Lab 3.2: Trend Prediction

There is considerable in-built functionality in `R` for linear regression and for carrying out hypothesis tests associated with linear models. Several packages have also been written to extend functionality, and in this Lab we shall make use of **leaps**, which contains functionality for stepwise regression; **lmtest**, which contains a suite of tests to carry out on fitted linear models; and **nlme**, which is a package generally used for fitting nonlinear mixed effects models (but we shall use it to fit linear models in the presence of correlated errors).

```r
library("leaps")
library("lmtest")
library("nlme")
```

In addition, we use **ape**, which is one of several packages that contain functionality for testing spatial or spatio-temporal independence with Moran's I statistic; and we use **FRK**, which contains functionality for constructing the basis functions that we will use inside the linear regression. We also make use of **broom** and **purrr** to easily carry out multiple tests on groups within our data set.

```r
library("ape")
library("broom")
library("FRK")
library("purrr")
```

We need the following for plotting purposes.

```r
library("lattice")
library("ggplot2")
library("RColorBrewer")
```

We also need the usual packages for data wrangling and handling of spatial/spatio-temporal objects as in the previous Labs.

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

Fitting the Model

For this Lab we again consider the NOAA data set, specifically the maximum temperature data for the month of July 1993. These data can be extracted as follows.

```
data("NOAA_df_1990", package = "STRbook")
Tmax <- filter(NOAA_df_1990,       # subset the data
               proc == "Tmax" &    # only max temperature
               month == 7 &        # July
               year == 1993)       # year of 1993
```

The linear model we fit has the form

$$Z(\mathbf{s}_i;t) = \beta_0 + \beta_1 X_1(\mathbf{s}_i;t) + \ldots + \beta_p X_p(\mathbf{s}_i;t) + e(\mathbf{s}_i;t), \quad (1)$$

for $i = 1,\ldots,n$ and $t = 1,\ldots,T$, where β_0 is the intercept and β_j $(j > 0)$ is a regression coefficient associated with $X_j(\mathbf{s}_i;t)$, the jth covariate at spatial location $\mathbf{s}_i$ and time t. We also assume independent errors such that $e(\mathbf{s}_i;t) \sim \;indep.\; N(0,\sigma_e^2)$. We consider a model with linear space-time interactions and a set of basis functions that fill the spatial domain:

- linear in lon-coordinate: $X_1(\mathbf{s}_i;t) = s_{1,i}$, for all t;
- linear in lat-coordinate: $X_2(\mathbf{s}_i;t) = s_{2,i}$, for all t;
- linear time (day) trend: $X_3(\mathbf{s}_i;t) = t$, for all $\mathbf{s}_i$;
- lon–lat interaction: $X_4(\mathbf{s}_i;t) = s_{1,i}s_{2,i}$, for all t;
- lon–t interaction: $X_5(\mathbf{s}_i;t) = s_{1,i}t_i$, for all $s_{2,i}$;
- lat–t interaction: $X_6(\mathbf{s}_i;t) = s_{2,i}t_i$, for all $s_{1,i}$;
- spatial basis functions: $X_j(\mathbf{s}_i;t) = \phi_{j-6}(\mathbf{s}_i), j = 7,\ldots,18$, for all t.

The set of basis functions can be constructed using the function **`auto_basis`** in **FRK**. The function takes as arguments `data`, which is a spatial object; `nres`, which is the number of "resolutions" to construct; and `type`, which indicates the type of basis function to use. Here we consider a single resolution of the Gaussian radial basis function; see Figure 1.

```
G <- auto_basis(data = Tmax[,c("lon","lat")] %>%   # Take Tmax
                       SpatialPoints(),            # To sp obj
                nres = 1,                          # One resolution
                type = "Gaussian")                 # Gaussian BFs
```

These basis functions evaluated at data locations are then the covariates we seek for fitting the data. The functions are evaluated at any arbitrary location using the function **`eval_basis`**. This function requires the locations as a `matrix` object, and it returns the evaluations as an object of class `Matrix`, which can be easily converted to a `matrix` as follows.

