Applied geostatistics
Exercise C: Spatio-temporal Geostatistics

D G Rossiter

May 18, 2020

Contents

山经之蹊间，介然用之而成路，
为间不用，则茅塞之矣。今茅塞子之心矣。
— 孟子卷十四 尽心下

"A footpath through the mountains if frequently used becomes a road; if not used, it quickly becomes choked with weeds. Now the weeds are choking your mind."
– Mencius, Jin Xin part 2, 67

1 Introduction

Spatio-temporal observations are those for which both a **spatial location** (georeference) and a **time of observation** are recorded, as well as **attributes measured at the specified location and time**.

Spatio-temporal datasets are found in many fields, for example climatology (time-series of weather observations at several stations), politics (voting records over time for a set of election districts), epidemiology (disease occurrence over time in reporting districts or presented to hospitals), wildlife ecology (locations of animals over time), soil science (monitoring of changing soil properties in a region), limnology and hydrology (repeated measurements at monitoring locations).

This exercise introduces the geostatistical analysis of spatio-temporal point observations at known locations and at known times, where both the location and time are important in the analysis, within the R environment for statistical computing. The exercise is inspired by an article by Pebesma [13], and uses his `spacetime` R package. This package extends the `sp` package for spatial data handling and the `xts` package for time series, thus combining spatial and temporal data structures.

At any **one location**, a time-series of observations can be analyzed using time-series analysis [e.g., 15]; at any **one time** a set of point observations can be analyzed using point geostatistics [e.g., 7, 16]; a set of area observations at one time can be analyzed with area-based spatial data analysis [e.g., 1, Chapter 9]. But when observations are located in **both space and time**, combined methods are needed. A good text introducing these concepts is by Cressie and Wikle [3]. The technical report of Gneiting et al. [6] is at a high mathematical level but has a worked example of wind velocity data from Ireland. Kyriakidis and Journel [11] review the different geostatistical models applied to space-time point data.

Pebesma has written a useful R "Task View" on spatio-temporal analysis[1] which explains the alternative approaches than can be taken.

> **Note:** The code in these exercises was tested with `knitr` package Version: 1.28 [17] on R version 3.6.3 (2020-02-29), `sp` package Version: 1.4-1, `spacetime` package Version: 1.2-3, `gstat` package Version: 2.0-5 and `xts` package Version: 0.12-0 running on Mac OS X 10.7.5. So, the text and graphical output you see here was automatically generated and incorporated into LaTeX by running the code through R and its packages. Then

[1] http://cran.r-project.org/web/views/SpatioTemporal.html

the LaTeX document was compiled into the PDF version you are now reading. Your output may be slightly different on different versions and on different platforms.

Most of this exercise (§2.1 – §11) uses an example space-time object, already in proper format, to illustrate space-time data structures and space-time geostatistical analysis. A final section (§12) presents empirical orthogonal functions for analyzing space-time structure. An appendix (§A) shows an example of building space-time objects from data frames, and how such data frames can be built from public databases (§B).

2 Space-time objects

TASK 1: Load the required packages. •

The `spacetime` package defines object classes inheriting from both `sp` classes for the spatial representation and `xts` classes for the time representation. We use the `gstat` package for point geostatistics. These are all loaded with the `require` function.

```
library(sp)
library(xts)
library(spacetime)
library(gstat)
```

TASK 2: Load an example dataset, list its objects and their classes. •

The `air` dataset provided with `spacetime` is a spatio-temporal dataset of air quality in rural areas over much of Germany from the European Air Quality Database ("AirBase")[2]; see `?air` for details. The `data` function loads a built-in R dataset from a loaded package into the workspace.

```
data("air")
ls()

[1] "air"      "dates"    "DE"       "DE_NUTS1" "stations"

class(air)

[1] "matrix"

str(air)

 num [1:70, 1:4383] NA NA NA NA NA NA NA NA NA NA ...
 - attr(*, "dimnames")=List of 2
  ..$ : chr [1:70] "DESH001" "DENI063" "DEUB038" "DEBE056" ...
  ..$ : NULL

dim(air)["space"]

space
   70

rownames(air)
```

[2] https://acm.eionet.europa.eu/databases/airbase

```
 [1] "DESH001" "DENI063" "DEUB038" "DEBE056" "DEBE062" "DEBE032"
 [7] "DEHE046" "DEUB007" "DENW081" "DESH008" "DEUB003" "DESN049"
[13] "DESN076" "DEUB002" "DETH026" "DENI059" "DEUB039" "DEHE028"
[19] "DEMV017" "DEMV004" "DEBB053" "DEUB034" "DENW063" "DETH061"
[25] "DERP014" "DEHE048" "DEUB035" "DEUB032" "DEMV012" "DEUB031"
[31] "DEUB033" "DEBY047" "DENW065" "DEUB030" "DEHE034" "DESL008"
[37] "DEBW103" "DENI058" "DEBB056" "DERP017" "DETH042" "DEBB075"
[43] "DESN051" "DEUB041" "DEUB017" "DEHE043" "DEUB004" "DEUB029"
[49] "DEUB040" "DESN074" "DEBW031" "DEBW087" "DEMV001" "DENW064"
[55] "DENW068" "DENI019" "DEUB026" "DEUB005" "DEBB051" "DEHE051"
[61] "DEBW030" "DENI060" "DERP015" "DEUB001" "DERP016" "DERP013"
[67] "DENI051" "DEUB028" "DESN052" "DEUB042"

dim(air)["time"]

time
4383

str(dates)

 Date[1:4383], format: "1998-01-01" "1998-01-02" "1998-01-03" "1998-01-04" ...

summary(stations)

Object of class SpatialPoints
Coordinates:
              min      max
coords.x1  6.28107 14.78617
coords.x2 47.80847 54.92497
Is projected: FALSE
proj4string : [+proj=longlat +datum=WGS84]
Number of points: 70

summary(DE_NUTS1)

Object of class SpatialPolygonsDataFrame
Coordinates:
         min      max
r1  5.871619 15.03811
r2 47.269858 55.05653
Is projected: FALSE
proj4string : [+proj=longlat +datum=WGS84]
Data attributes:
      ID_0     ISO         NAME_0        ID_1
 Min.   :60   DEU:16   Germany:16   Min.   :753.0
 1st Qu.:60                         1st Qu.:756.8
 Median :60                         Median :760.5
 Mean   :60                         Mean   :760.5
 3rd Qu.:60                         3rd Qu.:764.2
 Max.   :60                         Max.   :768.0

    NAME_1                              VARNAME_1 NL_NAME_1
 Length:16          Bavaria                  :1   NA's:16
 Class :character   Hesse                    :1
 Mode  :character   Lower Saxony             :1
                    Mecklenburg-West Pomerania:1
                    North Rhine-Westphalia   :1
                    (Other)                  :4
                    NA's                     :7
     HASC_1     CC_1     TYPE_1   ENGTYPE_1    VALIDFR_1
 DE.BE  : 1   NA's:16   Land:16   State:16   Unknown:16
 DE.BR  : 1
 DE.BW  : 1
 DE.BY  : 1
 DE.HB  : 1
 DE.HE  : 1
 (Other):10
   VALIDTO_1  REMARKS_1   Shape_Leng       Shape_Area
 Present:16   NA's:16   Min.   : 1.588   Min.   :0.04279
```

```
                    1st Qu.:10.710   1st Qu.:1.63730
                    Median :15.235   Median :2.57381
                    Mean   :15.809   Mean   :2.86920
                    3rd Qu.:20.373   3rd Qu.:4.01785
                    Max.   :32.255   Max.   :8.65611
```

This dataset has several dataframes:

air : matrix of 70 rows, i.e., observation stations, with names given as `rownames(air)`, and 4383 columns, i.e., air quality measurements;

stations : coördinates of the stations; no other information; this is of class `SpatialPoints`, i.e., there is no `data.frame` associated with the points.

dates : the dates of observations, one for each column in the `air` matrix; this is of class `Date`.

DE_NUTS1 : polygons of German states (1st level divisions); this is of class `SpatialPolygonsDataFrame`.

Both of the spatially-explicit dataset: `stations` (a `SpatialPoints` object) and `DE_NUTS1` (a `SpatialPolygonsDataFrame` object) have a coördinate reference system (CRS) that is not explicit enough for recent developments in the underlying `PROJ.6` specification. These need to be updated to include information on the ellipsoid used within the WGS84 datum. We do this with a call to the `CRS` function.

```
proj4string(stations)

[1] "+proj=longlat +datum=WGS84"

proj4string(DE_NUTS1)

[1] "+proj=longlat +datum=WGS84"

proj4string(stations) <- CRS(proj4string(stations))

Warning in `proj4string<-`(`*tmp*`, value = new("CRS", projargs = "+proj=longlat
+datum=WGS84 +ellps=WGS84 +towgs84=0,0,0")):  A new CRS was assigned to an object
with an existing CRS:
+proj=longlat +datum=WGS84
without reprojecting.
For reprojection, use function spTransform

proj4string(DE_NUTS1) <- CRS(proj4string(DE_NUTS1))

Warning in `proj4string<-`(`*tmp*`, value = new("CRS", projargs = "+proj=longlat
+datum=WGS84 +ellps=WGS84 +towgs84=0,0,0")):  A new CRS was assigned to an object
with an existing CRS:
+proj=longlat +datum=WGS84
without reprojecting.
For reprojection, use function spTransform

proj4string(stations)

[1] "+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0"

proj4string(DE_NUTS1)

[1] "+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0"
```

Notice the warning "A new CRS was assigned to an object with an existing CRS". But in this case, we have no more specific information on the ellipsoid, so we assume the WGS84 ellipsoid along with the datum.

The first job is to combine the spatial and temporal information into one data structure, a `STFDF` "space-time full-grid data frame" object, using the `STFDF` constructor. of the `spacetime` package. This requires:

1. The spatial points, here from the `stations` object;
2. The times of observation, here from the `dates` object;
3. The variable measured at each point and each time, here the `air` matrix, which we give a more meaningful name, `PM10`.

The number of spatial points and dates must match the number of rows and columns, respectively, in the matrix,

We name the object `rural`; these are rural stations.

```
rural <- STFDF(stations, dates, data.frame(PM10 = as.vector(air)))
```

The `class` function shows the class of a object in the workspace.

```
class(DE_NUTS1)

[1] "SpatialPolygonsDataFrame"
attr(,"package")
[1] "sp"

class(rural)

[1] "STFDF"
attr(,"package")
[1] "spacetime"
```

Q1 : *What are the classes of these two objects?* *Jump to A1* •

In this exercise we will ignore the polygons and work only with point data, i.e., the `rural` object.

2.1 Examining the structure of the dataset

TASK 3 : Display the structure of the `rural` object. •

This object structure allows space-time analysis. It is an object in the S4 class definition system [2], with four first-level **slots**. The slot names can be listed with the `slotNames` function, and of course the structure with the `str` function:

```
str(rural)

Formal class 'STFDF' [package "spacetime"] with 4 slots
  ..@ data   :'data.frame': 306810 obs. of  1 variable:
  .. ..$ PM10: num [1:306810] NA NA NA NA NA NA NA NA NA NA ...
  ..@ sp     :Formal class 'SpatialPoints' [package "sp"] with 3 slots
  .. .. ..@ coords     : num [1:70, 1:2] 9.59 9.69 9.79 13.65 13.3 ...
  .. .. .. ..- attr(*, "dimnames")=List of 2
  .. .. .. .. ..$ : chr [1:70] "DESH001" "DENI063" "DEUB038" "DEBE056" ...
```

```
  .. .. .. .. ..$ : chr [1:2] "coords.x1" "coords.x2"
  .. .. ..@ bbox       : num [1:2, 1:2] 6.28 47.81 14.79 54.92
  .. .. .. ..- attr(*, "dimnames")=List of 2
  .. .. .. .. ..$ : chr [1:2] "coords.x1" "coords.x2"
  .. .. .. .. ..$ : chr [1:2] "min" "max"
  .. .. ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
  .. .. .. .. ..@ projargs: chr "+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0"
  ..@ time   :An 'xts' object on 1998-01-01/2009-12-31 containing:
  Data: int [1:4383, 1] 1 2 3 4 5 6 7 8 9 10 ...
 - attr(*, "dimnames")=List of 2
  ..$ : NULL
  ..$ : chr "timeIndex"
  Indexed by objects of class: [Date] TZ: UTC
  xts Attributes:
 NULL
  ..@ endTime: POSIXct[1:4383], format: "1998-01-02" ...

slotNames(rural)

[1] "data"    "sp"      "time"    "endTime"

class(rural@data)

[1] "data.frame"

class(rural@sp)

[1] "SpatialPoints"
attr(,"package")
[1] "sp"

class(rural@time)

[1] "xts" "zoo"

class(rural@endTime)

[1] "POSIXct" "POSIXt"
```

The slots are:

@data : A dataframe with attribute values at each space-time point; here there is only only attribute, `PM10`.

@sp : a `SpatialPoints` S4 class object defined in the `sp` package, with the coördinates of each observation; it has second-level slots `coords` containing the point coördinates, `bbox` containing the extreme coördinates of the point set, and `proj4string` giving the coördinate reference system;

@time : an `xts` "extensible time-series" S3 class object defined in the `xts` "eXtensible Time Series" package;

@endTime : a vector whose elements are S3 class `POSIXct` objects, each specifying the end point of a time interval. `POSIXct` is a standard class for representing dates and times [14].

Note: The acronym POSIX stands for "Portable Operating System Interface", a family of standards specified by the IEEE Computer Society to ensure compatibility between operating systems. In this case the `ct` "continuous time" standard is a time format, the signed number of seconds since the beginning of 1970 in the UTC time zone.

Q2 : *What are the dimensions of the data frame, spatial points, and time-series? What is their relation? (Hint: look at the dimensions of these slots in the space-time object.)* *Jump to A2* •

Q3 : *In what coördinate reference system (CRS) is this dataset georeferenced? (Hint: look at the* `proj4string` *slot of the* `sp` *slot).* *Jump to A3* •

The description of this dataset[3] does not explain the meaning of the single variable, `PM10`. The abbreviation "PM" stands for "particulate matter", which are small particles of solid or liquid in the air. PM10 refers to aerosol particles smaller than $\approx 10\mu$m; these are particularly serious for human health, because they are too small for the nose and throat to filter, and so they may penetrate to the lungs and cause irritation; the smallest portion of these, known as PM2.5, can enter the gas exchange region of the lungs and even directly into the blood stream. PM10 is generally measured in μg m^{-3} of air; the US EPA national air quality standard[4] for PM10 is 50 μg m^{-3} measured as an annual mean and 150 μg m^{-3} measured as a daily concentration; German environmental standards[5] are stricter: 40 μg m^{-3} measured as an annual mean and 50 μg m^{-3} PM10 measured as a daily concentration, allowed to be exceeded 35 times per year.

2.2 Ensuring consistent time reference

There is a complication associated with slot `endtime`. It is an object of class `POSIXct` [14]; elements of this vector are the signed number of seconds since the beginning of 1970 in the UTC time zone. The `Sys.time` function shows the current time in this format, adjusted to the time zone set for the user's system. However, this time zone may not agree with that used in a data set. And, your operating system may not use an explicit time zone in the R settings, but somehow get it from the context.[6]

Task 4 : Determine the time zone settings of your system. •

We first examine your system's time zone using the `Sys.time` function to show the time and `Sys.timezone` to show the time zone as a character string, and the `get.env` "get environment variable" function to show the time zone as set in the R environment[7]. We compare these to the time in

[3] `help(air)`
[4] http://www.epa.gov/airtrends/aqtrnd95/pm10.html
[5] http://www.stadtentwicklung.berlin.de/umwelt/umweltatlas/ed312_01.htm
[6] For more on R date and time classes, see [8].
[7] In some systems the time zone is inherited from the operating system and not explicitly set in R.

UTC, by converting the time to an `POSIXlt` "long time" object with the `as.POSIXlt` function.

Your results may well be different from what is shown here:

```
Sys.time()

[1] "2020-05-19 00:18:08 CST"

as.POSIXlt(Sys.time(), "UTC")

[1] "2020-05-18 16:18:08 UTC"

Sys.timezone()

[1] "America/New_York"

Sys.getenv(x="TZ", unset=NA)

[1] "Asia/Shanghai"
```

If the result of the last function is `NA`, the time zone information has not been set in the R environment, but is inherited from the operating system.

