Lab 2.3: Exploratory Data Analysis

In this Lab we carry out exploratory data analysis (EDA), which typically requires visualization techniques similar to those utilized in Lab 2.2. There are several ways in which to carry out EDA with spatio-temporal data; in this Lab we consider the construction and visualization of the empirical means and covariances, the use of empirical orthogonal functions and their associated principal component time series, semivariogram analysis, and spatio-temporal canonical correlation analysis.

For the first part of the Lab, as in Lab 2.2, we shall consider the daily maximum temperatures in the NOAA data set between May 1993 and September 1993 (inclusive). The packages we need are **CCA, dplyr, ggplot2, gstat, sp, spacetime, STRbook** and **tidyr**.

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

In order to ensure consistency of results and visualizations, we fix the seed to 1.

```
set.seed(1)
```

We now load the NOAA data set using the `data` command. To keep the data size manageable, we take a subset of it corresponding to the maximum daily temperatures in the months May–September 1993. As in Lab 2.2 we also add a new variable `t` which starts at 1 at the beginning of the data set and increases by 1 each day.

```
data("NOAA_df_1990", package = "STRbook")
Tmax <- filter(NOAA_df_1990,      # subset the data
               proc == "Tmax" &   # only max temperature
               month %in% 5:9 &   # May to September
               year == 1993)      # year of 1993
Tmax$t <- Tmax$julian - 728049    # create a new time variable
```

Empirical Spatial Means

The empirical spatial mean of our data is given by

$$\widehat{\boldsymbol{\mu}}_z \equiv \begin{bmatrix} \widehat{\mu}_z(\mathbf{s}_1) \\ \vdots \\ \widehat{\mu}_z(\mathbf{s}_m) \end{bmatrix} \equiv \frac{1}{T}\sum_{j=1}^{T} \mathbf{z}_{t_j}, \tag{1}$$

where $\mathbf{z}_{t_j} \equiv (z(\mathbf{s}_1; t_j), \ldots, z(\mathbf{s}_m; t_j))'$. The empirical spatial mean is a spatial quantity that can be stored in a new data frame that contains the spatial locations and the respective average maximum temperature at each location. These, and other data manipulations to follow, can be carried out easily using the tools we learned in Lab 2.1. We group by longitude and latitude, and then we compute the average maximum temperature at each of the separate longitude–latitude coordinates.

```
spat_av <- group_by(Tmax, lat, lon) %>%     # group by lon-lat
           summarise(mu_emp = mean(z))      # mean for each lon-lat
```

We can now plot the average maximum temperature per station and see how this varies according to longitude and latitude. Use **print** to display the following plots.

```
lat_means <- ggplot(spat_av) +
             geom_point(aes(lat, mu_emp)) +
             xlab("Latitude (deg)") +
             ylab("Maximum temperature (degF)") + theme_bw()

lon_means <- ggplot(spat_av) +
             geom_point(aes(lon, mu_emp)) +
             xlab("Longitude (deg)") +
             ylab("Maximum temperature (degF)") + theme_bw()
```

Empirical Temporal Means

The empirical temporal mean can be computed easily using the tools we learned in Lab 2.1: first, group the data by time; and second, summarize using the **summarise** function.

```
Tmax_av <- group_by(Tmax, date) %>%
           summarise(meanTmax = mean(z))
```

The variable `Tmax_av` is a data frame containing the average maximum temperature on each day (averaged across all the stations). This can be visualized easily, together with the original raw data, using **ggplot2**.

```
gTmaxav <-
    ggplot() +
    geom_line(data = Tmax,aes(x = date, y = z, group = id),
              colour = "blue", alpha = 0.04) +
    geom_line(data = Tmax_av, aes(x = date, y = meanTmax)) +
    xlab("Month") + ylab("Maximum temperature (degF)") +
    theme_bw()
```

Empirical Covariances

Before obtaining the empirical covariances, it is important that all trends are removed (not just the intercept). One simple way to do this is to first fit a linear model (that has spatial and/or temporal covariates) to the data. Then plot the empirical covariances of the detrended data (i.e., the residuals). Linear-model fitting proceeds with use of the `lm` function in `R`. The residuals from `lm` can then be incorporated into the original data frame `Tmax`.