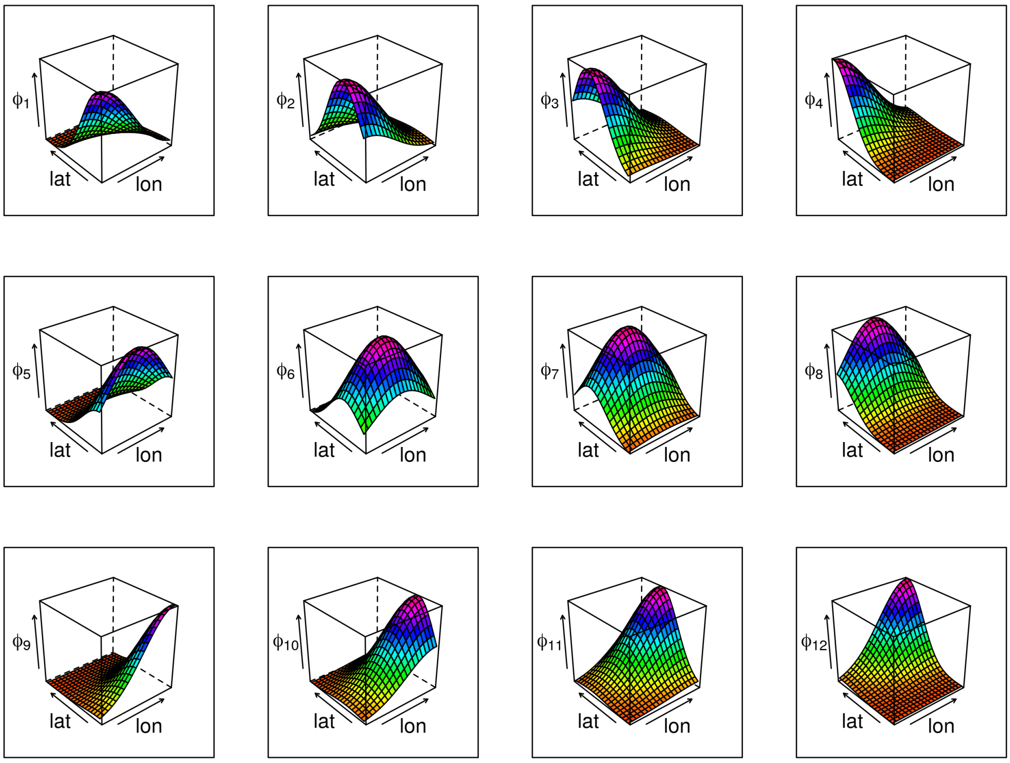

Figure 1: The time-invariant basis functions $\phi_1(\cdot), \ldots, \phi_{12}(\cdot)$ used for regression prediction of the NOAA maximum temperature data in July 1993.

```
S <- eval_basis(basis = G,                        # basis functions
                s = Tmax[,c("lon","lat")] %>%     # spat locations
                     as.matrix()) %>%             # conv. to matrix
     as.matrix()                                  # results as matrix
colnames(S) <- paste0("B", 1:ncol(S)) # assign column names
```

When fitting the linear model we shall use the convenient notation “`.`” to denote “all variables in the data frame” as covariates. This is particularly useful when we have many covariates, such as the 12 basis functions above. Therefore, we first remove all variables (except the field `id` that we shall omit manually later) that we do not wish to include in the model, and we save the resulting data frame as `Tmax2`.

```
Tmax2 <- cbind(Tmax, S) %>%                # append S to Tmax
         select(-year, -month, -proc,      # and remove vars we
                -julian, -date)            # will not be using in
                                           # the model
```

As we did in Lab 3.1, we also remove 14 July 1993 to see how predictions on this day are affected, given that we have no data on that day.

```
Tmax_no_14 <- filter(Tmax2, !(day == 14))  # remove day 14
```

We now fit the linear model using `lm`. The formula we use is `z ~ (lon + lat + day)^2 +` which indicates that we have as covariates longitude, latitude, day, and all the interactions between them, as well as the other covariates in the data frame (the 12 basis functions) without interactions.

```
Tmax_July_lm <- lm(z ~ (lon + lat + day)^2 + .,        # model
                   data = select(Tmax_no_14, -id))    # omit id
```