The `rural` object from the `air` dataset uses dates from midnight UTC ("Universal Coördinated Time" in French), commonly referred to as GMT ("Greenwich Mean Time"). So when we display dates of events in `rural`, these are shown in the current time zone. To illustrate this, we show the date as it appears in `rural` in the current time zone, and then in some other time zones. Notice that with time zones to the west of Greenwich (England), the date displayed changes to a day earlier.

```
rural@endTime[1]

[1] "1998-01-02 UTC"

as.POSIXlt(rural@endTime[1], tz="CET")

[1] "1998-01-02 01:00:00 CET"

as.POSIXlt(rural@endTime[1], tz="EST")

[1] "1998-01-01 19:00:00 EST"

as.POSIXlt(rural@endTime[1], tz="America/Chicago")

[1] "1998-01-01 18:00:00 CST"

as.POSIXlt(rural@endTime[1], tz="Asia/Shanghai")

[1] "1998-01-02 08:00:00 CST"
```

> **Note:** Quite confusingly, both Central Standard Time (USA) and Chinese Standard Time (China) are referred to as `CST`.

> **Note:** To find time zone names, see `?timezones`; a long list of time zone names can be displayed with the `OlsonNames` function.

The easiest way to ensure that dates are properly extracted is to set the time zone for this R session to UTC.

TASK 5 : Set the time zone for this R session to UTC. •

The `Sys.setenv` function is used to set environment variables; the variable TZ is a string naming the time zone.

> **Note:** On Unix-like systems such as Mac OS/X and Linux, time zone names and region/city combinations which represent them, are generally found in the `/usr/share/zoneinfo/` directory.

```
Sys.setenv(TZ="UTC")
Sys.timezone()

[1] "America/New_York"

Sys.getenv(x="TZ", unset=NA)

[1] "UTC"

Sys.time()

[1] "2020-05-18 16:18:08 UTC"

rural@endTime[1]

[1] "1998-01-02 UTC"
```

Now the time zone is set explicitly in the R environment, and the system time zone agrees with the time zone used for the `rural` object.

2.3 Long-term time series

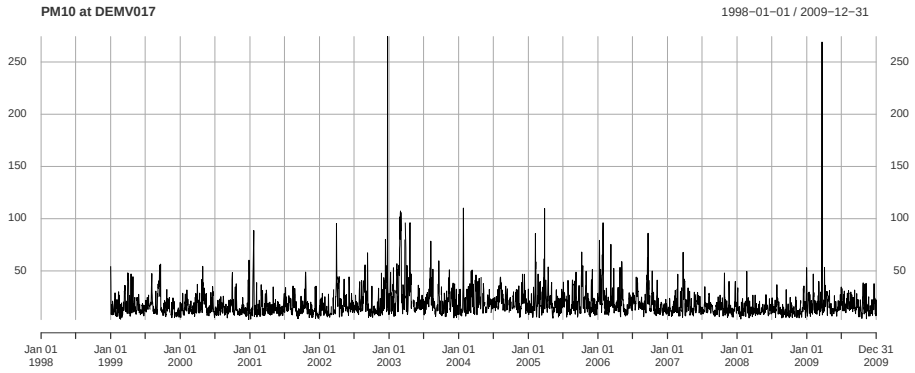
We can examine the time series for local structure (see §3), but first we examine the global structure, i.e., to see if there is any long-term trend or cycle.

TASK 6 : Find the longest time-series in the `rural` object. •

```
max.not.na <- 0; longest.station <- 1
for (station in 1:dim(rural)["space"]) {
    ts <- rural[station,][,"PM10"]
    (ix <- sum(!is.na(ts)))
    if (ix > max.not.na) { max.not.na <- ix; longest.station <- station }
    }
longest.station.name <- row.names(rural@sp@coords)[longest.station]
print(paste0("Station ", longest.station, " (",
             longest.station.name,
             ") has ", max.not.na, " PM10 readings"))

[1] "Station 19 (DEMV017) has 3940 PM10 readings"

long.ts <- rural[longest.station,]
rm(max.not.na, longest.station, station)
```

TASK 7 : Display the time-series of PM10 at this station. •

```
spacetime::plot(long.ts$PM10, lwd=0.5,
     main=paste("PM10 at",longest.station.name))
```



Q4 : *Describe this time series: (1) does there appear to be a long-term trend? (2) does there appear to be a periodic structure, and if so, what is the length of the period? (3) are there any obvious anomalies, and if so, when?* *Jump to A4* •

These correspond to three processes:

1. A long-term process that operates over the time spanned by the series;
2. A cyclic process that operates within each cycle;
3. A local process which causes variability between cycles.

Each of these processes is of interest and should be explained by the analyst.

This time series does not seem to have any long-term trend, nor any interesting cycle. The interpretation is that PM10 is a time-local phenomenon; we will investiate this below in §3.

2.4 Subsetting the data set

To examine local structure, we subset this object to work with a smaller dataset.

TASK 8 : Restrict the data to the time period from 2005 to 2010 inclusive. •

The `[]` matrix selection operator has been adapted for `STFDF` objects: the first dimension represents space, and the second time. We can see this by the class of objects selected by just the first and second dimensions:

```
class(rural[1,])

[1] "xts" "zoo"

class(rural[,1])

[1] "SpatialPointsDataFrame"
attr(,"package")
[1] "sp"
```

In the first case all times are selected for the first location, so the result is a time series. In the second case all locations are selected for the first time, so the result is a `SpatialPointsDataFrame`.

The second (time) dimension accepts any valid POSIX date as a character string, so to select years we can specify the year range using the `::` range operator; since we specify only years, the POSIX specification selects all dates in the named years.

```
rr <- rural[, "2005::2010"]
class(rr)

[1] "STFDF"
attr(,"package")
[1] "spacetime"

length(rural@data$PM10)

[1] 306810

length(rr@data$PM10)

[1] 127820
```

The number of space-time observations has been reduced to about 2/5 of the original series.

Some of the stations have no observations at all during these years; there is no need to include them in the further analysis.

TASK 9 : Remove stations that only have missing values in this period. •

The code for this somewhat complicated.

To find missing values in a station's record, we use the `is.na` function, which returns the logical value `TRUE` if a value is missing and `FALSE` otherwise, so if applied to a vector it returns a vector of `TRUE` and `FALSE`.

To determine if all the station's records are missing, we use the `all` function on the logical vector, which returns `TRUE` only if all of its arguments are `TRUE`.

The vector which we want to examine is the time-series portion of the object, of class `xts`; this can be extracted with the `as` object coercion method, which returns a matrix of observations: rows are times and columns are locations.

The `apply` method is used to apply a function on either rows or columns of a matrix or dataframe. We use it on the columns to process each location's time series; the construction `all(is.na)()` on this vector will be `TRUE` if and only if there is no data for that station.

Once we have these row indices, the `[]` matrix selection operator can be used to remove them; recall the first dimension of the `STFDF` object is

the spatial identifier. The minus sign - means to select all rows *except* the named ones, here the vector of stations with only missing values. We see the effect of the selection with the `dim` "dimensions" method, which has a specialization for the STFDF class.

```
dim(rr)

    space      time variables
       70      1826         1

(na.stations <- which(apply(as(rr, "xts"), 2, function(x) all(is.na(x)))))

DEBE062 DEUB007 DEUB003 DEUB002 DEMV004 DEUB034 DEHE048 DEUB032
      5       8      11      14      20      22      26      28
DEMV012 DEHE034 DESL008 DEUB041 DEUB017 DEMV001 DEBB051 DESN052
     29      35      36      44      45      53      59      69
DEUB042
     70

r5to10 <- rr[-na.stations,]
rm(na.stations)
dim(r5to10)

    space      time variables
       53      1826         1
```

Q5 : *How many stations were removed?* *Jump to A5* •

TASK 10 : Summarize this dataset. •

We use the `summary` method, which has a method for class STFDF:

```
summary(r5to10)

Object of class STFDF
 with Dimensions (s, t, attr): (53, 1826, 1)
[[Spatial:]]
Object of class SpatialPoints
Coordinates:
               min      max
coords.x1  6.28107 14.78617
coords.x2 47.80847 54.92497
Is projected: FALSE
proj4string :
[+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0]
Number of points: 53
[[Temporal:]]
     Index              timeIndex
 Min.   :2005-01-01   Min.   :2558
 1st Qu.:2006-04-02   1st Qu.:3014
 Median :2007-07-02   Median :3470
 Mean   :2007-07-02   Mean   :3470
 3rd Qu.:2008-09-30   3rd Qu.:3927
 Max.   :2009-12-31   Max.   :4383
[[Data attributes:]]
      PM10
 Min.   :  0.560
 1st Qu.:  9.275
 Median : 13.852
 Mean   : 16.261
 3rd Qu.: 20.333
 Max.   :269.079
 NA's   :21979
```

Q6 : *What are the three dimensions of the object? What information does each have?* *Jump to A6* •

TASK 11 : Examine the station codes. •

The station codes are stored as attributes of the `coords` slot of the `sp` slot; these are extracted with the `attributes` function:

```
str(attributes(r5to10@sp@coords))

List of 2
 $ dim     : int [1:2] 53 2
 $ dimnames:List of 2
  ..$ : chr [1:53] "DESH001" "DENI063" "DEUB038" "DEBE056" ...
  ..$ : chr [1:2] "coords.x1" "coords.x2"

(station.ids <- attributes(r5to10@sp@coords)$dimnames[[1]])

 [1] "DESH001" "DENI063" "DEUB038" "DEBE056" "DEBE032" "DEHE046"
 [7] "DENW081" "DESH008" "DESN049" "DESN076" "DETH026" "DENI059"
[13] "DEUB039" "DEHE028" "DEMV017" "DEBB053" "DENW063" "DETH061"
[19] "DERP014" "DEUB035" "DEUB031" "DEUB033" "DEBY047" "DENW065"
[25] "DEUB030" "DEBW103" "DENI058" "DEBB056" "DERP017" "DETH042"
[31] "DEBB075" "DESN051" "DEHE043" "DEUB004" "DEUB029" "DEUB040"
[37] "DESN074" "DEBW031" "DEBW087" "DENW064" "DENW068" "DENI019"
[43] "DEUB026" "DEUB005" "DEHE051" "DEBW030" "DENI060" "DERP015"
[49] "DEUB001" "DERP016" "DERP013" "DENI051" "DEUB028"
```

> **Note:** The station names can also be extracted by the `row.names` function applied to the `sp` slot: `row.names(r5to10@sp)`.

The codes all begin with `DE`, the ISO 3166-1 2-letter code for "Deutschland" (Germany). For simpler displays we remove the `DE` from all codes, using the `substr` function to extract the 3rd through 7th characters of each string:

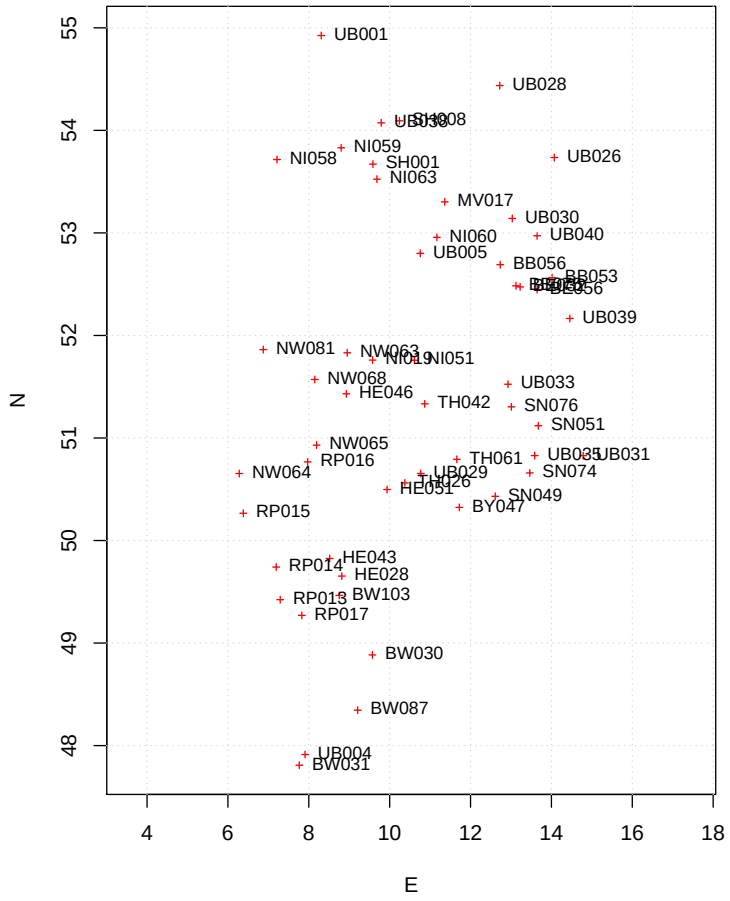
```
(station.ids <- substr(station.ids, start=3, stop=7))

 [1] "SH001" "NI063" "UB038" "BE056" "BE032" "HE046" "NW081" "SH008"
 [9] "SN049" "SN076" "TH026" "NI059" "UB039" "HE028" "MV017" "BB053"
[17] "NW063" "TH061" "RP014" "UB035" "UB031" "UB033" "BY047" "NW065"
[25] "UB030" "BW103" "NI058" "BB056" "RP017" "TH042" "BB075" "SN051"
[33] "HE043" "UB004" "UB029" "UB040" "SN074" "BW031" "BW087" "NW064"
[41] "NW068" "NI019" "UB026" "UB005" "HE051" "BW030" "NI060" "RP015"
[49] "UB001" "RP016" "RP013" "NI051" "UB028"
```

TASK 12 : Plot the locations of each station along with its label. •

The coördinates are in the `sp` "spatial" slot of the `STFDF` object; we extract them with the `coordinates` method. Finally, the `$` operator is used to select the proper attribute, and then the `[[]]` list extraction operator is used to select the first list, which has the first attribute's names.

> **Note:** To display the map in approximately correct aspect, we compute the aspect ratio from the cosine of the median N coördinate, using the `cos` function. Degrees of longitude shrink away from the equator.

```
(aspect.ratio <- cos(median(coordinates(r5to10@sp)[,2])))

[1] 0.3943696

plot(coordinates(r5to10@sp), pch=3, col="red",
     cex=0.5, asp=1/aspect.ratio,
     xlab="E", ylab="N")
grid()
text(coordinates(r5to10@sp), labels=station.ids, pos=4, cex=0.8)
```



TASK 13 : Interpret the station codes. •

After removing `DE`, the first and second letters indicate a region, which we can see with the `substr` function to extract the 1st and 2nd characters of each string, and then find the unique codes with the `unique` function:

```
unique(substr(station.ids, start=1, stop=2))

 [1] "SH" "NI" "UB" "BE" "HE" "NW" "SN" "TH" "MV" "BB" "RP" "BY" "BW"
```

Most of these codes appear to be ISO 3166-2:DE codes for German Länder (states): `SH` is Schleswig-Holstein, `NW` is Nordrhein-Westfalen (North Rhine - Westphalia, common abbreviation NRW), etc.; looking at the geo-

graphic distribution on the map confirms this. It appears that there are no stations for this period in Bavaria (Bayern, BY) or Saxony-Anhalt (ST). Code UB is spread around the country; this code likely stands for "übrig" (other), perhaps for stations not run by the states.

We check this by overlaying the boundary of Germany.

