
           dplyr::rename(ID = id)        # createSTdata expects "ID"
```

Now we can construct the `STdata` object by calling the function **`createSTdata`**.

```
STdata <- createSTdata(NOAA_sub_for_STdata, covars = covars)
```

The model used in **SpatioTemporal** assumes that $\nu(\mathbf{s};t)$ is temporally uncorrelated. Consequently, all temporal variability needs to be captured through the covariates or the basis functions. To check whether the data exhibit temporal autocorrelation (before adding any temporal basis functions), one can use the **`plot`** function. For example, we plot the estimated autocorrelation function for station 3812 in the left panel of Figure 1 (after the mean is removed from the data). The plot suggests that the data are correlated (the estimated lag-1 autocorrelation coefficient is larger than would be expected by chance at the 5% level of significance).

```
plot(STdata, "acf", ID = "3812")
```

The role of the temporal basis functions is to adequately capture temporal modes of variation. When modeling data over a time interval that spans years, one of these is typically a seasonal component. As another example, when modeling trace-gas emissions, one basis function to use would be one that captures weekday/weekend cycles typically found in gaseous pollutants (e.g., due to vehicular traffic). The package **SpatioTemporal** allows for user-defined basis functions (see the example at the end of this Lab) or data-driven basis functions (which we consider now). In both cases, the first temporal basis function, $\phi_1(t)$, is a constant; that is, $\phi_1(t) = 1$.

The basis functions extracted from the data are *smoothed*, *left singular vectors* (i.e., smoothed temporal EOFs) of the matrix $\widetilde{\mathbf{Z}}$. These make up the remaining $n_\alpha - 1$ basis functions, upon which smoothing is carried out using splines. In **SpatioTemporal**, these basis functions are found (or set) using the function **`updateTrend`**.

```
STdata <- updateTrend(STdata, n.basis = 2)
```

We can see that the lag-1 autocorrelation coefficient is no longer significant (at the 5% level) after adding in these basis functions; see the right panel of Figure 1. In practice, one should add basis functions until temporal autocorrelation in the data (at most stations) is considerably reduced. In this case study, it can be shown that 69% of stations record maximum temperature data that have lag-1 autocorrelation coefficients that are significant at

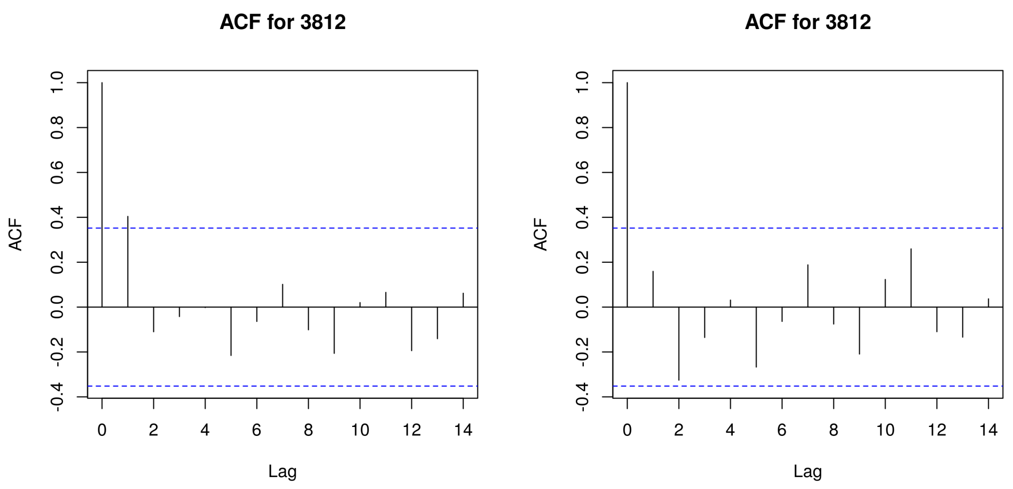

Figure 1: Left: Estimated autocorrelation function for the time series of maximum temperature `Tmax` at Station 3812. Right: Same as left panel, but with the data first detrended using an intercept and the two temporal basis functions shown in the top panel of Figure 2.

the 5% level. On the other hand, with `n.basis = 2` (i.e., with two temporal basis functions for capturing temporal variation), the proportion of stations with residuals exhibiting a significant lag-1 autocorrelation coefficient is 26%.

