Lab 2.2: Visualization

In this Lab we shall visualize maximum temperature data in the NOAA data set. Specifically, we consider the maximum recorded temperature between May 1993 and September 1993 (inclusive). The packages we need are **animation**, **dplyr**, **ggplot2**, **gstat**, **maps**, and **STRbook**.

```
library("animation")
library("dplyr")
library("ggplot2")
library("gstat")
library("maps")
library("STRbook")
```

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

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

We now load the data set and take a subset of it using the function `filter`.

```
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
```

The data frame we shall work with is hence denoted by `Tmax`. The first six records in `Tmax` are:

```
Tmax %>% select(lon, lat, date, julian, z) %>% head()

##         lon   lat       date julian  z
## 1 -81.43333 39.35 1993-05-01 728050 82
## 2 -81.43333 39.35 1993-05-02 728051 84
## 3 -81.43333 39.35 1993-05-03 728052 79
## 4 -81.43333 39.35 1993-05-04 728053 72
## 5 -81.43333 39.35 1993-05-05 728054 73
## 6 -81.43333 39.35 1993-05-06 728055 78
```

The first record has a Julian date of 728050, corresponding to 01 May 1993. To ease the following operations, we create a new variable `t` that is equal to 1 when `julian == 728050` and increases by 1 for each day in the record.

```
Tmax$t <- Tmax$julian - 728049      # create a new time variable
```

The first task faced by the spatio-temporal modeler is data visualization. This is an important preliminary task that needs to be carried out prior to the exploratory-data-analysis stage and the modeling stages. Throughout, we shall make extensive use of the *grammar of graphics* package **ggplot2**, which is a convenient way to plot and visualize data and results in R. The book by Wickham (2016) provides a comprehensive introduction to **ggplot2**.

Spatial Plots

Visualization techniques vary with the data being analyzed. The NOAA data are collected at stations that are fixed in space; therefore, initial plots should give the modeler an idea of the overall spatial variation of the observed data. If there are many time points, usually only a selection of time points are chosen for visualization. In this case we choose three time points.

```
Tmax_1 <- subset(Tmax, t %in% c(1, 15, 30))   # extract data
```

The variable `Tmax_1` contains the data associated with the first, fifteenth, and thirtieth day in `Tmax`. We now plot this data subset using **ggplot2**. Note that the function **`col_scale`**, below, is simply a wrapper for the **ggplot2** function **`scale_colour_distiller`**, and is provided with **STRbook**.

```
NOAA_plot <- ggplot(Tmax_1) +                  # plot points
    geom_point(aes(x = lon,y = lat,            # lon and lat
                   colour = z),                # attribute color
               size = 2) +                     # make all points larger
    col_scale(name = "degF") +                 # attach color scale
    xlab("Longitude (deg)") +                  # x-axis label
    ylab("Latitude (deg)") +                   # y-axis label
    geom_path(data = map_data("state"),        # add US states map
          aes(x = long, y = lat, group = group)) +
    facet_grid(~date) +                        # facet by time
    coord_fixed(xlim = c(-105, -75),
                ylim = c(25, 50))  +           # zoom in
    theme_bw()                                 # B&W theme
```

`NOAA_plot` is a plot of the spatial locations of the stations. The function **`aes`** (short for aesthetics) for **`geom_point`** identifies which field in the data frame `Tmax_1` is the x-coordinate and which is the y-coordinate. **ggplot2** also allows one to attribute color (and size, if desired) to other fields in a similar fashion. Use the command **`print`**`(NOAA_plot)` to display the plot. As can be seen, the stations are approximately regularly spaced within the domain.

When working with geographic data, it is also good practice to put the spatial locations of the data into perspective, by plotting country or state boundaries together with the data locations. Above, the US state boundaries are obtained from the **maps** package through the command `map_data("state")`. The boundaries are then overlayed on the plot using `geom_path`, which simply joins the points and draws the resulting path with `x` against `y`. Projections can be applied by adding another layer to the **ggplot2** object using `coord_map`. For example adding `+ coord_map(projection = "sinusoidal")` will plot using a sinusoidal projection. One can also plot in three dimensions by using `projection = "ortho"`.