TASK 14 : Make a bounding polygon of Germany and display it, along with each station location and its two-letter state codes. •

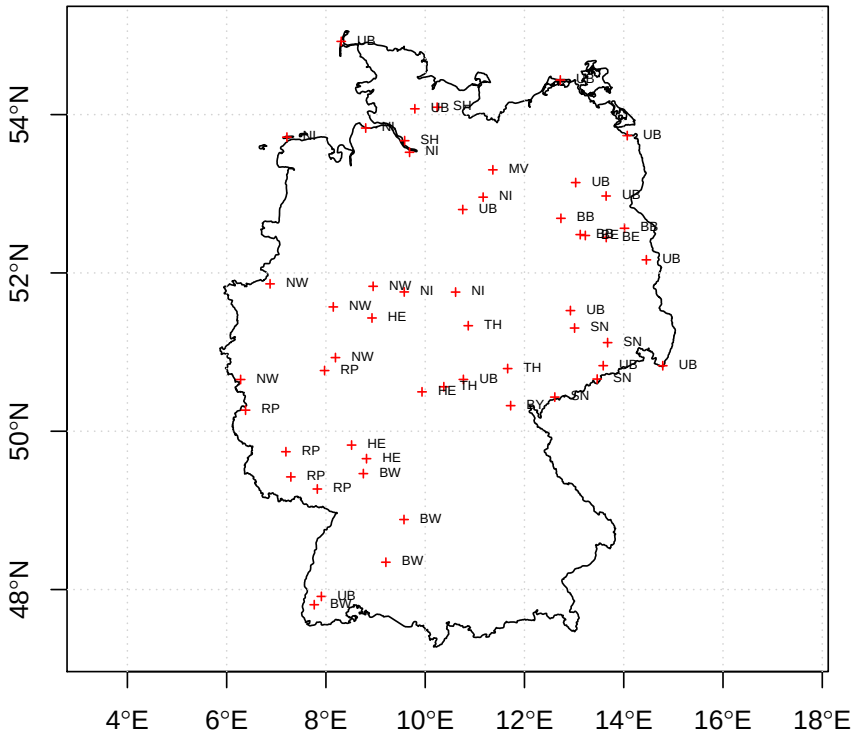
Country boundaries are provided by the `worldHires` dataset provided with the `mapdata` package, which was created from what its authors call a "cleaned-up" version of the CIA World Data Bank II data of 2003[8]. We extract the boundary with the `map` function, and then converted them to a SpatialPolygons object with the `map2SpatialPolygons` function of the `maptools` package. The `fill` argument to the `map` function converts the boundary coördinates into a polygon by joining the last and first points.

The boundary is first displayed with `plot`, the points are added with `points`, and text is added with `text`. The two-letter codes are the 3rd and 4th in the station identification, and are extracted with the `substr` "substring" function.

> **Note:** There is no need to supply an aspect ratio, since the boundary is a spatial object with a defined coördinate reference system.

```
library(mapdata)
tmp <- map('worldHires', 'Germany', fill=TRUE, plot=FALSE)
library(maptools)
de.boundary <-
    map2SpatialPolygons(tmp, IDs=tmp$names,
                        proj4string=CRS(proj4string(rr@sp)))
summary(de.boundary)

Object of class SpatialPolygons
Coordinates:
        min      max
x  5.864166 15.04003
y 47.274750 55.05666
Is projected: FALSE
proj4string :
[+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0]

plot(de.boundary, axes=T)
points(coordinates(r5to10@sp), pch=3, cex=0.5, col="red")
grid()
text(coordinates(r5to10@sp), labels=substr(station.ids,1,2), pos=4, cex=0.5)
```

[8] http://www.evl.uic.edu/pape/data/WDB/



The “übrig” (other) stations may be those not controlled by one of the states.

We can also display these in Google Earth by writing the station locations as a KML “place-mark” file, using the `writeOGR` function. However, we first need to convert just the spatial information in this `STFDF` object to a `SpatialPointsDataFrame`, with a data slot, here the station name.

```
library(rgdal)
r5to10sp <- SpatialPointsDataFrame(r5to10@sp,data=data.frame(id=station.ids))
str(r5to10sp)

Formal class 'SpatialPointsDataFrame' [package "sp"] with 5 slots
  ..@ data       :'data.frame': 53 obs. of  1 variable:
  .. ..$ id: Factor w/ 53 levels "BB053","BB056",..: 32 21 51 5 4 13 26 33 34 37 ...
  ..@ coords.nrs : num(0)
  ..@ coords     : num [1:53, 1:2] 9.59 9.69 9.79 13.65 13.23 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:53] "DESH001" "DENI063" "DEUB038" "DEBE056" ...
  .. .. ..$ : chr [1:2] "coords.x1" "coords.x2"
  ..@ bbox       : num [1:2, 1:2] 6.28 47.81 14.79 54.92
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:2] "coords.x1" "coords.x2"
  .. .. ..$ : chr [1:2] "min" "max"
  ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
  .. .. ..@ projargs: chr "+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0"
```

The coördinate reference system (slot `proj4string`) is already in the

system used by Google Earth, so there is no need for coördinate or datum transformation.

```
writeOGR(r5to10sp, "PM10points.kml", "id", driver="KML", overwrite_layer=TRUE)
```

Figure 1a shows the place-marks of the stations in North Rhine - Westphalia state; Figure 1b shows the location of station DENW063 near Soest (NRW); note the precision of the georeference.

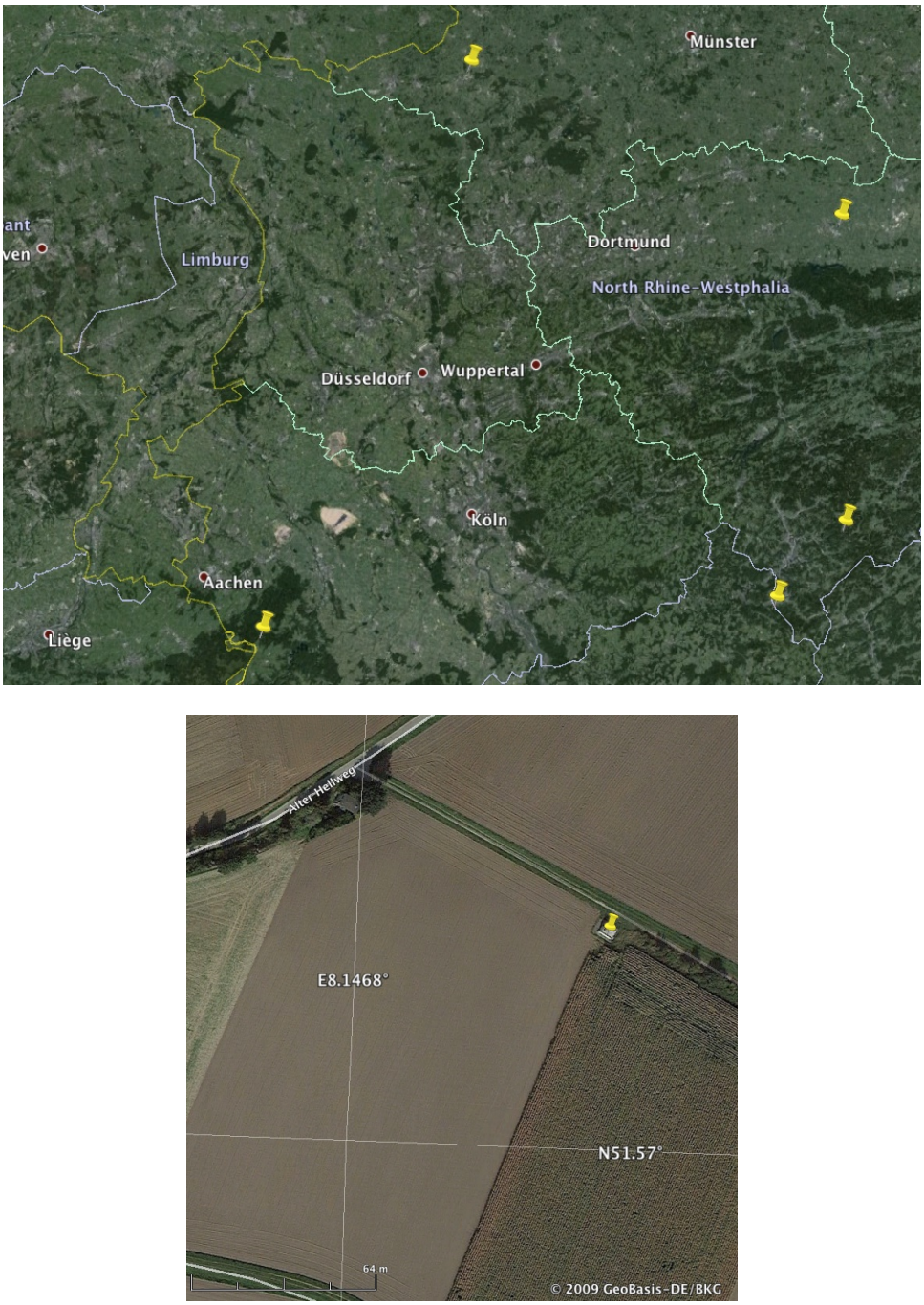


Figure 1: (a) PM10 monitoring stations in North Rhine-Westphalia; (b) closeup of station DENW063

3 Investigating local temporal structure

A time-series analysis is possible at each station.

TASK 15 : Extract the stations in Nordrhein-Westfalen. •

We identify the states in NRW with the == logical operator on the substring of the 1st and 2nd characters of the string, extracted with the `substr` function. We then extract just the station number, since we know they are all in NRW, and use the `attributes` function to store these in the `dimnames` field of the object's coördinates; this simplifies later displays.

```
(ix <- (substr(station.ids, start=1, stop=2)=="NW"))

 [1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
[12] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
[23] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[34] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE
[45] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE

dim(r5to10)

    space      time variables
       53      1826         1

r5to10nrw <- r5to10[ix,]
dim(r5to10nrw)

    space      time variables
        5      1826         1

(station.ids.nrw <- substr(station.ids[ix], 3, 5))

[1] "081" "063" "065" "064" "068"

attributes(r5to10nrw@sp@coords)$dimnames[[1]] <- station.ids.nrw
```

There are only five stations in NRW.

This is the first station in the list of five; it is located on the northeast edge of Borken in the northwestern part of the State, near the border with the Netherlands[9].

TASK 16 : Display the temporal autocorrelation of station `DENW081`. •

The temporal autocorrelation coefficient shows how much an observation before or after a given time is correlated, over the whole time series. The term "auto" means that the same variable is being correlated with itself, in this case by considering the same variable at different observation times. If the time series is 2nd order stationary (i.e., has same expected value and variance over the whole series), at lag k the autocorrelation is:

$$\rho_k = \frac{E[(z_t - \mu)(z_{t+k} - \mu)]}{\sigma_z^2} \quad (1)$$

[9] To see this, enter the coördinates into Google Earth or another mapping program.

which can be estimated as r_k:

$$r_k = c_k / c_0 \tag{2}$$

$$c_k = \frac{1}{N} \sum_{t=1}^{N-k} (z_t - \bar{z})(z_{t+k} - \bar{z}), k = 0, 1, 2 \ldots K \tag{3}$$

$$\text{var}[r_k] \approx \frac{1}{N} \left(1 + 2 \sum_{v=1}^{q} r_q^2 \right), k > q \tag{4}$$

The `acf` "time series autocorrelation" function computes and displays the autocorrelation. However, it can not process missing values, so we first remove them with the `na.omit` function on the time series vector.

> **Note:** A complication is that when only one record is selected from an STFDF object, it is converted to to an `xts` "extensible time-series" object, with a field `timeIndex` which shows the time sequence. So to do a univariate autocorrelation we need to explicitly convert this to a time series with the `as.ts` "as a time series" function, and limit it to just the target variable PM10.

```
class(tmp <- na.omit(r5to10nrw[1,]))
```

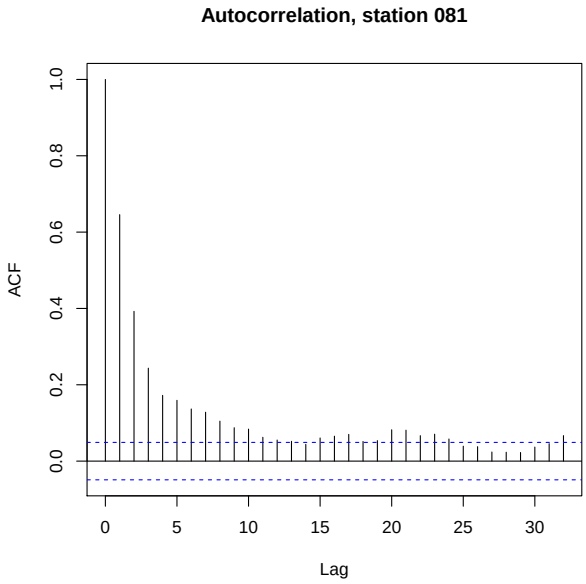
```
[1] "xts" "zoo"
```

```
str(tmp)
```

```
An 'xts' object on 2005-01-01/2009-12-31 containing:
  Data: num [1:1599, 1:2] 32 14.7 22.1 17 22.8 ...
 - attr(*, "dimnames")=List of 2
  ..$ : NULL
  ..$ : chr [1:2] "PM10" "timeIndex"
  Indexed by objects of class: [Date] TZ: UTC
  xts Attributes:
List of 1
 $ na.action: 'omit' int [1:227] 163 237 238 278 494 495 528 569 570 571 ...
  ..- attr(*, "index")= num [1:227] 1.12e+09 1.12e+09 1.13e+09 1.13e+09 1.15e+09 ...
```

```
r5to10nrw.1.ts <- as.ts(na.omit(r5to10nrw[1,]))[,"PM10"]
acf(r5to10nrw.1.ts,
    main=paste("Autocorrelation, station", station.ids.nrw[1]))
acf(r5to10nrw.1.ts, plot=FALSE)
```

```
Autocorrelations of series 'r5to10nrw.1.ts', by lag

    0     1     2     3     4     5     6     7     8     9    10
1.000 0.646 0.392 0.244 0.172 0.159 0.137 0.128 0.105 0.087 0.084
   11    12    13    14    15    16    17    18    19    20    21
0.062 0.055 0.052 0.044 0.061 0.065 0.070 0.052 0.054 0.082 0.081
   22    23    24    25    26    27    28    29    30    31    32
0.067 0.071 0.058 0.039 0.038 0.024 0.023 0.023 0.037 0.046 0.067
```



Q7 : *Describe the temporal autocorrelation of PM10 at this station.*
Jump to A7 •

Another way to look at autocorrelation is with the **partial** autocorrelation function. This measures the autocorrelation at each lag *after* accounting for previous lags. For example, if all autocorrelation can be explained at lag 1, then there is no partial autocorrelation at lags 2, 3, ...; that is, the apparent autocorrelation at these lags can be explained by repeated lag-1 correlations.

This assumes the process is **autoregressive** (AR) and can be modelled as:

$$\tilde{z}_t = \phi_1 \tilde{z}_{t-1} + \phi_2 \tilde{z}_{t-2} + \cdots + \phi_p \tilde{z}_{t-p} + a_t \tag{5}$$

where ϕ_i are the autocorrelation parameters (strength of dependence at each lag), and a_t is the white noise, sometimes called "shock" at time t. The simplest AR process is AR(1): $\tilde{z}_t = \phi_1 \tilde{z}_{t-1} + a_t$, i.e., the value at time t is only dependent on the immediately-preceding value at time $(t-1)$, and some random shock a_t.