The results of this fit can be viewed using `summary`. Note that latitude is no longer considered a significant effect, largely because of the presence of the latitude-by-day interaction in the model, which is considered significant.

```
Tmax_July_lm %>% summary()

##
## Call:
## lm(formula = z ~ (lon + lat + day)^2 + ., data = select(Tmax_no_14,
##     -id))
##
## Residuals:
##    Min      1Q Median      3Q     Max
## -17.51  -2.48   0.11    2.66   14.17
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 192.24324   97.85413    1.96  0.04953 *
## lon           1.75692    1.08817    1.61  0.10649
## lat          -1.31740    2.55563   -0.52  0.60624
## day          -1.21646    0.13355   -9.11  < 2e-16 ***
## B1           16.64662    4.83240    3.44  0.00058 ***
## B2           18.52816    3.05608    6.06  1.5e-09 ***
## B3           -6.60690    3.17176   -2.08  0.03731 *
## B4           30.54536    4.36959    6.99  3.2e-12 ***
## B5           14.73915    2.74687    5.37  8.5e-08 ***
## B6          -17.54118    3.42308   -5.12  3.1e-07 ***
## B7           28.47220    3.55190    8.02  1.4e-15 ***
## B8           -27.34815    3.16432   -8.64  < 2e-16 ***
## B9           -10.23478    4.45673   -2.30  0.02170 *
## B10           10.55823    3.32737    3.17  0.00152 **
## B11          -22.75766    3.53251   -6.44  1.3e-10 ***
## B12           21.86438    4.81294    4.54  5.7e-06 ***
## lon:lat       -0.02602    0.02823   -0.92  0.35675
## lon:day       -0.02270    0.00129  -17.62  < 2e-16 ***
## lat:day       -0.01903    0.00188  -10.15  < 2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.22 on 3970 degrees of freedom
## Multiple R-squared:  0.702,Adjusted R-squared:  0.701
## F-statistic:  520 on 18 and 3970 DF,  p-value: <2e-16
```

Correlated Errors

As we show later in this Lab, there is clearly correlation in the residuals, indicating that the fixed effects are not able to explain the spatio-temporal variability in the data. If we knew the spatio-temporal covariance function of these errors, we could then use generalized least squares to fit the model. For example, if we knew that the covariance function was a Gaussian function, isotropic, and with a range of 0.5 (see Chapter 4 for more details on covariance functions), then we could fit the model as follows.

```
Tmax_July_gls <- gls(z ~ (lon + lat + day)^2 + .,
                     data = select(Tmax_no_14, -id),
                     correlation = corGaus(value = 0.5,
                                          form = ~ lon + lat + day,
                                          fixed = TRUE))
```

Results of the linear fitting can be seen using `summary`. Note that the estimated coefficients are quite similar to those using linear regression, but the standard errors are larger.

Stepwise Selection

Stepwise selection is a procedure used to find a parsimonious model (where parsimony refers to a model with as few parameters as possible for a given criterion) from a large selection of explanatory variables, such that each variable is included or excluded in a *step*. In the simplest of cases, a step is the introduction of a variable (always the case in forward selection) or the removal of a variable (always the case in backward selection).

The function `step` takes as arguments the initial (usually the intercept) model as an `lm` object, the full model as its `scope` and, if `direction = "forward"`, starts from an intercept model and at each step introduces a new variable that minimizes the Akaike information criterion (AIC) of the fitted model. The following `for` loop retrieves the fitted model for each step of the stepwise AIC forward-selection method.

```
Tmax_July_lm4 <- list()    # initialize
for(i in 0:4) {            # for four steps (after intercept model)
   ## Carry out stepwise forward selection for i steps
   Tmax_July_lm4[[i+1]] <- step(lm(z ~ 1,
                               data = select(Tmax_no_14, -id)),
                               scope = z ~(lon + lat + day)^2 + .,
                               direction = 'forward',
                               steps = i)
}
```

Each model in the list can be analyzed using **summary**, as above.

Notice from the output of **summary** that `Tmax_July_lm4[[5]]` contains the covariate `lon` whose effect is not significant. This is fairly common with stepwise AIC procedures. One is more likely to include covariates whose effects are significant when minimizing the residual sum of squares at each step. This can be carried out using the function **regsubsets** from the **leaps** package, which can be called as follows.

```
regfit.full = regsubsets(z ~ 1 + (lon + lat + day)^2 + .,   # model
                         data = select(Tmax_no_14, -id),
                         method = "forward",
                         nvmax = 4)                      # 4 steps
```

All information from the stepwise-selection procedure is available in the object returned by the **summary** function.

```
regfit.summary <- summary(regfit.full)
```

You can type `regfit.summary` to see which covariates were selected in each step of the algorithm.

Multicollinearity

It is fairly common in spatio-temporal modeling to have multicollinearity, both in space and in time. For example, in a spatial setting, average salary might be highly correlated with unemployment levels, but both could be included in a model to explain life expectancy. It is beyond the scope of this course to discuss methods to deal with multicollinearity, but it is important to be aware of its implications.