In the above visualizations we observed a quadratic tendency of temperature over the chosen time span. Therefore, in what follows, we consider time and time squared as covariates. Note the use of the function `I`. This is required for `R` to interpret the power sign "^" as an arithmetic operator instead of a formula operator.

```
lm1 <- lm(z ~ lat + t + I(t^2), data = Tmax) # fit a linear model
Tmax$residuals <- residuals(lm1)             # store the residuals
```

We also need to consider the spatial locations of the stations, which we extract from `Tmax` used above.

```
spat_df <- filter(Tmax, t == 1) %>% # lon/lat coords of stations
            select(lon, lat)  %>%   # select lon/lat only
            arrange(lon, lat)       # sort ascending by lon/lat
m <- nrow(spat_av)                  # number of stations
```

The most straightforward way to compute the empirical covariance matrix

$$\widehat{\mathbf{C}}_z^{(\tau)} \equiv \frac{1}{T-\tau} \sum_{j=\tau+1}^{T} (\mathbf{z}_{t_j} - \widehat{\boldsymbol{\mu}}_z)(\mathbf{z}_{t_j-\tau} - \widehat{\boldsymbol{\mu}}_z)'; \quad \tau = 0, 1, \ldots, T-1, \tag{2}$$

is using the `cov` function in `R`. When there are missing data, the usual way forward is to drop all records that are not complete (provided there are not too many of these). Specifically, if any of the elements in $\mathbf{Z}_{t_j}$ or $\mathbf{Z}_{t_j-\tau}$ are missing, the associated term in the summation of (2) is ignored altogether. The function `cov` implements this when the argument `use = 'complete.obs'` is supplied. If there are too many records that are incomplete, imputation, or the consideration of only subsets of stations, might be required.

In order to compute the empirical covariance matrices, we first need to put the data into space-wide format using `spread`.

```
X <- select(Tmax, lon, lat, residuals, t) %>% # select columns
     spread(t, residuals) %>%                 # make time-wide
     select(-lon, -lat) %>%                   # drop coord info
     t()                                      # make space-wide
```

Now it is simply a matter of calling `cov(X, use = 'complete.obs')` for computing the lag-0 empirical covariance matrix. For the lag-1 empirical covariance matrix we

compute the covariance between the residuals from `X` excluding the first time point and `X` excluding the last time point.

```
Lag0_cov <- cov(X, use = 'complete.obs')
Lag1_cov <- cov(X[-1, ], X[-nrow(X),], use = 'complete.obs')
```

In practice, it is very hard to gain any intuition from these matrices, since points in a two-dimensional space do not have any specific ordering. One can, for example, order the stations by longitude and then plot the permuted spatial covariance matrix, but this works best when the domain of interest is rectangular with a longitude span that is much larger than the latitude span. In our case, with a roughly square domain, a workaround is to split the domain into either latitudinal or longitudinal strips, and then plot the spatial covariance matrix associated with each strip. In the following, we split the domain into four longitudinal strips (similar code can be used to generate latitudinal strips).

```
spat_df$n <- 1:nrow(spat_df)     # assign an index to each station
lim_lon <- range(spat_df$lon)    # range of lon coordinates
lon_strips <- seq(lim_lon[1],    # create 4 long. strip boundaries
                  lim_lon[2],
                  length = 5)
spat_df$lon_strip <- cut(spat_df$lon,      # bin the lon into
                         lon_strips,       # their respective bins
                         labels = FALSE,   # don't assign labels
                         include.lowest = TRUE) # include edges
```