To compute the partial autocorrelation coefficient ρ_j , define $\phi_{k,j}$ as the coefficient j of an autoregressive process of order k, then:

$$\rho_j = \phi_{k,1}\rho_{j-1} + \cdots + \phi_{k,k}\rho_{j-k},\ j = 1, 2, \ldots k \tag{6}$$

To compute the ρ solve the following linear system:

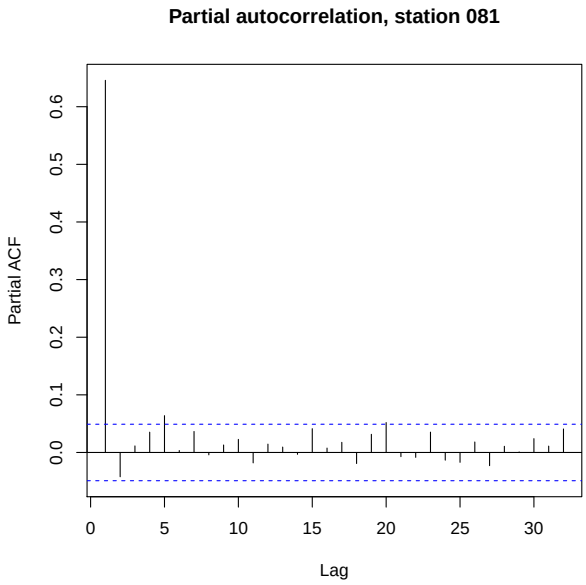
$$\mathbf{P}_k \boldsymbol{\phi}_k = \boldsymbol{\rho}_k \tag{7}$$

$$\mathbf{P}_k = \begin{bmatrix} 1 & \rho_1 & \rho_2 & \cdots & \rho_{k-1} \\ \rho_1 & 1 & \rho_1 & \cdots & \rho_{k-2} \\ . & . & . & \cdots & . \\ \rho_{k-1} & \rho_{k-2} & \rho_{k-3} & \cdots & 1 \end{bmatrix} \tag{8}$$

TASK 17 : Display the partial temporal autocorrelation of station DENW081. •

The `pacf` "time series partial autocorrelation" function computes and displays the partial autocorrelation.

```
pacf(r5to10nrw.1.ts,
     main=paste("Partial autocorrelation, station", station.ids.nrw[1]))
pacf(r5to10nrw.1.ts, plot=FALSE)


Partial autocorrelations of series 'r5to10nrw.1.ts', by lag

     1      2      3      4      5      6      7      8      9     10
 0.646 -0.042  0.011  0.035  0.064  0.003  0.036 -0.004  0.013  0.023
    11     12     13     14     15     16     17     18     19     20
-0.018  0.015  0.009 -0.003  0.041  0.008  0.018 -0.019  0.032  0.052
    21     22     23     24     25     26     27     28     29     30
-0.007 -0.008  0.035 -0.013 -0.017  0.018 -0.023  0.011  0.001  0.024
    31     32
 0.011  0.041
```



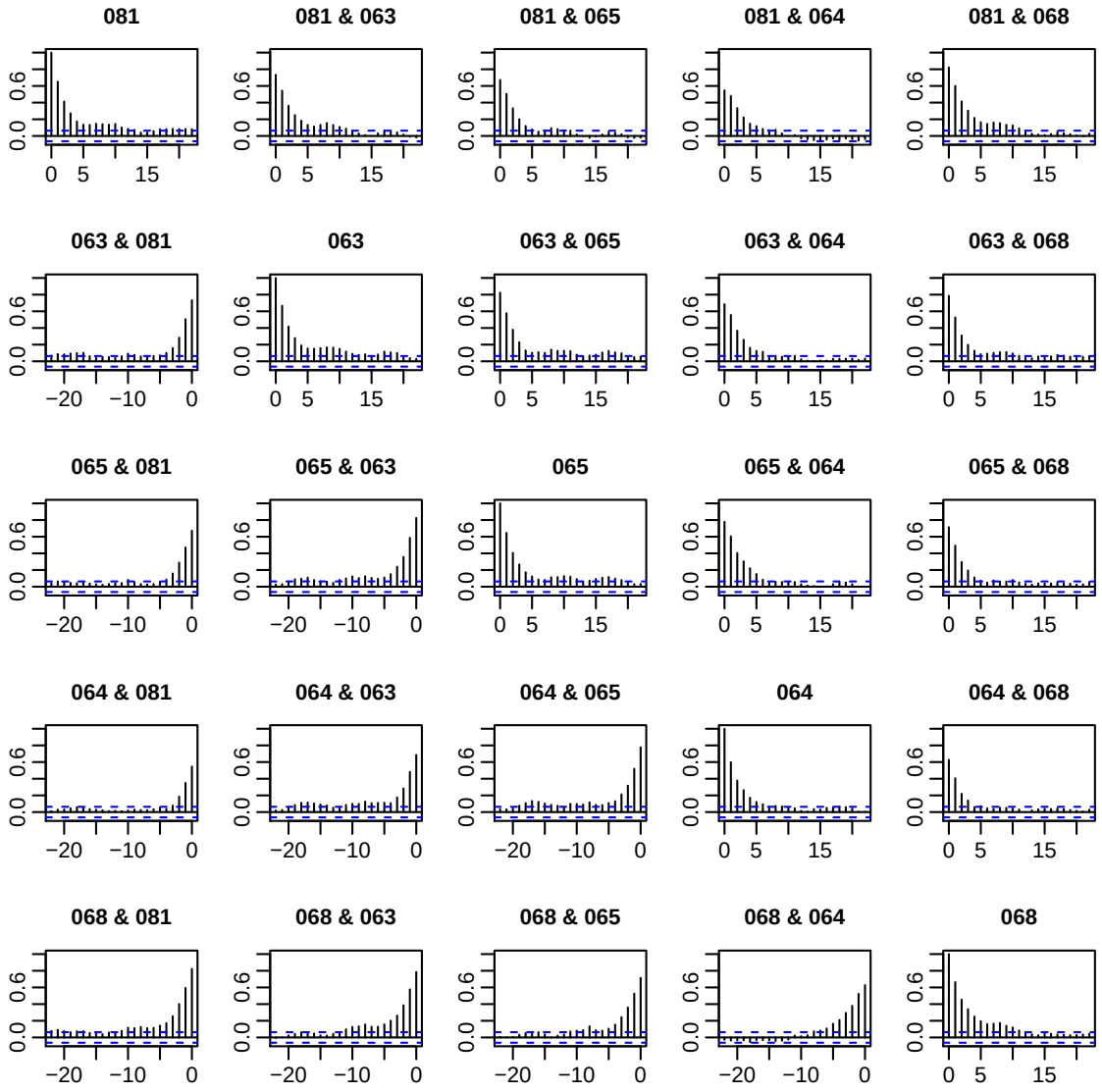
Q8 : *Describe the partial temporal autocorrelation of PM10 at this station. What can you infer about time behaviour of the process controlling PM10 concentration?* *Jump to A8* •

In the above steps we only considered one station. We can also compute temporal **cross-correlations** between stations; this shows how related they are, and may suggest common causes. This does not require any knowledge of the stations' coördinates, although we do have these.

TASK 18 : Compute and plot the direct and cross-correlations between the stations in NRW. •

Cross-correlation may be computed with the `ccf` function; but a simpler way is to compute both auto- and cross correlations by passing a multivariate time series object to the `acf` method. Class `xts` is the "extensible time series" class, used by the `spacetime` package to store time series. Since there is one time series per station, this is a multivariate series. The `as` method coerces the object given in its first argument, here of class `STFDF`, to the class named in its second argument, here `xts`.

```
acf(na.omit(as(r5to10nrw, "xts")), xlab="", ylab="")
```



The lag-0 cross-correlations do not have to equal 1 (perfect correlation), and rarely do.

Q9 : *Which stations appear to be most and least correlated in time?*
Jump to A9 •

Q10 : *Is the temporal cross-correlation symmetric (same forward and backward?)*
Jump to A10 •

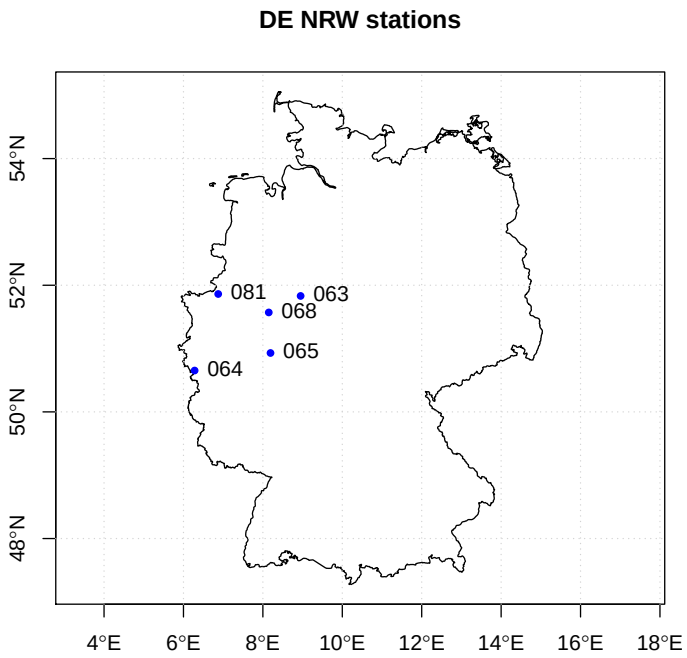
This shows that cross-correlations can be asymmetric. If we denote the correlation between station A at time t by $Z(s_A, t)$ and station B at time $(t + h)$, i.e., after a lag of h units, which may be positive or negative, by $Z(s_B, t + h)$ as $\rho_{AB}(h)$, then:

$$\rho_{AB}(h) = \rho_{BA}(-h) \neq \rho_{AB}(-h) = \rho_{BA}(h) \tag{9}$$

We can try to explain the difference in cross-correlation by the spatial distance – is it so that closer stations are more cross-correlated?

TASK 19 : Display the locations of the NRW stations. •

```
plot(de.boundary, axes=T, main="DE NRW stations")
points(coordinates(r5to10nrw@sp), pch=20, col="blue")
grid()
text(coordinates(r5to10nrw@sp), labels=station.ids.nrw, pos=4)
```



TASK 20 : Compute the distances between the NRW stations. •

The `spDists` function computes distances between locations given by `sp` class coördinates:

```
print(spDists(r5to10nrw@sp, longlat=TRUE), digits=3)

      [,1]  [,2]  [,3] [,4]  [,5]
[1,]   0.0 143.1 138.4  141  93.8
[2,] 143.1   0.0 113.3  228  62.6
[3,] 138.4 113.3   0.0  138  71.3
[4,] 140.7 227.8 138.2    0 165.9
[5,]  93.8  62.6  71.3  166   0.0

row.names(r5to10nrw@sp)

[1] "081" "063" "065" "064" "068"
```

Note that even though the coördinates are given in longitude and latitude, the distances are given in metric coördinates, namely km, because of the optional `TRUE` value of the `longlat` argument. The `spDists` function examines the `proj4string` slot of the `sp` slot of the `STFDF` object, and then computes distances on the ellipsoid.

Q11 : *Which stations are closest? furthest? Does this agree with the cross-correlation plots?* *Jump to A11* •

We thus have evidence of **spatio-temporal dependence**: the **temporal** correlation may be affected by **spatial** dependence.

4 Investigating spatial structure

At any one observation interval (here, day) we can investigate the spatial structure of the measured variable, with the standard tools of geostatistics.

Task 21 : Extract the PM10 observations for all the stations on the day with maximum pollution anywhere in Germany during the five-year period. •

First the index in the dataset of the maximum PM10, and its value:

```
(ix <- which.max(r5to10$PM10))

[1] 81688

(pm.max <- r5to10$PM10[ix])

[1] 269.079
```

To determine the date of this observation, we first find the station, and then search in the station record for the date. Each station has an equal-length time-series, so using the integer division operator %% of the index on the dimension of the data vector gives the station number.

The `index` function of the `zoo` "irregular time series" package (loaded by the `xts` "extensible time series" package, which in turn is loaded by

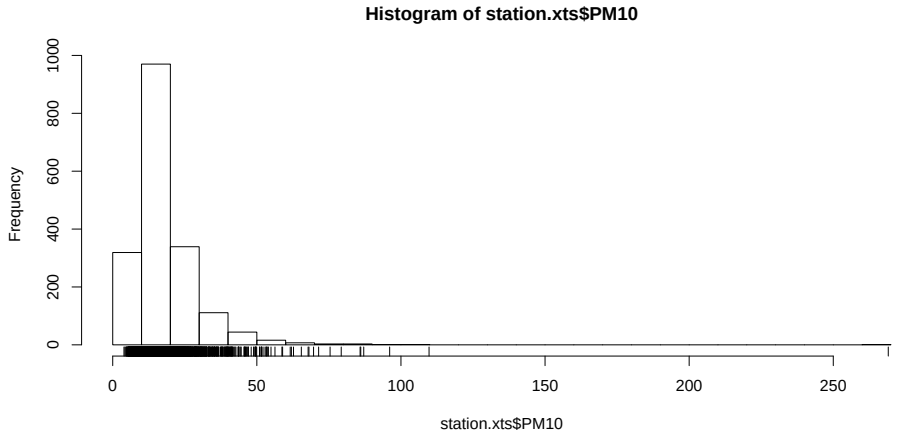
`spacetime` package) extracts the date(s) from a time series, here the single date.

```
dim(r5to10)

    space      time variables
       53      1826         1

(station.id <- ix%%dim(r5to10)[1])

space
   15

(station.name <- attributes(r5to10@sp@coords)$dimnames[[1]][station.id])

[1] "DEMV017"

coordinates(r5to10)[station.id,]

coords.x1 coords.x2
 11.36297  53.30235

station.xts <- r5to10[station.id,]
str(station.xts)

An 'xts' object on 2005-01-01/2009-12-31 containing:
  Data: num [1:1826, 1:2] 32.62 13 8.17 15.75 17.25 ...
 - attr(*, "dimnames")=List of 2
  ..$ : NULL
  ..$ : chr [1:2] "PM10" "timeIndex"
  Indexed by objects of class: [Date] TZ: UTC
  xts Attributes:
 NULL

summary(station.xts)

     Index                PM10           timeIndex
 Min.   :2005-01-01   Min.   :  3.898   Min.   :2558
 1st Qu.:2006-04-02   1st Qu.: 11.268   1st Qu.:3014
 Median :2007-07-02   Median : 15.250   Median :3470
 Mean   :2007-07-02   Mean   : 18.090   Mean   :3470
 3rd Qu.:2008-09-30   3rd Qu.: 21.312   3rd Qu.:3927
 Max.   :2009-12-31   Max.   :269.079   Max.   :4383
                      NA's   :11
```

The summary shows that the maximum PM10 value is very high; is this consistent with other observations?

To check, we make a histogram with the `hist` function, with a `rug` plot of actual observations, of this time series of PM10 values:

```
hist(station.xts$PM10, breaks=20)
rug(station.xts$PM10)
```



Q12 : *Describe the feature-space distribution of PM10 at this station over the five years. How unusual is the reported maximum?* *Jump to A12* •

Extract the record of the maximum; this becomes a one-item extended time series:

```
(date.xts <- station.xts[which.max(station.xts$PM10)])

               PM10 timeIndex
2009-03-22 269.079      4099

class(date.xts)

[1] "xts" "zoo"
```

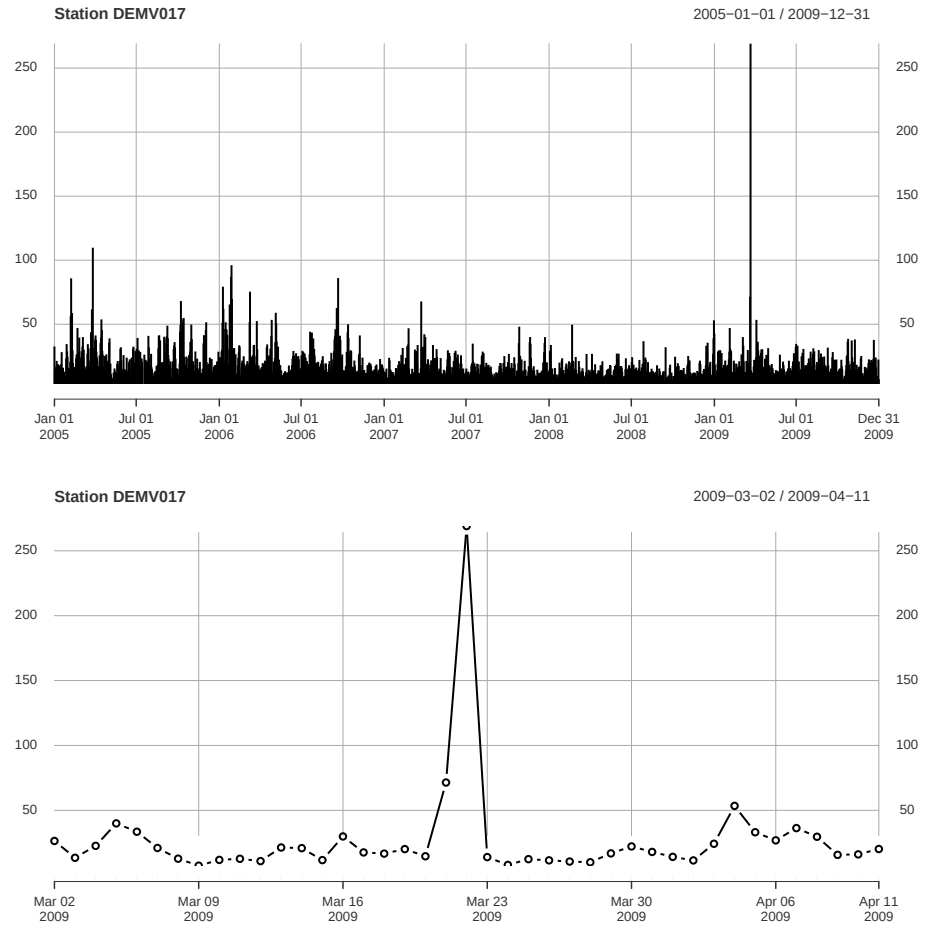
Knowing the record, find its index into the `xts` data structure; this is of class `Date`, a basic R class.

```
(date.ix <- index(date.xts))

[1] "2009-03-22"

class(date.ix)

[1] "Date"
```

Q13 : *What was the maximum PM10 in the five-year time series? At which station and on which date was it observed?* *Jump to A13* •

We can also visualize the time series for that station, to see the context of the maximum:

```
plot(station.xts$PM10, type='h', main="Station DEMV017",
     ylab="PM10 concentration")
plot(station.xts[seq(date.ix-20, date.ix+20, by=1)]$PM10,
     type="b", main="Station DEMV017",
     ylab="PM10 concentration")
```



Q14 : *In context, does this value seem likely?* *Jump to A14* •

We will see if this value is spatially-consistent with its neighbours.