```
plot(STdata, "acf", ID = "3812")
```

The basis functions, available in `STdata$trend`, are shown in the top panel of Figure 2.

In **SpatioTemporal**, the spatial quantities $\{\alpha_i(\mathbf{s})\}$ are themselves modeled as spatial fields. Once the $\{\phi_i(t)\}$ are declared, empirical estimates of $\{\alpha_i(\mathbf{s})\}$ can be found using the function **`estimateBetaFields`**. Note that we use the Greek letter "alpha" to denote these fields, which differs from the name "Beta" inside the command. The following and all subsequent references to "Beta" and "beta" should be interpreted as representing spatial fields $\{\alpha_i(\mathbf{s})\}$.

```
beta.lm <- estimateBetaFields(STdata)
```

The resulting object, `beta.lm`, contains two fields; `beta` (estimated coefficients) and `beta.sd` (standard error of the estimates) with row names equal to the station ID, and three columns corresponding to estimates of $\alpha_1(\mathbf{s})$, $\alpha_2(\mathbf{s})$, and $\alpha_3(\mathbf{s})$, respectively. We are interested in seeing whether the empirical estimates are correlated with our covariate, latitude. To this end, the authors of **SpatioTemporal** suggest using the package **plotrix**, and the function plotCI, to plot the estimates and covariance intervals against a covariate of choice. When plotting using plotCI, care should be taken that the ordering of the stations in beta and beta.sd is the same as that if the covariate data frame. For example, consider

```
head(row.names(beta.lm$beta))

## [1] "13865" "13866" "13871" "13873" "13874" "13876"

head(covars$ID)

## [1] 3804 3810 3811 3812 3813 3816
```

This illustrates a discrepancy, since the ordering of strings is not necessarily that of the ordered integers. For this reason we recommend employing best practice and always merging (e.g., using left_join) on a column variable; in this case, we choose the integer version of the field ID. In the following commands, we first convert the beta and beta.sd objects into data frames, add the column ID, join into a data frame BETA, and then combine with covars containing the latitude data.

```
beta.lm$beta <- data.frame(beta.lm$beta)
beta.lm$beta.sd <- data.frame(beta.lm$beta.sd)
beta.lm$beta$ID <- as.integer(row.names(beta.lm$beta))
BETA <- cbind(beta.lm$beta, beta.lm$beta.sd)
colnames(BETA) <- c("alpha1", "alpha2", "alpha3", "ID",
                    "alpha1_CI", "alpha2_CI", "alpha3_CI")
BETA <- left_join(BETA, covars, by = "ID")
```

Once BETA is constructed, the empirical estimates can be plotted using ggplot, with geom_errorbar to also plot error bars, as follows.