The first six records of `spat_df` are:

```
head(spat_df)    # print the first 6 records of spat_df

##      lon  lat n lon_strip
## 1 -100.0 37.8 1         1
## 2  -99.8 36.3 2         1
## 3  -99.7 32.4 3         1
## 4  -99.1 35.0 4         1
## 5  -98.8 38.9 5         1
## 6  -98.5 34.0 6         1
```

Now that we know in which strip each station falls, we can subset the station data frame by strip and then sort the subsetted data frame by latitude. In **STRbook** we provide a function **`plot_cov_strips`** that takes an empirical covariance matrix `C` and a data frame in the same format as `spat_df`, and then plots the covariance matrix associated with each longitudinal strip. Plotting requires the package **fields**. We can plot the resulting lag-0 and lag-1 covariance matrices using the following code.

```
plot_cov_strips(Lag0_cov, spat_df)   # plot the lag-0 matrices
plot_cov_strips(Lag1_cov, spat_df)   # plot the lag-1 matrices
```

As expected (from the figures), the empirical spatial covariance matrices reveal the presence of spatial correlation in the residuals. The four lag-0 plots seem to be qualitatively similar, suggesting that there is no strong dependence on longitude. However, there is a dependence on latitude, and the spatial covariance appears to decrease with decreasing latitude. This dependence is a type of spatial *non-stationarity*, and such plots can be used to assess whether non-stationary spatio-temporal models are required or not.

Similar code can be used to generate spatial correlation (instead of covariance) image plots.

Semivariogram Analysis

From now on, in order to simplify computations, we will use a subset of the data containing only observations in July. Computing the empirical semivariogram is much faster when using objects of class `STFDF` rather than `STIDF` since the regular space-time structure can be exploited. We hence take `STObj3` computed in Lab 2.1 (load using `data(STObj3)`) and subset the month of July 1993 as follows.

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

For computing the sample semivariogram we use the function `variogram`.[1] We bin the distances between measurement locations into bins of size 60 km, and consider at most six time lags.

```
vv <- variogram(object = z~1 + lat, # 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. The plot suggests that there are considerable spatio-temporal correlations in the data; spatio-temporal modeling of the residuals is thus warranted.

Empirical Orthogonal Functions

Empirical orthogonal functions (EOFs) can reveal spatial structure in the data and can also be used for subsequent dimensionality reduction. EOFs can be obtained from the data

[1] Although the function is named "variogram," it is in fact the sample semivariogram that is computed.

through either a spectral decomposition of the covariance matrix or a singular value decomposition (SVD) of the detrended space-time data matrix. The data matrix has to be in space-wide format (i.e., where space varies along the columns and time varies along the rows).

In this Lab we consider the approach using SVD. Let $\mathbf{Z} \equiv (\mathbf{z}_1, \ldots, \mathbf{z}_T)'$ be the $T \times m$ data matrix and then let $\widetilde{\mathbf{Z}}$ be the "detrended" and scaled data matrix,

$$\widetilde{\mathbf{Z}} \equiv \frac{1}{\sqrt{T-1}}(\mathbf{Z} - \mathbf{1}_T \widehat{\boldsymbol{\mu}}_z'), \tag{3}$$

where $\mathbf{1}_T$ is a T-dimensional vector of ones and $\widehat{\boldsymbol{\mu}}_z$ is the spatial mean vector given by (1). Now, consider the SVD of the detrended and scaled data matrix,

$$\widetilde{\mathbf{Z}} = \mathbf{U}\mathbf{D}\mathbf{V}', \tag{4}$$

where $\mathbf{U}$ is the $T \times T$ matrix of left singular vectors, $\mathbf{D}$ is a $T \times m$ matrix containing singular values on the main diagonal, and $\mathbf{V}$ is an $m \times m$ matrix containing the right singular vectors, where both $\mathbf{U}$ and $\mathbf{V}$ are orthonormal matrices. It can be shown (details omitted) that $\mathbf{V}$ contains the EOFs and the eigenvalues are $\boldsymbol{\Lambda} = \mathbf{D}'\mathbf{D}$. In addition, it is straightforward to show that the first m columns of $\mathbf{U}(\sqrt{T-1})$ correspond to the normalized PC time series.

For this part of the Lab we use the SST data set. The SST data set does not contain any missing values, which renders our task slightly easier than when data are missing. When data are missing, one typically needs to consider interpolation, median polishing, or other imputation methods to fill in the missing values prior to computing the EOFs.

First we load the sea-land mask, the lon-lat coordinates of the SST grid, and the SST data set itself which is in time-wide format.