TASK 22 : Select the records for all stations for the chosen date. •

The `[]` matrix selection operator can be used; recall that the second matrix coördinate of an `ST` object is time. However, the index is of class `Date`, whereas the time index into the `STFDF` object `r5to10` must be a subscript that can be interpreted as an index in an `xts` object. The simplest way to do this is convert the date to a character string with the `as` method, and then use this as the subscript.

```
class(date.ix)

[1] "Date"

date.ix <- as.POSIXct(date.ix)
class(date.ix)

[1] "POSIXct" "POSIXt"

date.ix <- as(date.ix,"character")
class(date.ix)
```

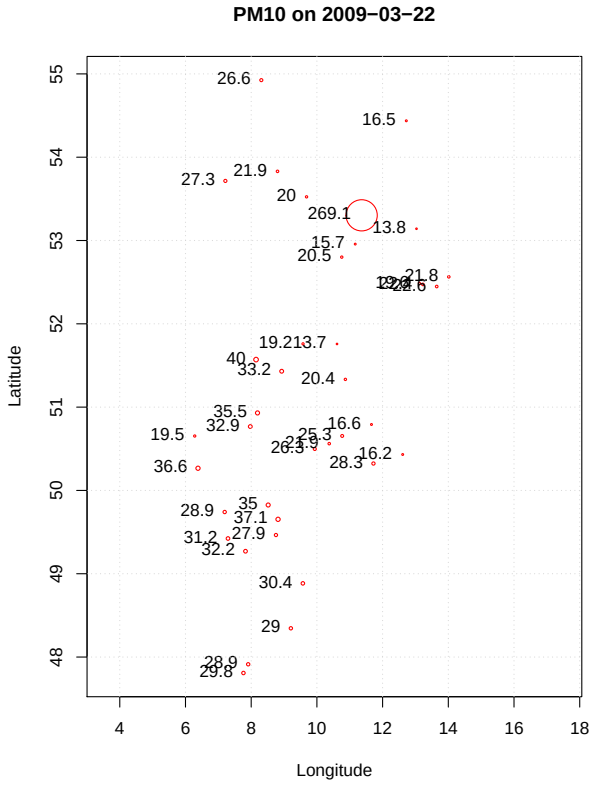
```
[1] "character"

dim(r5to10)

    space      time variables
       53      1826         1

r.date <- r5to10[,date.ix]
dim(r.date)

[1] 53  1

summary(r.date)

Object of class SpatialPointsDataFrame
Coordinates:
              min      max
coords.x1  6.28107 14.78617
coords.x2 47.80847 54.92497
Is projected: FALSE
proj4string :
[+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0]
Number of points: 53
Data attributes:
      PM10
 Min.   : 13.71
 1st Qu.: 20.07
 Median : 26.45
 Mean   : 31.94
 3rd Qu.: 31.03
 Max.   :269.08
 NA's   :15
```

Since there is only one date, this is now a `SpatialPointsDataFrame`, which we can analyze as usual with `gstat`.

TASK 23 : Make a post-plot of the PM10 concentrations on this date, also with the actual values. •

The `plot` method on the coördinates, extracted with the `coordinates` function, plots the locations; the `text` function adds the labels.

```
plot(coordinates(r.date),
     cex=4*r.date$PM10/max(na.omit(r.date$PM10)),
     col="red", asp=1/aspect.ratio, xlab="Longitude", ylab="Latitude",
     main=paste("PM10 on",date.ix))
text(coordinates(r.date), labels=round(r.date$PM10,1), pos=2)
grid()
```



Q15 : *How consistent is the very high measurement at station DEMV017 with its neighbours?* *Jump to A15* •

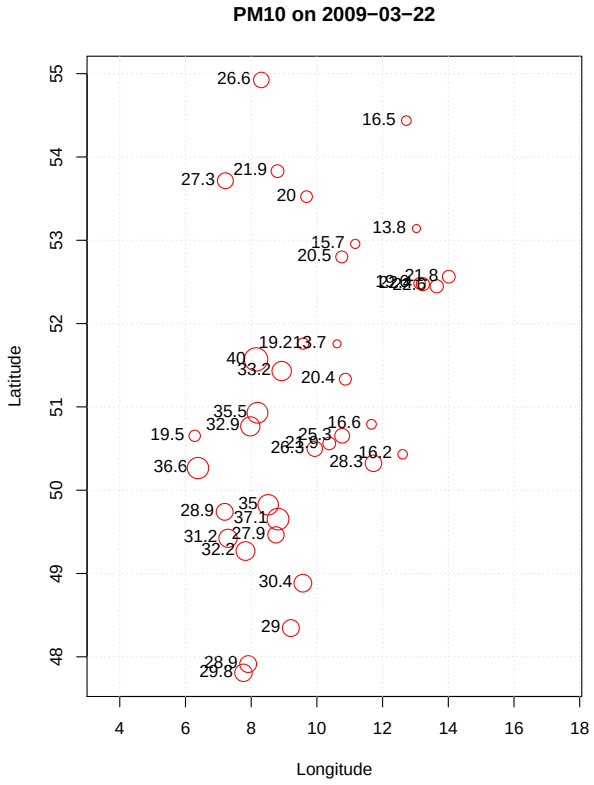
We have no definite grounds to reject this value; however we can see that it is not at all consistent with surrounding values, and would seriously distort variogram analysis. One way to deal with this is to remove it from this day's spatial dataset, while noting the anomaly and explaining it from an unusual cause (or recording error – this requires further investigation).

TASK 24 : Remove this measurement from the single day dataset, and then display a post-plot of the reduced dataset. •

```
summary(r.date@data)

      PM10
 Min.   : 13.71
 1st Qu.: 20.07
 Median : 26.45
 Mean   : 31.94
 3rd Qu.: 31.03
 Max.   :269.08
 NA's   :15

r.date <- r.date[-station.id,]
summary(r.date@data)

      PM10
```

```
 Min.   :13.71
 1st Qu.:19.97
 Median :26.29
 Mean   :25.53
 3rd Qu.:30.39
 Max.   :40.05
 NA's   :15
```

```
plot(coordinates(r.date),
     cex=3*r.date$PM10/max(na.omit(r.date$PM10)),
     col="red", asp=1/aspect.ratio, xlab="Longitude", ylab="Latitude",
     main=paste("PM10 on",date.ix))
text(coordinates(r.date), labels=round(r.date$PM10,1), pos=2)
grid()
```



Q16 : *Does there appear to be spatial correlation of PM10 on the selected date?* *Jump to A16*

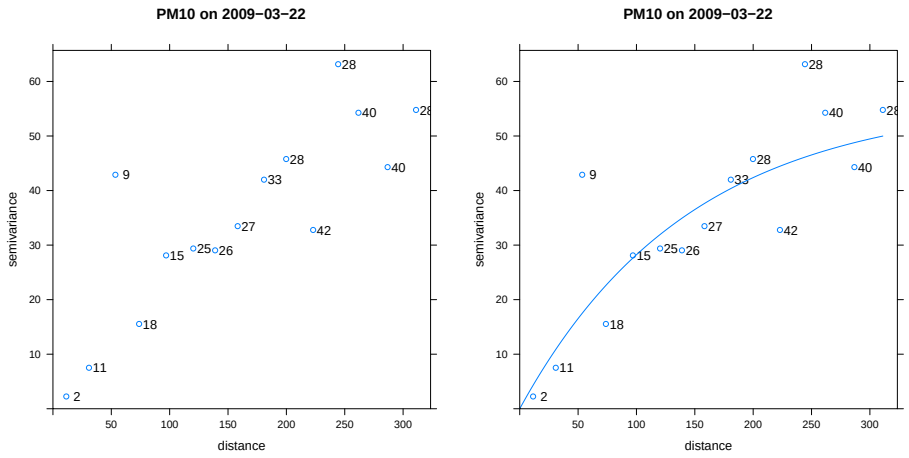
•

TASK 25 : Compute, display, and model the omnidirectional variogram of PM10 on this date. •

> **Note:** Although there may be anisotropy because of, e.g., wind direction, the number of observations is too small to analyze it.

We use the `variogram` method of the `gstat` package. This can not deal with missing values, so we first remove them.

```
str(r.date)

Formal class 'SpatialPointsDataFrame' [package "sp"] with 5 slots
  ..@ data       :'data.frame': 52 obs. of  1 variable:
  .. ..$ PM10: num [1:52] NA 20 NA 22.6 22.4 ...
  ..@ coords.nrs : num(0)
  ..@ coords     : num [1:52, 1:2] 9.59 9.69 9.79 13.65 13.23 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:52] "DESH001" "DENI063" "DEUB038" "DEBE056" ...
  .. .. ..$ : chr [1:2] "coords.x1" "coords.x2"
  ..@ bbox       : num [1:2, 1:2] 6.28 47.81 14.79 54.92
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:2] "coords.x1" "coords.x2"
  .. .. ..$ : chr [1:2] "min" "max"
  ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
  .. .. ..@ projargs: chr "+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0"

sum(is.na(r.date$PM10))

[1] 15

sum(!is.na(r.date$PM10))

[1] 37

vo <- variogram(PM10 ~ 1, r.date[!is.na(r.date$PM10),])
plot(vo, plot.numbers=T, main=paste("PM10 on",date.ix))
(vom <- fit.variogram(vo, model=vgm(50, "Exp", 300/3, 0)))

  model    psill    range
1   Nug  0.00000   0.0000
2   Exp 56.57785 144.6796

plot(vo, plot.numbers=T, model=vom, main=paste("PM10 on",date.ix))
```



Q17 : *Describe the variogram. Does it provide evidence of spatial correlation of PM10 on the selected date?* *Jump to A17*

•

The model is fairly good despite the small number of points. We can interpolate over the study area at this one date; this is a step in a simple space-time analysis:

1. Compute a variogram on each date;

2. Interpolate (e.g., by kriging) on that date over a grid covering the study area;
3. View this time-series of graphs in sequence, e.g., by animation.

This does not take any advantage of the extra information that might be provided by temporally-correlated measurements. Also, it relies on variograms computed from a small set of points, which are expected to fluctuate day-to-day. Another approach is to compute a lumped (average) variogram model; see below §7.

5 Constructing a grid for prediction

Once we have a variogram model, we can use it to "predict", although since the observations are in the past, this is more properly called "interpolation" to unobserved locations. Interpolation is often over a regular grid, to produce a map.

TASK 26 : Create a space-time grid covering Germany with a resolution of 0.25° x 0.25° of longitude and latitude. •

We do this by first making a square grid covering the bounding box for Germany. The `seq` function makes regular sequences of numbers, here coördinates; the `rep` function repeats them as needed by the `SpatialPoints` constructor. We then restrict the grid to Germany by overlaying the bounding polygon imported and formatted in §2, using the `over` method.

```
bbox(de.boundary)
```

```
        min      max
x  5.864166 15.04003
y 47.274750 55.05666
```

```
x1 <- seq(from=5.75,to=15.25,by=0.25)
x2 <- seq(from=47.25,to=55.25,by=0.25)
de.bbox.grid <- SpatialPoints(cbind(rep(x1,length(x2)),
                                    rep(x2,each=length(x1))),
                              proj4string=CRS(proj4string(r5to10@sp)))
gridded(de.bbox.grid) <- TRUE
summary(de.bbox.grid)
```

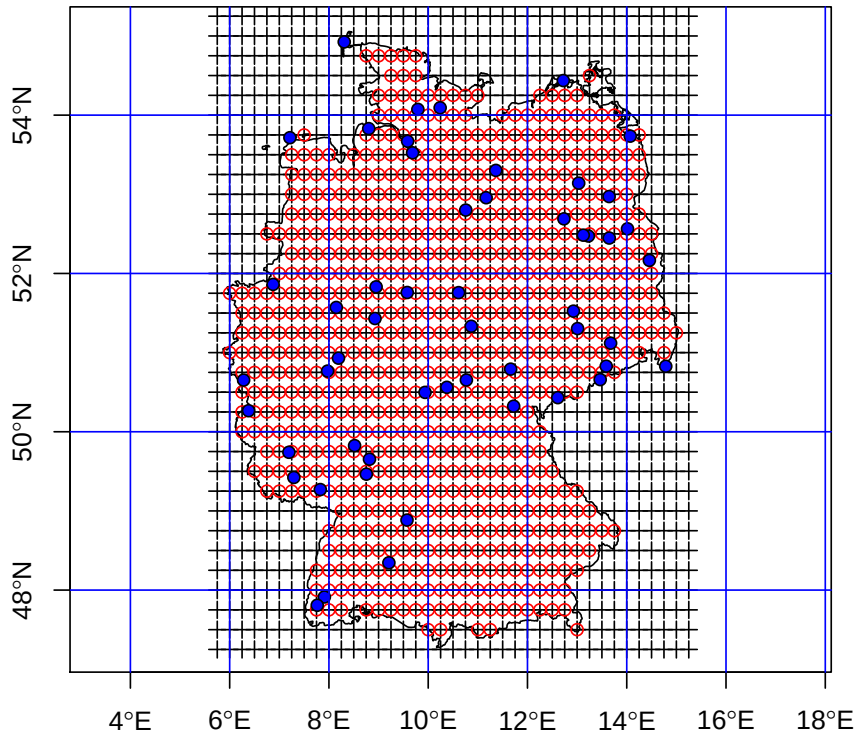
```
Object of class SpatialPixels
Coordinates:
             min    max
coords.x1  5.625 15.375
coords.x2 47.125 55.375
Is projected: FALSE
proj4string :
[+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0]
Number of points: 1287
Grid attributes:
          cellcentre.offset cellsize cells.dim
coords.x1              5.75     0.25        39
coords.x2             47.25     0.25        33
```

```
de.grid <- de.bbox.grid[!is.na(over(de.bbox.grid, de.boundary)),]
summary(de.grid)
```

```
Object of class SpatialPixels
Coordinates:
```

```
             min    max
coords.x1  5.625 15.375
coords.x2 47.125 55.375
Is projected: FALSE
proj4string :
[+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0]
Number of points: 729
Grid attributes:
          cellcentre.offset cellsize cells.dim
coords.x1              5.75     0.25        39
coords.x2             47.25     0.25        33
```

```
plot(de.boundary, axes=T)
points(coordinates(de.bbox.grid), pch=3)
points(coordinates(de.grid), pch=1, col="red")
points(coordinates(r5to10@sp), pch=21, bg="blue")
grid(lty=1, col="blue")
```



Both grids are of spatial class `SpatialPixels`. Because of the grid resolution, some areas of Germany, including the observation stations at Sylt in the extreme N and Zittau in the extreme SE, were missed by the overlay of the bounding polygon.

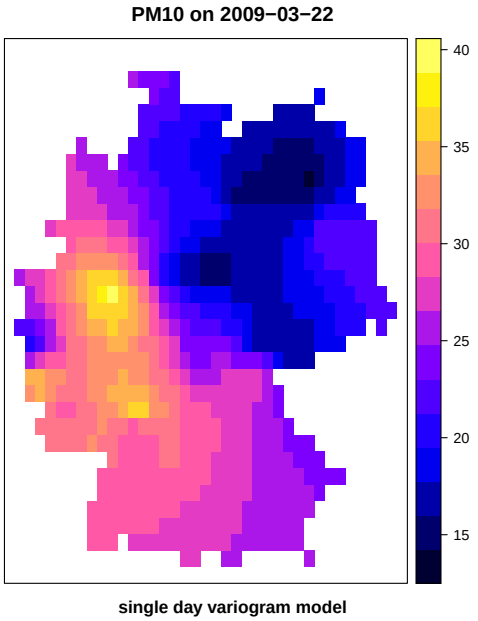
6 Spatial prediction

We can now map PM10 concentrations over Germany on any day for which we have station measurements and for which we have computed a variogram model.