Consider, for example, a setting where we have a 13th basis function that is simply the 5th basis function corrupted by some noise.

```
set.seed(1) # Fix seed for reproducibility
Tmax_no_14_2 <- Tmax_no_14 %>%
                mutate(B13 = B5 + 0.01*rnorm(nrow(Tmax_no_14)))
```

If we fit the same linear model, but this time we include the 13th basis function, then the effects of both the 5th and the 13th basis functions are no longer considered significant at the 1% level, although the effect of the 5th basis function was considered very significant initially (without the 13th basis function being present).

```
Tmax_July_lm3 <- lm(z ~ (lon + lat + day)^2 + .,
                    data = Tmax_no_14_2 %>%
                           select(-id))
```

```
summary(Tmax_July_lm3)

##
## Call:
## lm(formula = z ~ (lon + lat + day)^2 + ., data = Tmax_no_14_2 %>%
##     select(-id))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -17.787  -2.495   0.103   2.674  14.318
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 195.36509   97.81955    2.00  0.04587 *
## lon           1.78547    1.08775    1.64  0.10079
## lat          -1.41413    2.55484   -0.55  0.57995
## day          -1.21585    0.13349   -9.11  < 2e-16 ***
## B1           16.67581    4.83018    3.45  0.00056 ***
## B2           18.37950    3.05544    6.02  2.0e-09 ***
## B3           -6.47093    3.17092   -2.04  0.04135 *
## B4           30.30399    4.36900    6.94  4.7e-12 ***
## B5            0.60329    7.09215    0.09  0.93221
## B6          -17.32202    3.42300   -5.06  4.4e-07 ***
## B7           28.13555    3.55367    7.92  3.1e-15 ***
## B8          -27.00216    3.16690   -8.53  < 2e-16 ***
## B9          -10.18176    4.45474   -2.29  0.02233 *
## B10          10.37967    3.32686    3.12  0.00182 **
## B11         -22.41943    3.53434   -6.34  2.5e-10 ***
## B12          21.66451    4.81160    4.50  6.9e-06 ***
## B13          13.99856    6.47562    2.16  0.03070 *
## lon:lat      -0.02694    0.02822   -0.95  0.33981
## lon:day      -0.02263    0.00129  -17.56  < 2e-16 ***
## lat:day      -0.01888    0.00188  -10.07  < 2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.22 on 3969 degrees of freedom
## Multiple R-squared:  0.703,Adjusted R-squared:  0.701
## F-statistic:  494 on 19 and 3969 DF,  p-value: <2e-16
```

The introduction of the 13th basis function will not adversely affect the predictions and prediction standard errors, but it does compromise our ability to correctly interpret the fixed effects. Multicollinearity will result in a high positive or negative correlation between the estimators of the regression coefficients. For example, the correlation matrix of the estimators of the fixed effects corresponding to these two basis functions is given by

```
vcov(Tmax_July_lm3)[c("B5", "B13"),c("B5", "B13")] %>%
    cov2cor()

##             B5      B13
## B5   1.00000 -0.92202
## B13 -0.92202  1.00000
```

Analyzing the Residuals

Having fitted a spatio-temporal model, it is good practice to check the residuals. If they are still spatio-temporally correlated, then our model will not have captured adequately the spatial and temporal variability in the data. We extract the residuals from our linear model using the function `residuals`.

```
Tmax_no_14$residuals <- residuals(Tmax_July_lm)
```

Now let us plot the residuals of the last eight days. Notice how these residuals are strongly spatially correlated. The triangles in the image correspond to the two stations whose time series of residuals we shall analyze later.

```
g <- ggplot(filter(Tmax_no_14, day %in% 24:31)) +
  geom_point(aes(lon, lat, colour = residuals)) +
  facet_wrap(~ day, ncol=4) +
  col_scale(name = "degF") +
  geom_point(data = filter(Tmax_no_14,day %in% 24:31 &
                                         id %in% c(3810, 3889)),
             aes(lon, lat), colour = "black",
             pch = 2, size = 2.5) +
  theme_bw()
```

```
print(g)
```