```
ggplot(BETA) + geom_point(aes(x = lat, y = alpha1)) +
    geom_errorbar(aes(x = lat,
                      ymin = alpha1 - 1.96*alpha1_CI,
                      ymax = alpha1 + 1.96*alpha1_CI)) +
    ylab(expression(alpha[1](s))) +
    xlab("lat (deg)") + theme_bw()
```

The three empirical estimates, plotted as a function of latitude, are shown in Figure 3. The function $\alpha_1(\mathbf{s})$ exhibits a strong latitudinal trend, as expected; $\alpha_2(\mathbf{s})$ shows a weak latitudinal trend; and $\alpha_3(\mathbf{s})$ exhibits no trend. For this reason we model the expectations of

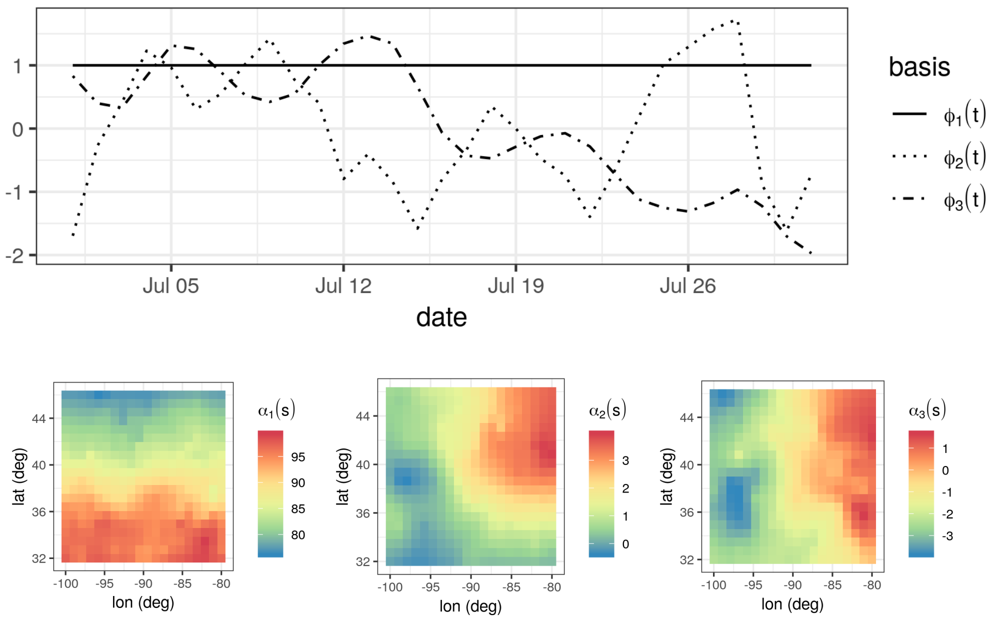

Figure 2: Top: Basis functions $\phi_1(t)$, $\phi_2(t)$, and $\phi_3(t)$, where the latter two were obtained from the left-singular vectors following a singular value decomposition of the data matrix (i.e., temporal EOFs). Bottom: $E(\alpha_1(\mathbf{s}) \mid \mathbf{z})$, $E(\alpha_2(\mathbf{s}) \mid \mathbf{z})$, and $E(\alpha_3(\mathbf{s}) \mid \mathbf{z})$.

these fields as

$$E(\alpha_1(\mathbf{s})) = \alpha_{11} + \alpha_{12}s_2, \tag{2}$$
$$E(\alpha_2(\mathbf{s})) = \alpha_{21} + \alpha_{22}s_2, \tag{3}$$
$$E(\alpha_3(\mathbf{s})) = \alpha_{31}, \tag{4}$$

where s_2 denotes the latitude coordinate at $\mathbf{s} = (s_1, s_2)'$. Note that in this model we do not consider any spatio-temporal covariates, and hence the term $\mathbf{x}(\mathbf{s}; t)'\boldsymbol{\beta} = 0$ in (1). This does not mean that we do not have an intercept in our model: although it is random, the spatial field $\alpha_1(\mathbf{s})$ acts as a temporally invariant spatial covariate and includes a global space-time mean (α_{11} in (2)), which is estimated.

We let the covariance functions $\mathrm{cov}(\alpha_i(\mathbf{s}), \alpha_i(\mathbf{s}+\mathbf{h}))$, $i = 1, 2, 3$, be exponential covariance functions without a nugget-effect term. In **SpatioTemporal** these are declared as follows.

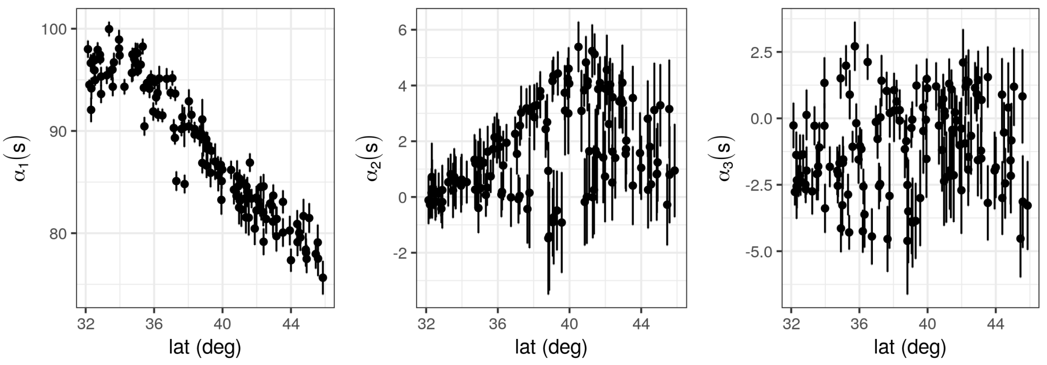

Figure 3: Empirical estimates of $\alpha_1(\mathbf{s})$, $\alpha_2(\mathbf{s})$, and $\alpha_3(\mathbf{s})$ at each station, with 95% confidence intervals, plotted as a function of latitude.