TASK 27 : Predict PM10 concentration over the 0.25° x 0.25° geographic grid, on the selected day, by Ordinary Kriging. Plot the predictions. •

> **Note:** Since the prediction is over blocks, it might be preferable to use block kriging to predict an average PM10 concentration. Using punctual kriging gives single prediction at the block centre only; the single value there is presumed to represent any point location in the block. However, the `krige` function has not implemented block kriging for geographic coördinates. It would be necessary to transform the datasets into metric coördinates.

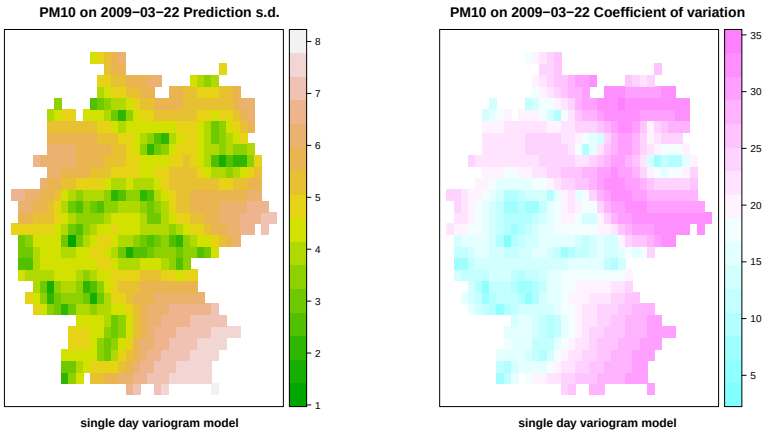
```
proj4string(r.date) <- CRS(proj4string(r.date))
k.one <- krige(PM10 ~ 1, loc=r.date[!is.na(r.date$PM10),],
               newdata=de.grid, model=vom)

[using ordinary kriging]

spplot(k.one, zcol="var1.pred", col.regions=bpy.colors(64),
       main=paste("PM10 on",date.ix),
       sub="single day variogram model")
```



The uncertainty of prediction is given by the kriging prediction variance. The standard deviation is on the same scale as the prediction and thus easier to interpret. Even easier is the coefficient of variation.

TASK 28 : Plot the kriging prediction standard deviation and coefficient of variability. •

```
k.one$var1.sd <- sqrt(k.one$var1.var)
k.one$var1.cv <- 100*k.one$var1.sd/k.one$var1.pred
spplot(k.one, zcol="var1.sd", col.regions=terrain.colors(64),
       main=paste("PM10 on",date.ix, "Prediction s.d."),
       sub="single day variogram model")
spplot(k.one, zcol="var1.cv", col.regions=cm.colors(64),
       main=paste("PM10 on",date.ix, "Coefficient of variation"),
       sub="single day variogram model")
```



Q18 : *Where are the highest kriging prediction variances and the highest coefficients of variation? Why?* *Jump to A18* •

7 Investigating temporally-averaged spatial structure

One way to obtain a more reliable variogram model is to compute a so-called **lumped-time** model; that is, spatial structure averaged over time. This assumes that the spatial structure is the same over time, although the data values change. In this case we can compute a **lumped** or **averaged** spatial variogram, using many dates as replications. This will give a large number of point-pairs and allow reliable variogram modelling. The resulting variogram model could be then applied at each date to produce that date's map; this avoids the need to compute variogram models on each date from small point sets.

TASK 29 : Sample 100 dates at random as a `SpatialPointsDataFrame`, with an extra attribute (besides the PM10) to record the date. •

> **Note:** We do not use the entire dataset because it would be very large; if our assumption is correct any reasonably large subset should give the same result.

The `lapply` function applies a function over a list; here we build a list of the date indices with the `sample` function applied to the time (2nd) dimension of the spacetime object.

The function to apply is a here written by us, using the `function` function[10]. This **user-defined function** first extracts the station information at the given date, then adds a new variable to the data frame, and finally returns the newly-composed object. This list is then converted to a single dataframe with the `rbind` function applied to the list with the `do.call` function.

> **Note:** The `set.seed` function initialises the random number generator; this is only necessary if you want your results to be the same as these. In principle you should get similar results with any (or no) random seed.

```
set.seed(6345789)
sort(sample.index <- sample(dim(r5to10)[2], 100))
```

```
  [1]   36   41   53   54   69  100  144  167  177  193  195  206  210
 [14]  230  240  280  281  306  362  379  384  389  427  431  457  495
 [27]  497  514  516  518  539  583  617  667  670  675  678  687  691
 [40]  731  757  781  836  839  862  905  906  918  946  960  969  975
 [53]  986 1011 1030 1035 1046 1062 1083 1094 1109 1111 1126 1129 1138
 [66] 1158 1216 1281 1282 1290 1302 1320 1324 1330 1398 1416 1442 1443
 [79] 1448 1461 1497 1509 1518 1540 1554 1576 1582 1586 1596 1646 1664
 [92] 1686 1696 1711 1723 1731 1743 1754 1773 1818
```

```
spdf.lst <- lapply(sample.index,
                   function(i) {
                       x = r5to10[,i]
                       x$ti = i
                       return(x)}
                   )
spdf <- do.call(rbind, spdf.lst)
summary(spdf)
```

```
Object of class SpatialPointsDataFrame
Coordinates:
               min      max
coords.x1  6.28107 14.78617
coords.x2 47.80847 54.92497
Is projected: FALSE
proj4string :
[+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0]
Number of points: 5300
Data attributes:
      PM10              ti
 Min.   :  0.727   Min.   :  36.0
 1st Qu.:  9.601   1st Qu.: 485.5
 Median : 13.950   Median : 964.5
 Mean   : 16.696   Mean   : 925.2
 3rd Qu.: 20.583   3rd Qu.:1402.5
 Max.   :109.094   Max.   :1818.0
 NA's   :1193
```

> **Note:** The function can be more compactly written as:
> `function(i) {x = r5to10[,i]; x$ti = i; x}`

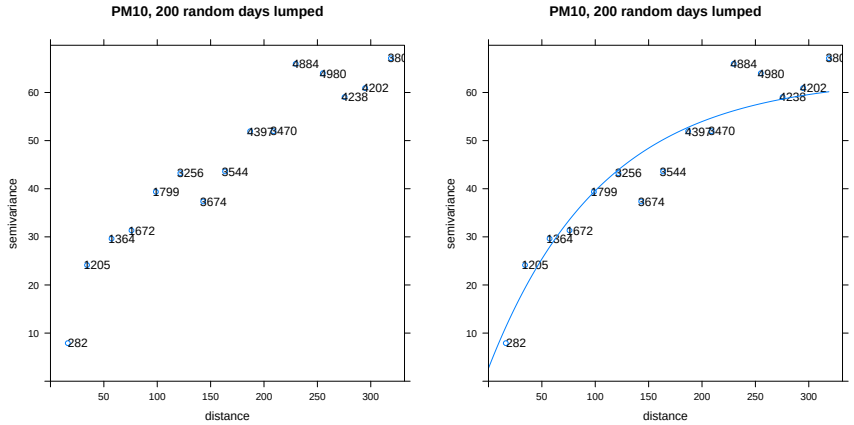
TASK 30 : Compute and model the lumped variogram. •

We again use the `variogram` method of the `gstat` package, but with some differences to make a pooled variogram. First, we specify a regressor, here the time index (field `ti`), i.e., the variogram is residual after accounting for the time of each observation. This is necessary because the

[10] Confused? Read again!

different dates will generally have different overall levels of PM10, even though we assume the same spatial structure. So the formula `PM10 ~ ti` removes the mean PM10 for at each time, before computing the semi-variances. Second, the `dX` optional argument specifies the maximum distance of the regressor for which point-pairs will be included in the variogram. Here we specify `dX=0`, meaning that only observations on the same date (the regressor) will be included in the calculation.

```
vl <- variogram(PM10 ~ ti, spdf[!is.na(spdf$PM10),], dX=0)
plot(vl, plot.numbers=T, main=paste("PM10, 200 random days lumped"))
(vlm <- fit.variogram(vl, model=vgm(60, "Exp", 250/3, 0)))

  model     psill    range
1   Nug  2.690848   0.0000
2   Exp 60.463423 106.1837

plot(vl, plot.numbers=T, model=vlm, main="PM10, 200 random days lumped")
```



Q19 : *Describe the variogram. Does it provide evidence of spatial correlation of PM10 on the selected date? How does the fitted model compare to the model for the first date?* *Jump to A19* •

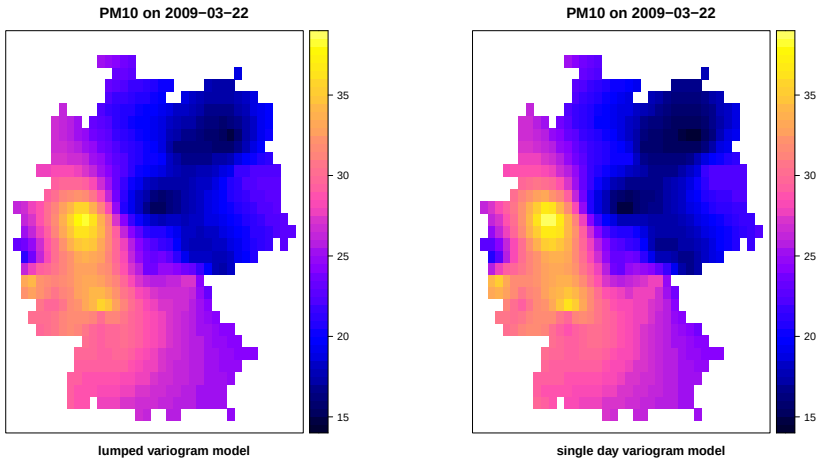
8 Spatial prediction with a temporally-averaged spatial structure

---

TASK 31 : Predict PM10 concentration over the 0.25° x 0.25° geographic grid, on the selected day by Ordinary Kriging, but now with the lumped variogram. Plot the kriging predictions next to the prediction from the single-date variogram model, on the same scale. •

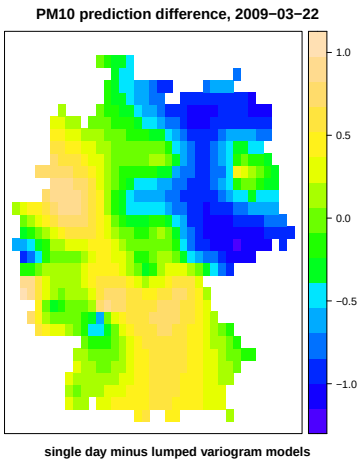
```
k.one.lumped <- krige(PM10 ~ 1, loc=r.date[!is.na(r.date$PM10),],
                      newdata=de.grid, model=vlm)

[using ordinary kriging]

legend.breaks <- seq(floor(min(k.one$var1.pred, k.one.lumped$var1.pred)),
                     ceiling(max(k.one$var1.pred, k.one.lumped$var1.pred)),
                     by=0.5)
spplot(k.one.lumped, zcol="var1.pred", col.regions=bpy.colors(64),
       main=paste("PM10 on",date.ix),
       sub="lumped variogram model",
       at=legend.breaks)
```

```
spplot(k.one, zcol="var1.pred", col.regions=bpy.colors(64),
       main=paste("PM10 on",date.ix),
       sub="single day variogram model",
       at=legend.breaks)
```



TASK 32 : Plot the differences between the two predictions. •

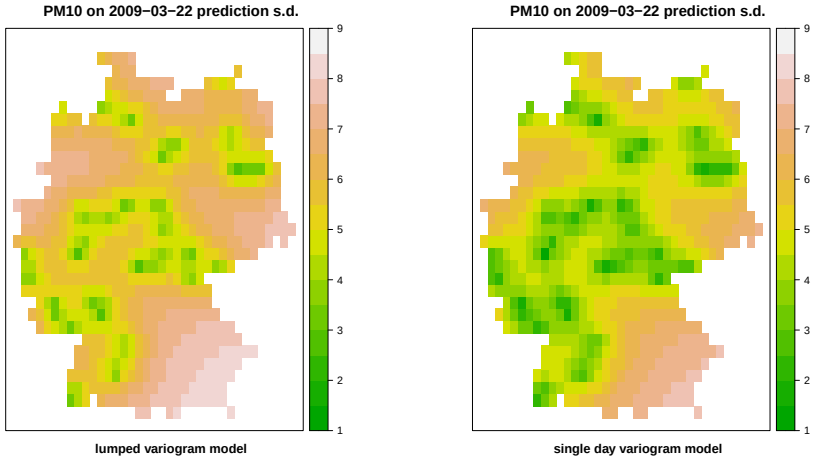
```
summary(k.one$diff <-
        k.one$var1.pred - k.one.lumped$var1.pred)

    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-1.15195 -0.53228  0.02291 -0.06277  0.40996  0.97843

spplot(k.one, zcol="diff", col.regions=topo.colors(64),
       main=paste("PM10 prediction difference,",date.ix),
       sub="single day minus lumped variogram models")
```



Q20 : *How similar are the predictions made with the single-date and lumped variogram models?* *Jump to A20* •

TASK 33 : Plot the kriging prediction standard deviations from the averaged and single-day variogram model predictions, side-by-side on the same scale. •

```
k.one.lumped$var1.sd <- sqrt(k.one.lumped$var1.var)
legend.breaks <- seq(floor(min(k.one$var1.sd, k.one.lumped$var1.sd)),
                     ceiling(max(k.one$var1.sd, k.one.lumped$var1.sd)),
                     by=0.5)
spplot(k.one.lumped, zcol="var1.sd", col.regions=terrain.colors(64),
       main=paste("PM10 on",date.ix,"prediction s.d."),
       sub="lumped variogram model",
       at=legend.breaks)
spplot(k.one, zcol="var1.sd", col.regions=terrain.colors(64),
       main=paste("PM10 on",date.ix,"prediction s.d."),
       sub="single day variogram model",
       at=legend.breaks)
```



Q21 : *How similar are the prediction standard deviations from the single-date and lumped variogram model kriging predictions?* *Jump to A21* •

This is as far as we want to go with spatial modelling and interpolation. The really interesting part comes now, when we combine space and time in the analysis.

9 Investigating spatio-temporal structure

Now we consider both space and time together. One way to do this is to look at the spatial variograms at different time lags – that is, how correlated are observations in space, but at different times?

The hypothesis here is that the spatial structure becomes weaker as the time differences increase.

TASK 34 : Compute a spatio-temporal variogram of the stations over time. •

The spatio-temporal variogram summarizes the semivariance at all combinations of space and time separations. A single semivariance is estimated as:

$$\gamma(\mathbf{h}, u) = \frac{1}{2}\sum(z_{\mathbf{s},t} - z_{\mathbf{s}+\mathbf{h},t+u})^2 \tag{10}$$

These are summarized in spatio-temporal bins, to have enough point-pairs to estimate structure.

If the `variogram` method of the `gstat` package is passed an object of class `STFDF`, it will specialise to the `variogramST` function, which will be used to compute a spatio-temporal variogram. With all stations and times, this can take a while; to see how long we use the `system.time` function, which records the elapsed time necessary to run any R code supplied as its argument.

Here you can see the results on the author's system:

```
system.time(vst <- variogramST(PM10 ~ 1, r5to10))
```

```
      user  system elapsed
717.436  90.161 807.852
```

This took about 13.5 minutes on the author's system. As a compromise, you could compute this on a sequence of times, e.g., over the first 200 days:

```
system.time(vst <- variogramST(PM10 ~ 1, r5to10[,1:200]))
```

```
      user  system elapsed
 56.821   0.813  57.688
```

This took a bit less than one minute on the author's system.