In this example we have used **ggplot2** to plot *point-referenced data*. Plots of regular lattice data are generated similarly by using `geom_tile` instead. Plots of irregular lattice data are generated using `geom_polygon`. As an example of the latter, consider the BEA income data set. These data can be loaded from **STRbook** as follows.

```
data("BEA", package = "STRbook")
head(BEA %>% select(-Description), 3)

##           NAME10 X1970 X1980 X1990
## 6     Adair, MO  2723  7399 12755
## 9    Andrew, MO  3577  7937 15059
## 12 Atchison, MO  3770  5743 14748
```

From the first three records, we can see that the data set contains the personal income, in dollars, by county and by year for the years 1970, 1980 and 1990. These data need to be merged with Missouri county data which contain geospatial information. These county data, which are also available in **STRbook**, were originally processed from a shapefile that was freely available online.[1]

```
data("MOcounties", package = "STRbook")
head(MOcounties %>% select(long, lat, NAME10), 3)

##       long     lat    NAME10
## 1 627911.9 4473554 Clark, MO
## 2 627921.4 4473559 Clark, MO
## 3 627923.0 4473560 Clark, MO
```

The data set contains the boundary points for the counties, amongst several other variables which we do not explore here. For example, to plot the boundary of the first county one can simply type

[1] http://msdis-archive.missouri.edu/archive/metadata_gos/MO_2010_TIGER_Census_County_Boundaries.xml

```
County1 <- filter(MOcounties, NAME10 == "Clark, MO")
plot(County1$long, County1$lat)
```

To add the BEA income data to the county data containing geospatial information we use **left_join**.

```
MOcounties <- left_join(MOcounties, BEA, by = "NAME10")
```

Now it is just a matter of calling **ggplot** with **geom_polygon** to display the BEA income data as spatial polygons. We also use **geom_path** to draw the county boundaries. Below we show the code for 1970; similar code would be needed for 1980 and 1990. Note the use of the `group` argument to identify which points correspond to which county.

```
g1 <- ggplot(MOcounties) +
    geom_polygon(aes(x = long, y = lat,      # county boundary
                     group = NAME10,         # county group
                     fill = log(X1970))) +   # log of income
    geom_path(aes(x = long, y = lat,         # county boundary
                  group = NAME10)) +         # county group
    fill_scale(limits = c(7.5,10.2),
               name = "log($)")  +
    coord_fixed() + ggtitle("1970") +        # annotations
    xlab("x (m)") + ylab("y (m)") + theme_bw()
```

Type **print**`(g1)` in the `R` console to display the plot.

Time-Series Plots

Next, we look at the time series associated with the maximum temperature data in the NOAA data set. One can plot the time series at all 139 weather stations (and this is recommended); here we look at the time series at a set of stations selected at random. We first obtain the set of unique station identifiers, choose 10 at random from these, and extract the data associated with these 10 stations from the data set.

```
UIDs <- unique(Tmax$id)                      # extract IDs
UIDs_sub <- sample(UIDs, 10)                 # sample 10 IDs
Tmax_sub <- filter(Tmax, id %in% UIDs_sub)   # subset data
```

To visualize the time series at these stations, we use *facets*. When given a long data frame, one can first subdivide the data frame into groups and generate a plot for each group. The following code displays the time series at each station. The command we use is **facet_wrap**, which automatically adjusts the number of rows and columns in which to display the facets. The command **facet_grid** instead uses columns for one grouping variable and rows for a second grouping variable, if specified.

```
TmaxTS <- ggplot(Tmax_sub) +
    geom_line(aes(x = t, y = z)) + # line plot of z against t
    facet_wrap(~id, ncol = 5) +    # facet by station
    xlab("Day number (days)") +    # x label
    ylab("Tmax (degF)") +          # y label
    theme_bw() +                   # BW theme
    theme(panel.spacing = unit(1, "lines")) # facet spacing
```

The argument `~id` supplied to **`facet_wrap`** is a `formula` in `R`. In this case, the formula is used to denote the groups by which we are faceting. The syntax `x~y` can be used to facet by two variables. The command **`print`**`(TmaxTS)` produces the required figure.

Hovmöller Plots

A Hovmöller plot is a two-dimensional space-time visualization, where space is collapsed (projected or averaged) onto one dimension; the second dimension then denotes time. A Hovmöller plot can be generated relatively easily if the data are on a space-time grid, but unfortunately this is rarely the case! This is where data-wrangling techniques such as those explored in Lab 2.1 come in handy.