```
cov.beta <- list(covf = "exp", nugget = FALSE)
```

All that remains for constructing the spatio-temporal model is to define the spatial covariance function of the zero-mean, temporally independent, residual process $\nu(\mathbf{s};t)$. We choose this to be an exponential covariance function with a nugget effect to account for measurement error. The argument `random.effect = FALSE` is used to indicate that there is no random mean offset for the field at each time point.

```
cov.nu <- list(covf = "exp",
               nugget = ~1,
               random.effect = FALSE) # No random mean
                                      # for each nu
```

The function to create the spatio-temporal model is **`createSTmodel`**. This takes as data the object `STdata`, the covariates for the α-fields (an intercept and latitude for $\alpha_1(\mathbf{s})$ and $\alpha_2(\mathbf{s})$, and just an intercept for $\alpha_3(\mathbf{s})$; see (2)–(4), the covariance functions of the α-fields and the ν-field, and a list containing the names of station coordinate fields (`lon` and `lat`).

```
locations <- list(coords = c("lon", "lat"))
LUR <- list(~lat, ~lat, ~1)   # lat trend for phi1 and phi2 only
STmodel <- createSTmodel(STdata,                # data
                         LUR = LUR,             # spatial covariates
                         cov.beta = cov.beta, # cov. of alphas
                         cov.nu = cov.nu,       # cov. of nu
                         locations = locations) # coord. names
```

In order to fit the spatio-temporal model to the data, we need to provide initial values of the parameter estimates. The required parameter names can be extracted using the function **loglikeSTnames** and, for our model, are as follows.

```
parnames <- loglikeSTnames(STmodel, all = FALSE)
print(parnames)

## [1] "log.range.const.exp"          "log.sill.const.exp"
## [3] "log.range.V1.exp"             "log.sill.V1.exp"
## [5] "log.range.V2.exp"             "log.sill.V2.exp"
## [7] "nu.log.range.exp"             "nu.log.sill.exp"
## [9] "nu.log.nugget.(Intercept).exp"
```

Noting that all parameters are log-transforms of the quantities of interest, we let all of the initial values be equal to 3 (so that all initial ranges and sills are $e^3 \approx 20$). This seems reasonable when the temperature is varying on the order of several degrees Fahrenheit, and where the domain also spans several degrees (in latitude and longitude).

We use the function **estimate** below to fit the spatio-temporal model to the data. This may take several minutes on a standard desktop computer. In this instance, the resulting object `SpatioTemporalfit1` has been pre-computed and can be loaded directly from **STRbook** by typing `data("SpatioTemporalfit1", package = "STRbook")`.

```
x.init <- matrix(3, 9, 1)
rownames(x.init) <- loglikeSTnames(STmodel, all = FALSE)
SpatioTemporalfit1 <- estimate(STmodel, x.init)
```

The fitted coefficients for the parameters described by `parnames` above can be extracted from the fitted object using the function **coef**.