One of the most used tests for spatial dependence for lattice spatial data is Moran's IG test. This can be applied to the data directly, or to the residuals from some spatial regression model. Let Z_i represent spatially referenced data (or residuals) for $i = 1, \ldots, n$ spatial locations. Then Moran's I is calculated as

$$I = \frac{n \sum_{i=1}^n \sum_{j=1}^n w_{ij}(Z_i - \bar{Z})(Z_j - \bar{Z})}{(\sum_{i=1}^n \sum_{j=1}^n w_{ij})(\sum_{i=1}^n (Z_i - \bar{Z})^2)}, \qquad (2)$$

where $\bar{Z} = (1/n)\sum_{i=1}^n Z_i$ is the spatial mean and w_{ij} are spatial adjacency "weights" between location i and location j (where we require $w_{ii} = 0$ for all $i = 1, \ldots, n$). Thus, Moran's I statistic is simply a weighted form of the usual Pearson correlation coefficient, where the weights are the spatial proximity weights, and it takes values between -1 and 1. If (2) is positive, then neighboring values tend to have similar values, and if it is negative, then neighboring regions tend to have different values. In the following code, we take each day in our data set, compute the distances, form the weight matrix, and carry out Moran's I test using the function `Moran.I` from the package **ape**.

```
P <- list()                                       # init list
days <- c(1:13, 15:31)                            # set of days
for(i in seq_along(days)) {                       # for each day
  Tmax_day <- filter(Tmax_no_14,
                     day == days[i])              # filter by day
  station.dists <- Tmax_day %>%                   # take the data
    select(lon, lat) %>%                          # extract coords.
    dist() %>%                                    # comp. dists.
    as.matrix()                                   # conv. to matrix
  station.dists.inv <- 1/station.dists            # weight matrix
  diag(station.dists.inv) <- 0                    # 0 on diag
  P[[i]] <- Moran.I(Tmax_day$residuals,           # run Moran's I
                    station.dists.inv) %>%
            do.call("cbind", .)                   # conv. to df
}
```

The object `P` is a list of single-row data frames that can be collapsed into a single data frame by calling `do.call` and proceeding to row-bind the elements of each list item together. We print the first six records of the resulting data frame below.

```
do.call("rbind", P) %>% head()

##      observed   expected       sd p.value
## [1,]  0.27167 -0.0075758 0.012356       0
## [2,]  0.22641 -0.0075758 0.012369       0
## [3,]  0.21140 -0.0075758 0.012364       0
## [4,]  0.16265 -0.0075758 0.012389       0
## [5,]  0.25803 -0.0075758 0.012410       0
## [6,]  0.12167 -0.0075758 0.012272       0
```

The maximum p-value from the 30 tests is 8.04019×10^{-6}, which is very small. Since we are in a multiple-hypothesis setting, we need to control the familywise error rate and, for a level of significance α, reject the null hypothesis of no correlation only if the p-value is less than $c(\alpha)$ $(< \alpha)$, where $c(\cdot)$ is a correction function. In this case, even the very conservative Bonferroni correction (for which $c(\alpha) = \alpha/T$, where T is the number of time points) will result in rejecting the null hypothesis at each time point.

It is straightforward to extend Moran's I test to the spatio-temporal setting, as one need only extend the concept of "spatial distance" to "spatio-temporal distance." We are faced with the usual problem of how to appropriately scale time to make a Euclidean distance across space and time have a realistic interpretation. One way to do this is to fit a dependence model that allows for scaling in time, and subsequently scale time by an estimate of the scaling factor prior to computing the Euclidean distance. We shall work with one such model, which uses an anisotropic covariance function, in the next Lab. For now, as we did with IDW, we do not scale time and compute distances on the spatio-temporal domain (which happens to be reasonable for these data).

```
station.dists <- Tmax_no_14 %>%  # take the data
  select(lon, lat, day) %>%      # extract coordinates
  dist() %>%                     # compute distances
  as.matrix()                    # convert to matrix
```

We now need to compute the weights from the distances, set the diagonal to zero and call `Moran.I`.

```
station.dists.inv <- 1/station.dists
diag(station.dists.inv) <- 0
Moran.I(Tmax_no_14$residuals, station.dists.inv)$p.value

## [1] 0
```

Unsurprisingly, given what we saw when analyzing individual time slices, the p-value is very small, strongly suggesting that there is spatio-temporal dependence in the data.