Consider the latitudinal Hovmöller plot. The first step is to generate a regular grid of, say, 25 spatial points and 100 temporal points using the function **`expand.grid`**, with limits set to the latitudinal and temporal limits available in the data set.

```
lim_lat <- range(Tmax$lat)         # latitude range
lim_t <- range(Tmax$t)             # time range
lat_axis <- seq(lim_lat[1],        # latitude axis
                lim_lat[2],
                length=25)
t_axis <- seq(lim_t[1],            # time axis
              lim_t[2],
              length=100)
lat_t_grid <- expand.grid(lat = lat_axis,
                          t = t_axis)
```

We next need to associate each station's latitudinal coordinate with the closest one on the grid. This can be done by finding the distance from the station's latitudinal coordinate to each point of the grid, finding which gridpoint is the closest, and allocating that to it. We store the gridded data in `Tmax_grid`.

```
Tmax_grid <- Tmax
dists <- abs(outer(Tmax$lat, lat_axis, "-"))
Tmax_grid$lat <- lat_axis[apply(dists, 1, which.min)]
```

Now that we have associated each station with a latitudinal coordinate, all that is left is to group by latitude and time, and then we average all station values falling in the latitude–time bands.

```
Tmax_lat_Hov <- group_by(Tmax_grid, lat, t) %>%
                summarise(z = mean(z))
```

In this case, every latitude–time band contains at least one data point, so that the Hovmöller plot contains no missing points on the established grid. This may not always be the case, and simple interpolation methods, such as `interp` from the **akima** package, can be used to fill out grid cells with no data.

Plotting gridded data is facilitated using the **ggplot2** function `geom_tile`. The function `geom_tile` is similar to `geom_point`, except that it assumes regularly spaced data and automatically uses rectangular patches in the plot. Since rectangular patches are "filled," we use the **STRbook** function `fill_scale` instead of `col_scale`, which takes the legend title in the argument `name`.

```
Hovmoller_lat <- ggplot(Tmax_lat_Hov) +             # take data
        geom_tile(aes(x = lat, y = t, fill = z)) + # plot
        fill_scale(name = "degF") +      # add color scale
        scale_y_reverse() +              # rev y scale
        ylab("Day number (days)") +      # add y label
        xlab("Latitude (degrees)") +     # add x label
        theme_bw()                       # change theme
```

The function `scale_y_reverse` ensures that time increases from top to bottom, as is typical in Hovmöller plots. We can generate a longitude-based Hovmöller plot in the same way. Type `print(Hovmoller_lat)` in the `R` console to display the plot.

Animations

To generate an animation in `R`, one can use the package **animation**. First, we define a function that plots a spatial map of the maximum temperature as a function of time:

```
Tmax_t <- function(tau) {
    Tmax_sub <- filter(Tmax, t == tau)         # subset data
    ggplot(Tmax_sub) +
        geom_point(aes(x = lon,y = lat, colour = z),    # plot
                   size = 4) +                          # pt. size
        col_scale(name = "z", limits = c(40, 110)) +
        theme_bw() # B&W theme
}
```

The function above takes a day number `tau`, filters the data frame according to the day number, and then plots the maximum temperature at the stations as a spatial map.

Next, we construct a function that plots the data for every day in the data set. The function that generates the animation within an HTML webpage is **saveHTML**. This takes the function that plots the sequence of images and embeds them in a webpage (by default named `index.html`) using JavaScript. The function **saveHTML** takes many arguments; type the command

```
help(saveHTML)
```

in the `R` console for more details.

```
gen_anim <- function() {
    for(t in lim_t[1]:lim_t[2]){   # for each time point
       plot(Tmax_t(t))             # plot data at this time point
    }
}

ani.options(interval = 0.2)      # 0.2s interval between frames
saveHTML(gen_anim(),             # run the main function
         autoplay = FALSE,       # do not play on load
         loop = FALSE,           # do not loop
         verbose = FALSE,        # no verbose
         outdir = ".",           # save to current dir
         single.opts = "'controls': ['first', 'previous',
                                     'play', 'next', 'last',
                                      'loop', 'speed'],
                                      'delayMin': 0",
         htmlfile = "NOAA_anim.html")  # save filename
```

To view the animation, load `NOAA_anim.html` from your working directory. The animation reveals dynamics within the spatio-temporal data that are not apparent using other visualization methods. For example, the maximum temperature clearly drifts from west to east at several points during the animation. This suggests that a dynamic spatio-temporal model that can capture this drift could provide a good fit to these data.

Bibliography

Wickham, H. (2016), *ggplot2: Elegant Graphics for Data Analysis*, 2nd ed., New York: Springer.