```
x.final <- coef(SpatioTemporalfit1, pars = "cov")$par
```

Having fitted the model, we now predict at unobserved locations. First, we establish the spatial and temporal grid upon which to predict; this proceeds by first initializing an `STdata` object on a grid. We construct the grid following a very similar approach to what was done in Lab 4.1.

```
## Define space-time grid
spat_pred_grid <- expand.grid(lon = seq(-100, -80, length = 20),
                              lat = seq(32, 46, length = 20))
spat_pred_grid$id <- 1:nrow(spat_pred_grid)
temp_pred_grid <- as.Date("1993-07-01") + seq(3, 28, length = 6)

## Initialize data matrix
obs_pred_wide <- matrix(0, nrow = 6, ncol = 400)
```

```
## Set row names and column names
rownames(obs_pred_wide) <- as.character(temp_pred_grid)
colnames(obs_pred_wide) <- spat_pred_grid$id

covars_pred <- spat_pred_grid                        # covariates
STdata_pred <- createSTdata(obs = obs_pred_wide,   # ST object
                            covars = covars_pred)
```

Now prediction proceeds using the function **predict**, which requires as arguments the model, the fitted model parameters, and the data matrix `STdata_pred`.

```
E <- predict(STmodel, x.final, STdata = STdata_pred)
```

The returned object `E` contains both the α-fields predictions as well as the Y-field prediction at the unobserved locations. For example, `E$beta$EX` contains the conditional expectations of $\alpha_1(\mathbf{s}), \alpha_2(\mathbf{s})$, and $\alpha_3(\mathbf{s})$ given the data. For conciseness, we do not illustrate the plotting commands here. In the bottom panels of Figure 2, we show the conditional expectations, while in Figures 4 and 5 we show the predictions and prediction standard errors of maximum temperature over six days of interest in July 1993.

Using SpatioTemporal for Modeling Spatial Effects of Temporal Covariates

In the first part of this Lab, we extracted the temporal basis functions from the data. However, **SpatioTemporal** can also be used to model the spatially varying effect of exogenous temporal covariates. This can be done by manually setting the `STdata$trend` data frame. When modeling temperature, interesting covariates may include a periodic signal with period equal to one year, or an index such as the El Niño Southern Oscillation (ENSO) Index.

To use a pre-existing covariate, we need to use the `fnc` argument in **updateTrend** to define a function that takes a `Date` object as an input and returns the covariate at these dates. The easiest way to do this in this example is to specify a look-up table in the function containing the covariate for each date, but an interpolant can also be used when the covariate has missing information for one or more dates.

As an exercise, repeat the Lab above, but this time use a single linear temporal trend as a temporal covariate. The look-up table we need is just a two-column data frame containing the `date` in the first column, and `V1` (first covariate) in the second column. This can be set up as follows.