We continue with the full space-time variogram; if you computed with a shorter time sequence your results will look somewhat different.

```
summary(vst)
```

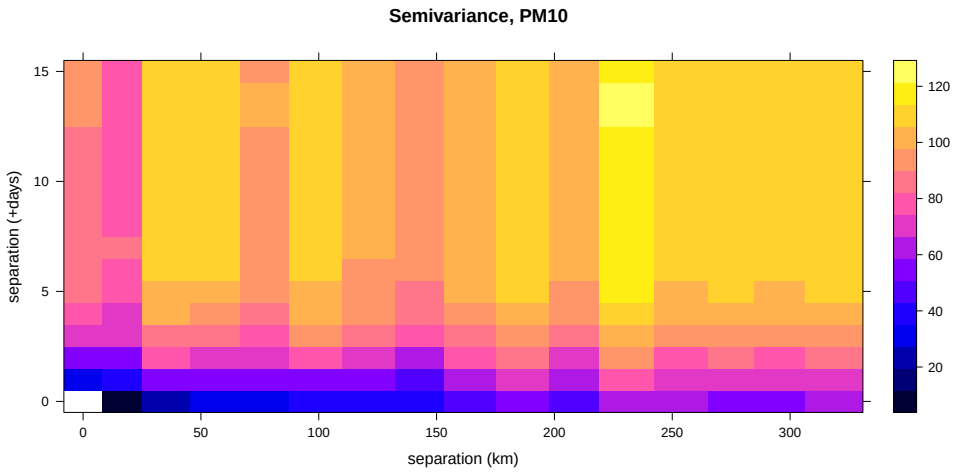
```
       np              dist           gamma
 Min.   :     0   Min.   :  0.00   Min.   : 11.54
 1st Qu.: 60115   1st Qu.: 75.87   1st Qu.: 83.95
 Median :124608   Median :163.46   Median : 99.23
 Mean   :106520   Mean   :155.55   Mean   : 93.55
 3rd Qu.:150068   3rd Qu.:242.30   3rd Qu.:109.21
 Max.   :180704   Max.   :318.74   Max.   :121.51
                  NA's   :1        NA's   :1
     id              timelag         spacelag
 Length:256         Min.   : 0.00   Min.   :  0.00
 Class :character   1st Qu.: 3.75   1st Qu.: 71.27
 Mode  :character   Median : 7.50   Median :153.51
                    Mean   : 7.50   Mean   :154.20
                    3rd Qu.:11.25   3rd Qu.:235.75
                    Max.   :15.00   Max.   :317.99
```

Note that the subset must be a sequence, because of the assumption of equal time lags between observations on the time axis.

TASK 35 : Plot the spatio-temporal variogram. •

Plotting in three dimensions (two space, one time) is challenging; we look at three visualisations. First, a 2.5D plot with time vs. separation, with the semivariances shown as a colour ramp. This uses the `plot` method, but when applied to an object of class `STFDF` it specializes to the `plot.gstatVariogram` method. Its default space-time variogram plot is the 2.5D plot.

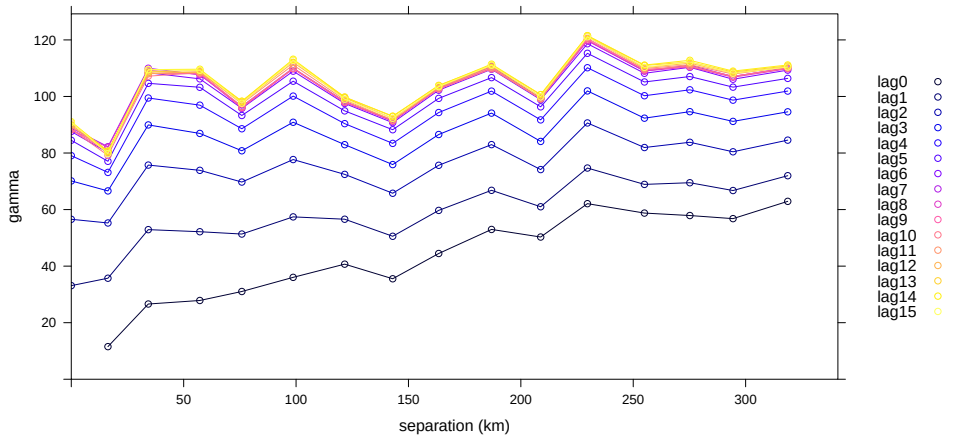
```
plot(vst, xlab="separation (km)", ylab="separation (+days)",
     main="Semivariance, PM10")
```



Note: Since the sequence of average separations is generally not at equal intervals, the underlying `levelplot` graphics function of the `lattice` package gives a warning to this effect. Since the separations are averages but not too different from the centres of the variogram bins, this does not seriously affect the visualization.

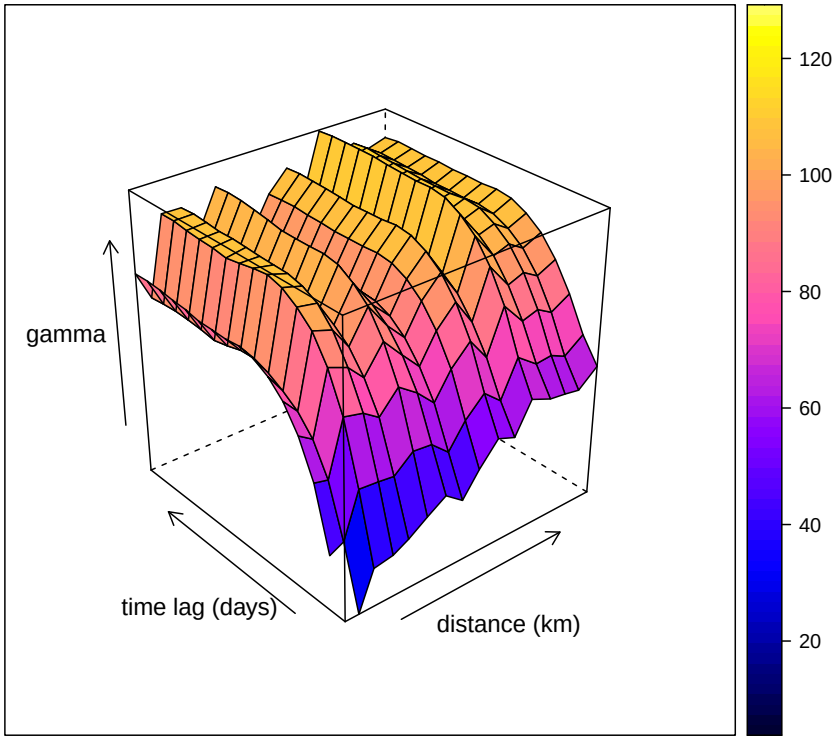
Second, parallel spatial variograms, one per time lag. This also uses the `plot.gstatVariogram` method

```
plot(vst, map = FALSE, xlab="separation (km)",
     ylab = "Semivariance, PM10")
```



Third, a 3D wireframe plot, showing both space and time; this is obtained by specifying the optional `wireframe` argument to `plot.gstatVariogram`:

```
library(lattice)
plot(vst, wireframe=TRUE)
```



Q22 : *What is the approximate semivariance at 150 km spatial and 5 days temporal separation (time lag)?* *Jump to A22* •

To answer this, we extract the relevant row from the space-time variogram, using logical conditions ==, < and >, joined with the& logical operator:

```
str(vst)

Classes 'StVariogram' and 'data.frame': 256 obs. of  6 variables:
 $ np      : num  0 5131 21920 24689 30139 ...
 $ dist    : num  NA 16.3 34.3 57.2 75.9 ...
 $ gamma   : num  NA 11.5 26.6 27.9 31 ...
 $ id      : chr  "lag0" "lag0" "lag0" "lag0" ...
 $ timelag : num  0 0 0 0 0 0 0 0 0 0 ...
  ..- attr(*, "units")= chr "days"
 $ spacelag: num  0 11 32.9 54.8 76.8 ...
  ..- attr(*, "units")= chr "km"
 - attr(*, "boundaries")= num  0 21.9 43.9 65.8 87.7 ...

vst[(vst$timelag==5) & (vst$spacelag > 140) & (vst$spacelag < 160),]

       np    dist    gamma   id timelag spacelag
88 133395 142.999 88.23597 lag5       5 142.5486
```

Q23 : *Describe the spatio-temporal dependence structure.* *Jump to A23* •

10 Modelling spatio-temporal structure

We now attempt to model the spatio-temporal dependence revealed in the variogram. There are several possible models, differing on how they combine the temporal and spatial dependences. Here we consider three, in increasing order of complexity.

1. The simplest is the **metric** model (§10.1): time is considered as another dimension, which must be re-scaled to match the spatial dimension
2. The **separable** model (§10.2) has separate spatial and temporal structures, which are considered to interact only multiplicatively and so can be fit independently but with a common sill.
3. The **sumMetric** model (§10.3) has separate terms for the spatial and temporal components, and an interaction component for the **residuals** not accounted for by these two.

10.1 Metric models

The simplest approach is a **metric** model: time is just another dimension, which must be re-scaled to match the spatial dimension. The time dimension is of course **anisotropic** – the dependence in the time dimension is on a different scale than that in the two space dimensions. The assumed covariance structure is:

$$C(h,u) = C\left(\sqrt{h^2 + (\alpha u)^2}\right) \tag{11}$$

where h is the distance lag, generally 2D, u is the time lag, here another "dimension", and α is a scaling factor ("metric") to match spatial and temporal units.

The covariance structure allows **geometric anisotropy** between space and time, i.e., same structure, nugget and partial sill but the range varies in different dimensions (here, the time vs. space dimensions). For example, $\alpha = 20$ km d^{-1} means that a lag of 20 km in space is equivalent to a lag of 1 day in time.

Space-time variograms are structured as lists, built with the `vgmST` function of `gstat`, with various components depending on the type of variogram. For a metric variogram, specified as `"metric"` in the `stModel` argument, there is a `joint` argument, which specifies a spatial variogram model specified with the `vgm` "variogram model" function of `gstat`; also a `nugget` argument to specify the joint nugget (semivariance at zero time and distance lag), and an anisotropy factor for the time dimension specified with the `stAni` argument.

The critical parameter here is the time anisotropy; we estimate this by looking at the ratio of the two axes when arriving at a similar semivariance. One way to estimate this is to compute the anisotropy ratio

(distance/time) for all the variogram bins within a certain range of semi-variances, summarize it, and use the mean ratio within this range as the initial value.

```
dim(tmp <- vst[(vst$gamma > 70) &
               (vst$gamma < 80) &
               (vst$timelag !=0),])

[1] 15  6

summary(metric.aniso <- tmp$spacelag/tmp$timelag)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.000   1.645  27.413  62.831  71.274 317.993
```

Now build the initial metric variogram model:

```
(vgm.metric <- vgmST(stModel="metric",
                     joint=vgm(50,"Exp",100,0),
                     nugget=10,
                     stAni=mean(metric.aniso)))

joint component:
  model psill range
1   Nug     0     0
2   Exp    50   100
stAni: 62.8310483257133
```

The `fit.StVariogram` function fits the estimate to the empirical variogram. This has an optional `method` argument that specifies how optimization is to be done; this is passed to R's standard `optim` "general purpose optimization" function. Since non-linear optimization is sensitive to starting values and may not converge, to avoid excessive computation we use the optional `control` argument to control the behaviour of `optim`. This argument is a list of control parameters; we specify `maxit` "maximum number of iterations".

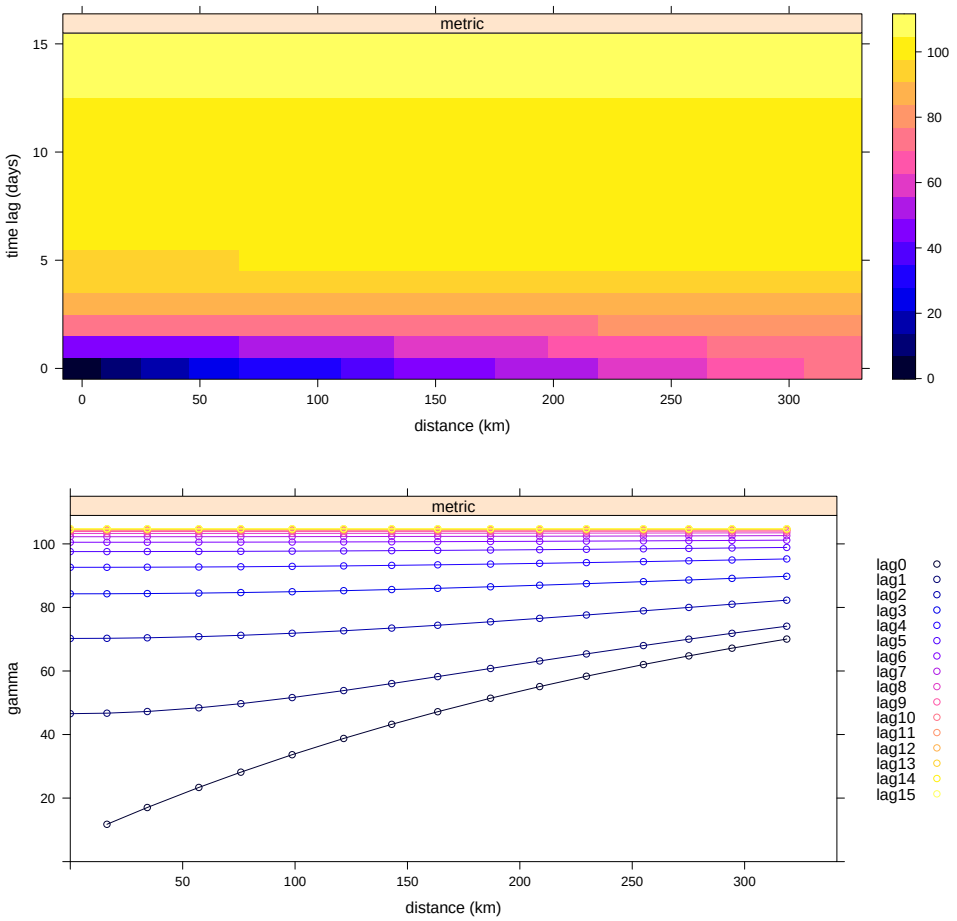
```
vgmf.metric <- fit.StVariogram(vst, vgm.metric,
                               method="L-BFGS-B",
                               control=list(maxit=1024))
print(vgmf.metric)

joint component:
  model     psill    range
1   Nug  6.676082   0.0000
2   Exp 98.126933 307.0716
stAni: 160.214957645994
```

Q24 : *What are the parameters of the fitted model, including the time anisotropy?* *Jump to A24* •

TASK 36 : Plot the fitted model along with the empirical variogram. •

The `plot` method has a specialization `plot.gstatVariogram` for empirical variograms and their fitted models. For space-time variograms these take the form of a 2.5D space-time matrix or a set of parallel variograms, see above.

```
plot(vst, vgmf.metric)
plot(vst, vgmf.metric, map=FALSE)
```



Q25 : *How well does the model fit the empirical variogram? What can you conclude about the appropriateness of the metric model?* *Jump to A25* •

10.2 Separable models

The metric model is rarely realistic; more common is a **separable** model where space and time are fit independently. There is interaction between space and time, but it is simply the product of the two dimensions considered separately. The covariance structure is:

$$C(h,u) = C_s(h) \cdot C_t(u) \tag{12}$$

To fit such a model, we model the spatial and temporal dependences separately; however since the two covariances are multiplied, there can only be one total variogram sill. This maximum semivariance corresponds to the minimum covariance when there is no spatial or temporal dependence. The spatial and temporal sills are then expressed as proportions of the total sill. The "distance" units for the spatial and temporal dependence are the squares of the original units, so there is no need to re-scale time to match space, as was necessary in the metric model.

TASK 37 : Build and fit a separable space-time variogram model. •

Here there is no anisotropy because time is not considered as another "spatial" dimension. The two ranges (space and time) are expressed in their units, here km (space) and days (time). We estimate their values from the empirical variogram. For space, use the lag-0 variogram; recall that the exponential variogram model's range parameter is 1/3 of the effective range (where the variogram reaches 95% of the partial sill). For time, use the lag when there appears to be no more spatial structure in that lagged spatial variogram.

In a separable model there are five parameters:

1. spatial range;
2. proportional spatial nugget;
3. temporal range;
4. proportional temporal nugget;
5. overall sill, specified with the `sill` argument

The partial sill arguments in the space and time variograms are ignored in favour of the specified overall sill; they are kept at a constant (1 - nugget) for space and time; these are then proportional to the overall total sill.

The overall sill can be estimated from the empirical variogram as some proportion of the maximum semivariance in the space-time variogram.