When the data are regularly spaced in time, as is the case here, one may also look at the "temporal" residuals at some location and test for temporal correlation in these residuals using the Durbin–Watson test. For example, consider the maximum temperature (`Tmax`) residuals at stations 3810 and 3889.

```
TS1 <- filter(Tmax_no_14, id == 3810)$residuals
TS2 <- filter(Tmax_no_14, id == 3889)$residuals
```

These residuals can be easily plotted using base `R` graphics as follows.

```
par(mar=c(4, 4, 1, 1))
plot(TS1,                              # Station 3810 residuals
     xlab = "day of July 1993",
     ylab = "residuals (degF)",
     type = 'o', ylim = c(-8, 7))
lines(TS2,                             # Station 3889 residuals
      xlab = "day of July 1993",
      ylab = "residuals (degF)",
      type = 'o', col = "red")
```

Note that there is clear temporal correlation in the residuals; that is, residuals close to each other in time tend to be more similar than residuals further apart. It is also interesting to note that the residuals are correlated between the stations; that is, at the same time point, the residuals at both stations are more similar than at different time points. This is due to the spatial correlation in the residuals that was tested for above (these two stations happen to be quite close to each other; recall the previous image of the spatial residuals). One may also look at the *correlogram* (the empirical autocorrelation function) of the residuals by typing `acf(TS1)` and `acf(TS2)`, respectively. From these plots it can be clearly seen that there is significant lag-1 correlation in both these residual time series.

Now let us proceed with carrying out a Durbin–Watson test for the residuals at every station. This can be done using a `for` loop as we did with Moran's I test; however, we shall now introduce a more sophisticated way of carrying out multiple tests and predictions on groups of data within a data frame, using the packages **tidyr**, **purrr**, and **broom**.

In Lab 2.1 we investigated data wrangling techniques for putting data that are in a data frame into groups using `group_by`, and then we performed an operation on each of those groups using `summarise`. The grouped data frame returned by `group_by` is simply the original frame with each row associated with a group. A more elaborate representation of these data is in a *nested data frame*, where we have a data frame containing one row *for each group*. The "nested" property comes from the fact that we may have a data frame, conventionally under the field name "data," for each group. For example, if we group `Tmax_no_14` by `lon` and `lat`, we obtain the following first three records.

```
nested_Tmax_no_14 <- group_by(Tmax_no_14, lon, lat) %>% nest()
head(nested_Tmax_no_14, 3)

## # A tibble: 3 x 3
##     lon   lat data
##   <dbl> <dbl> <list>
## 1 -81.4  39.3 <tibble [30 x 16]>
## 2 -81.4  35.7 <tibble [30 x 16]>
## 3 -88.9  35.6 <tibble [30 x 16]>
```

Note the third column, `data`, which is a column of *tibbles* (which, for the purposes of this course, should be treated as sophisticated data frames). Next we define a function that takes the data frame associated with a single group, carries out the test (in this case the Durbin–Watson test), and returns the results. The function **`dwtest`** takes an `R` formula as the first argument and the data as the second argument. In this case, we test for autocorrelation in the residuals after removing a temporal (constant) trend and by using the formula `residuals ~ 1`.

```
dwtest_one_station <- function(data)
                          dwtest(residuals ~ 1, data = data)
```

Calling **`dwtest_one_station`** for the data in the first record will carry out the test at the first station, in the second record at the second station, and so on. For example,

```
dwtest_one_station(nested_Tmax_no_14$data[[1]])
```

carries out the Durbin–Watson test on the residuals at the first station.

To carry out the test on each record in the nested data frame, we use the function **`map`** from the package **purrr**. For example, the command

```
map(nested_Tmax_no_14$data, dwtest_one_station) %>% head()
```

shows the test results for the first six stations. These results can be assigned to another column within our nested data frame using **`mutate`**. These results are of class `htest` and not easy to analyze or visualize in their native form. We therefore use the function **`tidy`** from the package **broom** to extract the key information from the test (in this case the statistic, the p-value, the method, and the hypothesis) and put it into a data frame. For example,

```
dwtest_one_station_tidy <- nested_Tmax_no_14$data[[1]] %>%
                           dwtest_one_station() %>%
                           tidy()
```

tidies up the results at the first station. The first three columns of the returned data are

```
dwtest_one_station_tidy[, 1:3]

## # A tibble: 1 x 3
##   statistic p.value method
##       <dbl>   <dbl> <chr>
## 1     0.982 0.00122 Durbin-Watson test
```