```
all_dates <- NOAA_sub$date %>% unique()       # dates
lookup <- data.frame(date = all_dates,         # covariate (linear)
                     V1 = scale(as.numeric(all_dates)))
```

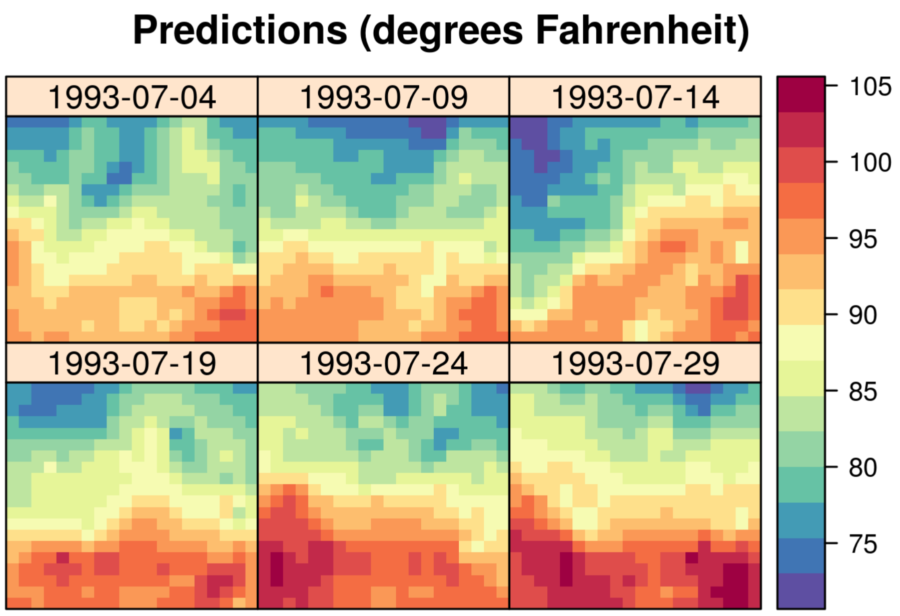

Figure 4: Predictions of `Tmax` in degrees Fahrenheit within a square lat-lon box defining the spatial domain of interest, for six days in July 1993, using temporal basis functions. Data for 14 July 1993 were deliberately omitted from the original data set.

Type `plot(lookup)` to see the temporal covariate that we have just created. Now we need to create the function that takes a `Date` object as input and returns the required covariate values. This can be done using `left_join`.

```
## Function that returns the covariates in a data frame
## at the required dates
fnc <- function(dates) {
  left_join(data.frame(date = dates),
            lookup, by = "date") %>%
  select(-date)
}
```

Now we can call `updateTrend` with our covariate function as argument.

```
STdata <- updateTrend(STdata, fnc = fnc)
```

The rest of the code remains largely similar, except that now we are considering only two temporal basis functions and not three (the first basis function is constant in time, and

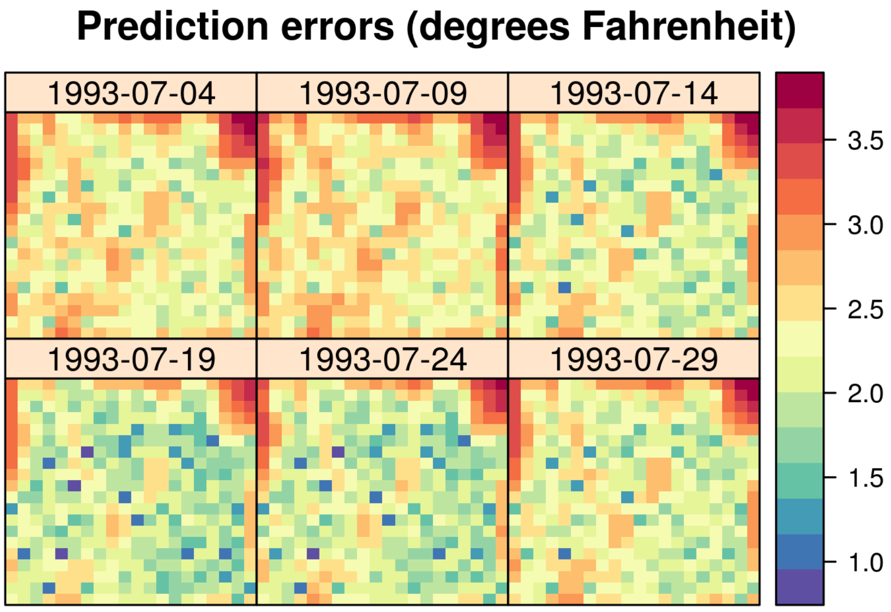

Figure 5: Prediction standard errors of `Tmax` in degrees Fahrenheit within a square lat-lon box enclosing the spatial domain of interest, for six days in July 1993, using temporal basis functions. Data for 14 July 1993 were deliberately omitted from the original data set.

the second one is linear in time). Changing the required parts of the code is left as an exercise.