```
data("SSTlandmask", package = "STRbook")
data("SSTlonlat", package = "STRbook")
data("SSTdata", package = "STRbook")
```

Since `SSTdata` contains readings over land,[2] we delete these using `SSTlandmask`. Further, in order to consider whole years only, we take the first 396 months (33 years) of the data, containing SST values spanning 1970–2002.

```
delete_rows <- which(SSTlandmask == 1)
SSTdata <- SSTdata[-delete_rows, 1:396]
```

From (3) recall that prior to carrying out an SVD, we need to put the data set into space-wide format, mean-correct it, and then standardize it. Since `SSTdata` is in time-wide format, we first transpose it to make it space-wide.

[2]The land SST data are "pseudo-data," and just there to help analysts re-grid the SST data to different resolutions.

```
## Put data into space-wide form
Z <- t(SSTdata)
dim(Z)

## [1]  396 2261
```

Note that `Z` is of size 396×2261, and it is hence in space-wide format as required. Equation (3) is implemented as follows.

```
## First find the matrix we need to subtract:
spat_mean <- apply(SSTdata, 1, mean)
nT <- ncol(SSTdata)

## Then subtract and standardize:
Zspat_detrend <- Z - outer(rep(1, nT), spat_mean)
Zt <- 1/sqrt(nT - 1)*Zspat_detrend
```

Finally, to carry out the SVD we run

```
E <- svd(Zt)
```

The SVD returns a list `E` containing the matrices $\mathbf{V}$, $\mathbf{U}$, and the singular values $\mathrm{diag}(\mathbf{D})$. The matrix $\mathbf{V}$ contains the EOFs in space-wide format. We change the column names of this matrix, and append the lon-lat coordinates to it as follows.

```
V <- E$v
colnames(E$v) <- paste0("EOF", 1:ncol(SSTdata)) # label columns
EOFs <- cbind(SSTlonlat[-delete_rows, ], E$v)
head(EOFs[, 1:6])

##    lon lat      EOF1     EOF2    EOF3     EOF4
## 16 154 -29 -0.004915 -0.01213 -0.0288 8.54e-05
## 17 156 -29 -0.001412 -0.00228 -0.0255 6.73e-03
## 18 158 -29  0.000246  0.00230 -0.0193 8.59e-03
## 19 160 -29  0.001455  0.00230 -0.0191 1.03e-02
## 20 162 -29  0.002266  0.00164 -0.0225 1.13e-02
## 21 164 -29  0.003599  0.00391 -0.0231 1.00e-02
```

The matrix $\mathbf{U}$ returned from `svd` contains the principal component time series in wide-table format (i.e., each column corresponds to a time series associated with an EOF). Here we use the function `gather` in the package **tidyr** that reverses the operation `spread`. That is, the function takes a spatio-temporal data set in wide-table format and puts it into long-table format. We instruct the function to gather every column except the column denoting time, and we assign the key–value pair `EOF-PC`:

```
TS <- data.frame(E$u) %>%                # convert U to data frame
      mutate(t = 1:nrow(E$u)) %>%        # add a time field
      gather(EOF, PC, -t)                # put columns (except time)
                                         # into long-table format with
                                         # EOF-PC as key-value pair
```

Finally, the normalized time series are given by:

```
TS$nPC <- TS$PC * sqrt(nT-1)
```

We now can use the visualization tools discussed earlier to visualize the EOFs and the (normalized) principal component time series during July 2003. We can use the following code to illustrate the first EOF:

```
ggplot(EOFs) + geom_tile(aes(x = lon, y = lat, fill = EOF1)) +
    fill_scale(name = "degC") + theme_bw() +
    xlab("Longitude (deg)") + ylab("Latitude (deg)")
```

Plotting of other EOFs and principal component time series is left as an excercise to the reader. The EOFs reveal interesting spatial structure in the residuals. The second EOF is a west–east gradient, while the third EOF again reveals a temporally dependent north–south gradient. This north–south gradient has a lower effect in the initial part of the time series, and a higher effect towards the end.

EOFs can also be constructed by using `eof` in the package **spacetime**. With the latter, one must cast the data into an `STFDF` object using the function `stConstruct` before calling the function `eof`. The last example in the help file of `stConstruct` shows how one can do this from a space-wide matrix. The function `eof` uses `prcomp` (short for principal component analysis) to find the EOFs, which in turn uses `svd`.

Spatio-Temporal Canonical Correlation Analysis

We can carry out a canonical correlation analysis (CCA) using the package **CCA** in `R`. One cannot implement CCA on the raw data since $T < n$. Instead we carry out CCA on the SST projected onto EOF space, specifically the first 10 EOFs which explain just over 74% of the variance of the signal (you can show this from the singular values in the object `E`). In this example we consider the problem of long-lead prediction, and we check whether SST is a useful predictor for SST in 7 months' time. To this end, we split the data set into two parts, one containing SST and another containing SST lagged by 7 months.

```
nEOF <- 10
EOFset1 <- E$u[1:(nT-7), 1:nEOF] * sqrt(nT - 1)
EOFset2 <- E$u[8:nT, 1:nEOF] * sqrt(nT - 1)
```

The CCA is carried out by running the function **cancor**.

```
cc <- cancor(EOFset1, EOFset2)   # compute CCA
options(digits = 3)              # print to three d.p.
print(cc$cor[1:5])               # print

## [1] 0.843 0.758 0.649 0.584 0.463

print(cc$cor[6:10])

## [1] 0.4137 0.3067 0.2058 0.0700 0.0273
```

The returned quantity `cc$cor` provides the correlations between the canonical variates of the unshifted and shifted SSTs in EOF space. The correlations decrease, as expected, but the first two canonical variates are highly correlated. The time series of the first canonical variables can be found by multiplying the EOF weights with the computed coefficients as follows

```
CCA_df <- data.frame(t = 1:(nT - 7),
                     CCAvar1 = (EOFset1 %*% cc$xcoef[,1])[,1],
                     CCAvar2 = (EOFset2 %*% cc$ycoef[,1])[,1])
```

A plot can be made using standard **ggplot2** commands.

```
t_breaks <- seq(1, nT, by = 60)      # breaks for x-labels
year_breaks <- seq(1970,2002,by=5)   # labels for x-axis
g <- ggplot(CCA_df) +
     geom_line(aes(t, CCAvar1), col = "dark blue") +
     geom_line(aes(t, CCAvar2), col = "dark red") +
     scale_x_continuous(breaks = t_breaks, labels = year_breaks) +
     ylab("CCA variables") + xlab("Year") + theme_bw()
```

The plot shows a high correlation between the first pair of canonical variables. What are these canonical variables? They are simply a linear combination of the EOFs, where the linear weights are given in `cc$xcoef[,1]` and `cc$ycoef[,1]`, respectively.

```
EOFs_CCA <- EOFs[,1:4] # first two columns are lon-lat
EOFs_CCA[,3] <- c(as.matrix(EOFs[,3:12]) %*% cc$xcoef[,1])
EOFs_CCA[,4] <- c(as.matrix(EOFs[,3:12]) %*% cc$ycoef[,1])
```

Plotting of the weights as spatial maps is straightforward and left as an exercise.