To assign the test results to each record in the nested data frame as added fields (instead of as another data frame), we then use the function **unnest**. In summary, the code

```
Tmax_DW_no_14 <- nested_Tmax_no_14 %>%
    mutate(dwtest = map(data, dwtest_one_station)) %>%
    mutate(test_df = map(dwtest, tidy)) %>%
    unnest(test_df)
```

provides all the information we need. The first three records, excluding the last two columns, are

```
Tmax_DW_no_14 %>% select(-method, -alternative) %>% head(3)

## # A tibble: 3 x 6
##     lon   lat data              dwtest     statistic p.value
##   <dbl> <dbl> <list>            <list>         <dbl>   <dbl>
## 1 -81.4  39.3 <tibble [30 ~ <S3: hte~      0.982 1.22e-3
## 2 -81.4  35.7 <tibble [30 ~ <S3: hte~      0.921 5.86e-4
## 3 -88.9  35.6 <tibble [30 ~ <S3: hte~      1.59  1.29e-1
```

The proportion of p-values *below* the 5% level of significance divided by the number of tests (Bonferroni correction) is

```
mean(Tmax_DW_no_14$p.value < 0.05/nrow(Tmax_DW_no_14)) * 100

## [1] 21.8
```

This proportion of 21.8% is reasonably high and provides evidence that there is considerable temporal autocorrelation in the residuals, as expected.

Finally, we can also compute and visualize the empirical spatio-temporal semivariogram of the residuals. Recall that in Lab 2.1 we put the maximum temperature data in the NOAA data set into an `STFDF` object that we labeled `STObj3`. We now load these data and subset the month of July 1993.

```
data("STObj3", package = "STRbook")
STObj4 <- STObj3[, "1993-07-01::1993-07-31"]
```

All we need to do is merge `Tmax_no_14`, which contains the residuals, with the `STFDF` object `STObj4`, so that the empirical semivariogram of the residuals can be computed. This can be done quickly, and safely, using the function **left_join**.

```
STObj4@data <- left_join(STObj4@data, Tmax_no_14)
```

As in Lab 2.3, we now compute the empirical semivariogram with the function **variogram**.

```
vv <- variogram(object = residuals ~ 1, # fixed effect component
                data = STObj4,      # July data
                width = 80,         # spatial bin (80 km)
                cutoff = 1000,      # consider pts < 1000 km apart
                tlags = 0.01:6.01) # 0 days to 6 days
```

The command `plot(vv)` displays the empirical semivariogram of the residuals. This empirical semivariogram is clearly different from that of the data and has a lower sill, but it suggests that there is still spatial and temporal correlation in the residuals.

Predictions

Prediction from linear or generalized linear models in `R` is carried out using the function `predict`. As in Lab 3.1, we use the following prediction grid.

```
pred_grid <- expand.grid(lon = seq(-100, -80, length = 20),
                         lat = seq(32, 46, length = 20),
                         day = seq(4, 29, length = 6))
```

We require all the covariate values at all the prediction locations. Hence, the 12 basis functions need to be evaluated on this grid. As above, we do this by calling `eval_basis` and converting the result to a matrix, which we then attach to our prediction grid.

```
Spred <- eval_basis(basis = G,                         # basis functs
                s = pred_grid[,c("lon","lat")] %>%   # pred locs
                     as.matrix()) %>%            # conv. to matrix
     as.matrix()                                  # results as matrix
colnames(Spred) <- paste0("B", 1:ncol(Spred)) # assign col names
pred_grid <- cbind(pred_grid, Spred)           # attach to grid
```

Now that we have all the covariates in place, we can call `predict`. We supply `predict` with the model `Tmax_July_lm`, the prediction grid, and the argument `interval = "prediction"`, so that `predict` returns the prediction intervals.

```
linreg_pred <- predict(Tmax_July_lm,
                       newdata = pred_grid,
                       interval = "prediction")
```

When `predict` is called as above, it returns a matrix containing three columns with names `fit`, `lwr`, and `upr`, which contain the prediction and the lower and upper bounds of the 95% prediction interval, respectively. Since in this case the prediction interval is the prediction $\pm$ 1.96 times the prediction standard error, we can calculate the prediction standard error from the given interval as follows.

```r
## Assign prediction and prediction s.e. to the prediction grid
pred_grid$z_pred <- linreg_pred[,1]
pred_grid$z_err <- (linreg_pred[,3] - linreg_pred[,2]) / (2*1.96)
```

Plotting the prediction and prediction standard error proceeds in a straightforward fashion using **ggplot2**. This is left as an exercise for the reader.