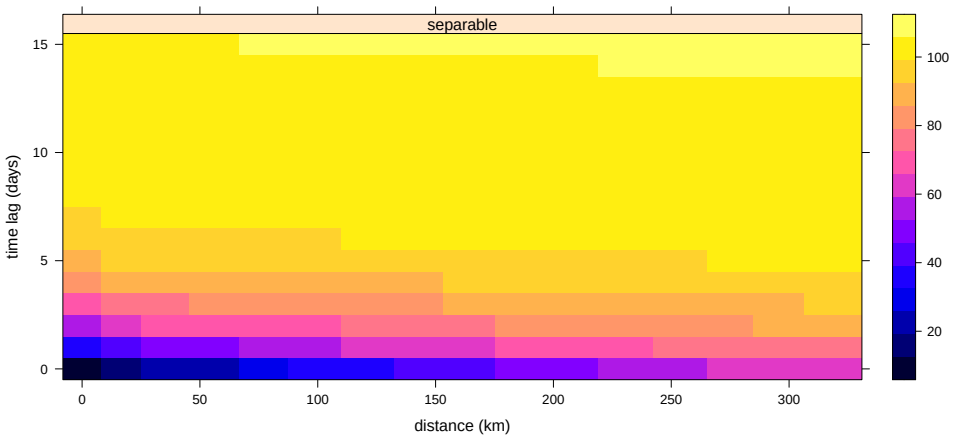
```
(estimated.sill <- quantile(na.omit(vst$gamma), .8))

    80%
109.856

(vgm.sep <- vgmST(stModel="separable",
                  space=vgm(0.9,"Exp", 300,0.1),
                  time=vgm(0.95,"Exp", 2, 0.05),
                  sill=estimated.sill))

space component:
  model psill range
1   Nug   0.1     0
2   Exp   0.9   300
time component:
  model psill range
1   Nug  0.05     0
2   Exp  0.95     2
sill: 109.85600255371
```

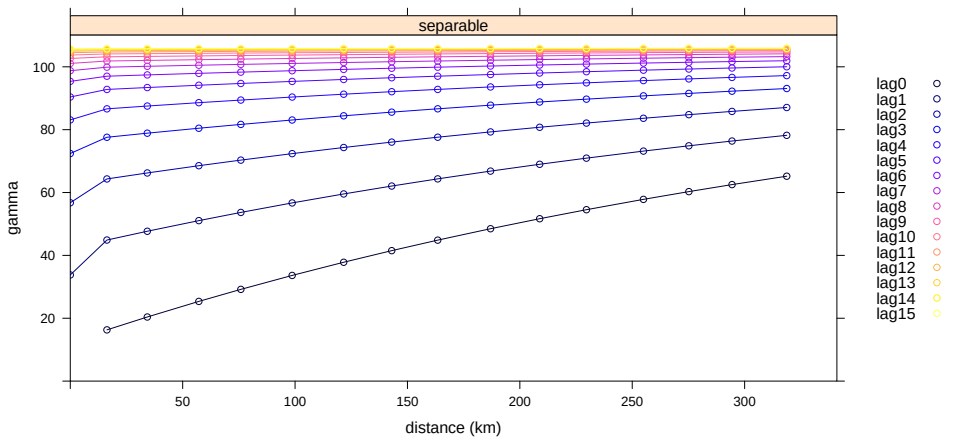
Fitting is again with `fit.StVariogram`. However, we have to ensure that the `optim` "optimize" function called by `fit.StVariogram` does not try to use un-physical values, in particular, negative ranges or semivariances. This can be ensured by passing the optional `lower` argument to `optim`. We use the `c` function to make a vector of five reasonable limiting values. We limit the nuggets to just above zero to avoid convergence problems, and again limit the iterations.

The attributes of the fitted model include the fitted parameters and the value of the optimization function.

```
vgmf.sep <- fit.StVariogram(vst, vgm.sep,
                            method="L-BFGS-B",
                            lower=c(100,0.001,1,0.001,40),
                            control=list(maxit=500))
class(vgmf.sep)

[1] "StVariogramModel" "list"

attr(vgmf.sep, "optim.output")$par

   range.s   nugget.s    range.t   nugget.t       sill
384.257141   0.117017   2.614601   0.001000 106.045063

attr(vgmf.sep, "optim.output")$value

[1] 74.90121
```

```
plot(vst, vgmf.sep)
plot(vst, vgmf.sep, map=FALSE)
```





Q26 : *Compare the variogram fits with the metric and separable models. Which model appears to best match the empirical spatio-temporal variogram?* *Jump to A26* •

10.3 Sum-metric models

The metric model (§10.1) can not represent interaction between space and time; the separable model (§10.2) can only represent the interactions a product of two marginal variograms. A more flexible model form to represent interactions is the sum-metric mode, with the following covariance structure:

$$C(h,u) = C_s(h) + C_t(u) + C_{st}\left(\sqrt{h^2 + (\alpha u)^2}\right) \tag{13}$$

where h is the distance lag, u is the time lag, and α is a scaling factor (hence the "metric") to match spatial and temporal units in the interaction term. The interaction term has the same structure as the metric model, but here only applying to the residuals after accounting for the marginal spatial and temporal correlations. Thus the marginal spatial and temporal semivariances are added, along with the interaction term.

The interaction term allows **geometric anisotropy**, i.e., same structure, nugget and partial sill but the range can vary in different dimensions (here, the time vs. space dimensions), but here (unlike in the metric model) only for the residuals.

This model has been successfully used by Heuvelink and collaborators in several studies [e.g. 9, 10].

As the models get more complicated, so does their parameterization. The sum-metric model requires 10 parameters:

- the **spatial marginal** variogram, 3 parameters: partial sill, nugget, range; this model is specified with the `space` argument;
- the **temporal marginal** variogram, 3 parameters: partial sill, nugget, range; this model is specified with the `time` argument;
- the **space-time residual variogram**, 4 parameters: (1) partial sill, nugget, range of the residuals; this model is specified with the `joint` argument; (2) anisotropy ratio of the residuals, i.e., the ratio between the space and time and ranges; this multiplies the time lag to match the space lag.

The two marginal variograms can be estimated by the variograms at lag 0 in the remaining dimension:

```
(v.sp <- vst[vst$timelag==0,c("spacelag","gamma")])

    spacelag    gamma
1    0.00000       NA
2   10.96528 11.53544
3   32.89584 26.60173
4   54.82639 27.85167
5   76.75695 31.02826
6   98.68751 36.03219
7  120.61807 40.70121
8  142.54863 35.52874
9  164.47918 44.46770
10 186.40974 52.97760
11 208.34030 50.29442
```

```
12 230.27086 62.08886
13 252.20142 58.76482
14 274.13197 57.89326
15 296.06253 56.79585
16 317.99309 62.91278
```

```
(v.t <- vst[vst$spacelag==0,c("timelag","gamma")])
```

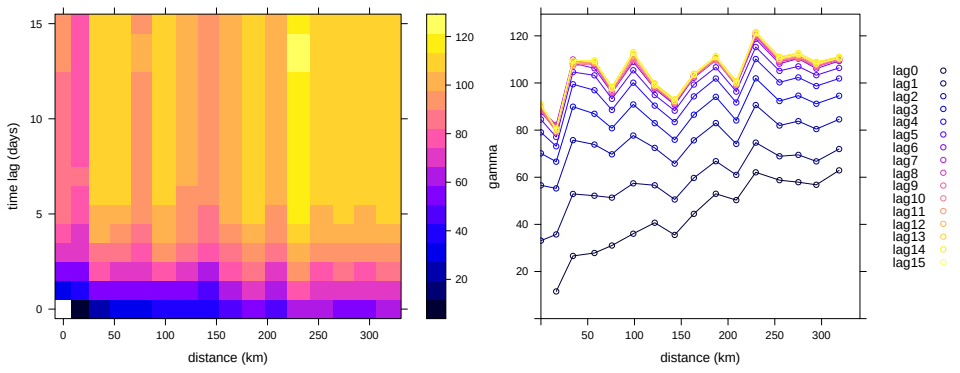
```
    timelag    gamma
1         0       NA
17        1 33.10095
33        2 56.56212
49        3 70.14048
65        4 79.06865
81        5 84.50289
97        6 87.60269
113       7 88.86242
129       8 88.20736
145       9 88.38639
161      10 88.38651
177      11 89.55300
193      12 89.96482
209      13 90.69451
225      14 91.05768
241      15 90.86111
```

Since exponential models are used, the range parameter is specified as 1/3 of the effective range as estimated from the marginal variogram, i.e., where the semivariance reaches 95% of the estimated sill. For example, here by inspection the space-marginal variogram appears to reach a sill at about 240 km and the time-marginal variogram at about 6 days, so the corresponding initial values are 240/3 and 6/3. The anisotropy parameter `stAni` is estimated as the ratio of these two, i.e., the distance equivalent to days to reach the respective effective ranges. Here we estimate as 240/6 km d^{-1}.

> **Note:** The `gstat` package version 1.0-25 and later now has a function `estiStAni` to estimate this parameter; I have not experimented with it.

To estimate the parameters of the `joint` variogram, we look at the space-time empirical variogram, in both the grid and line forms:

```
p1 <- plot(vst)
p2 <- plot(vst, map=F)
plot(p1, split=c(1,1,2,1), more=T)
plot(p2, split=c(2,1,2,1), more=F)
```



We find the range at which there seems to be no more interaction between space and time, that is, where the single-time variograms have the same shape, but with different total sills. This is in distance (space) units; here we estimate about 50 km effective range, which again is divided by 3 because the model is exponential. It is reasonable to specify a zero nugget; we assume the nugget effects are in either space or time but not their interaction. The partial sill is estimated as the semivariance at zero time lag at this estimated range; here this appears to be about 30.

```
vgm.sum.metric <- vgmST(stModel="sumMetric",
                        space=vgm(0.9*max(v.sp$gamma, na.rm=TRUE),
                            "Exp", 250/3,0),
                        time=vgm(0.9*max(v.t$gamma, na.rm=TRUE),
                            "Exp", 6/3, 0),
                        joint=vgm(30,
                            "Exp",50/3, 0),
                        stAni=240/6)
```

Fitting is again with `fit.StVariogram`. We again bound the parameters; these are in the order: spatial sill, range, nugget; temporal sill, range, nugget; joint sill, range, nugget; joint anisotropy.

The nuggets and ranges must be bounded below by zero; note that `optim` has no way of knowing that it is fitting a variogram model, which can not have negative values of any parameter. The lower bounds for the marginal sills can be estimated from the 3D wireframe plot of the empirical variogram and from the empirical variogram itself.

We estimate the two marginal sill lower bounds as 70% of the respective maximum marginal semivariances. The interaction sill lower bound is set to zero, in case there is no interaction of the space and time residuals. The range lower bounds are the first bin where the lower bound of the respective marginal sill is exceeded, less one bin.

```
(sp.sill.lb <- 0.7 * max(v.sp$gamma, na.rm=TRUE))

[1] 44.03895

(sp.range.lb <- v.sp[which(v.sp > sp.sill.lb)[1]-1, "spacelag"])

[1] 32.89584

(t.sill.lb <- 0.7 * max(v.t$gamma, na.rm=TRUE))

[1] 63.74037

(t.range.lb <- v.t[which(v.t$gamma > t.sill.lb)[1]-1, "timelag"])

[1] 2
```

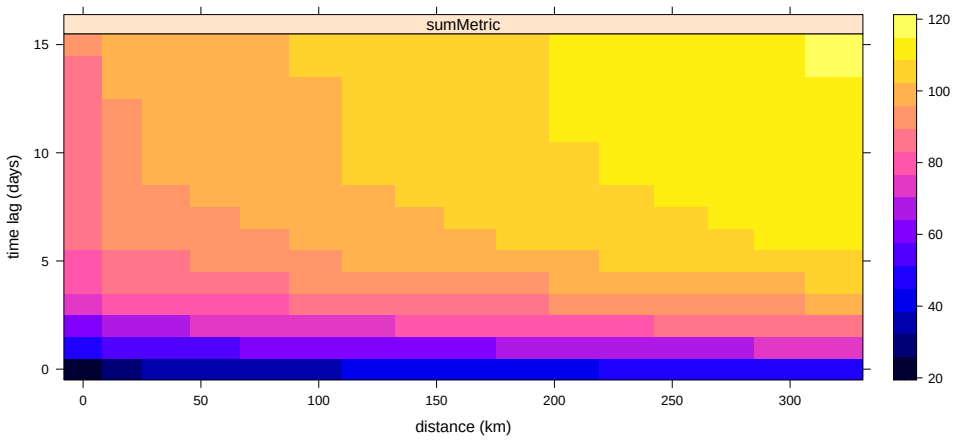
We again time the fitting with the `system.time` function:

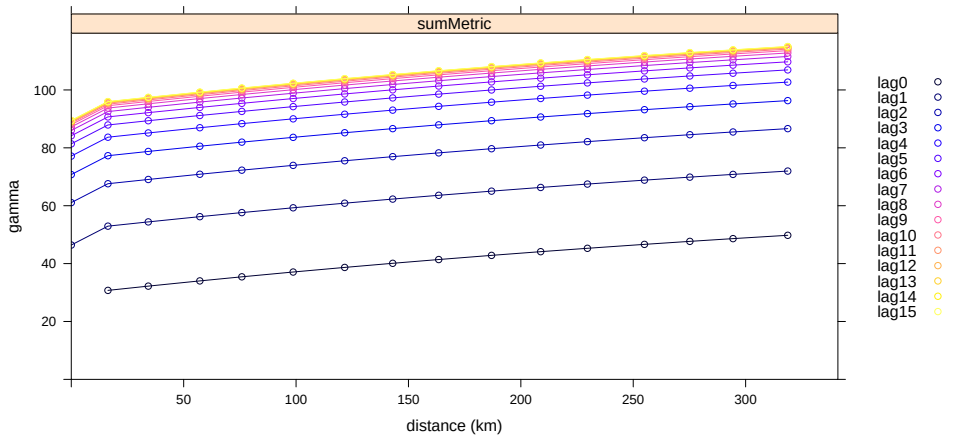
```
system.time(vgmf.sum.metric <-
            fit.StVariogram(vst, vgm.sum.metric,
                            method="L-BFGS-B",
                            control=list(maxit=500),
                            lower=c(sp.sill.lb, sp.range.lb/3,0,
                                t.sill.lb, t.range.lb/3,0,
                                0,1,0,
                                1)))

   user  system elapsed
```

```
 20.313   0.042  20.382

attr(vgmf.sum.metric, "optim.output")$par

    sill.s    range.s   nugget.s     sill.t    range.t   nugget.t
 44.038948 511.785848   5.141498  65.366211   2.410724   0.000000
   sill.st   range.st  nugget.st       anis
  3.701447   1.005772  20.522214  49.557274

attr(vgmf.sum.metric, "optim.output")$value

[1] 60.26628
```

Here are the empirical variograms, along with the fits:

```
plot(vst, vgmf.sum.metric)
plot(vst, vgmf.sum.metric, map=FALSE)
```





Q27 : *Compare the variogram fit from the sum-metric model with the metric and separable models. What features of the empirical variogram does it better represent?* *Jump to A27* •

Challenge: The model fitting by optimization is usually sensitive to the initial parameters and the control list. Experiment with different settings to see how much the fitted parameters change as these are varied.

10.4 Comparing variogram model fits

TASK 38 : Compare the goodness-of-fit of the three models to the empirical variogram. •

The goodness-of-fit is expressed by the error sum of squares in the `value` attribute of the fitted model:

```
attr(vgmf.metric, "optim.output")$value

[1] 83.12474

attr(vgmf.sep, "optim.output")$value

[1] 74.90121

attr(vgmf.sum.metric, "optim.output")$value

[1] 60.26628
```

Q28 : *Which space-time model best fits the empirical space-time variogram?* *Jump to A28*

•

11 Spatio-temporal kriging

We now have several models of space-time dependence, which we can use to predict by kriging, over a user-defined grid of prediction locations. To predict over Germany on any given day, we must assume that the space-time structure as modelled applies to that day and surrounding days, up to the temporal range of the fitted variogram model. That is, we assume second-order stationarity in space, in time, and in space-time interaction as expressed in the covariance model.

We have a spatial grid; to do space-time kriging the grid must have both a space and time component.

TASK 39 : Add a time component, with daily time resolution for six days of your choice in the 2005 – 2010 period, to the spatial grids created in §5. •

I arbitrarily chose 21–26 June 2006 as the time series. Note the use of the / symbol to indicate a date range for the time dimension.

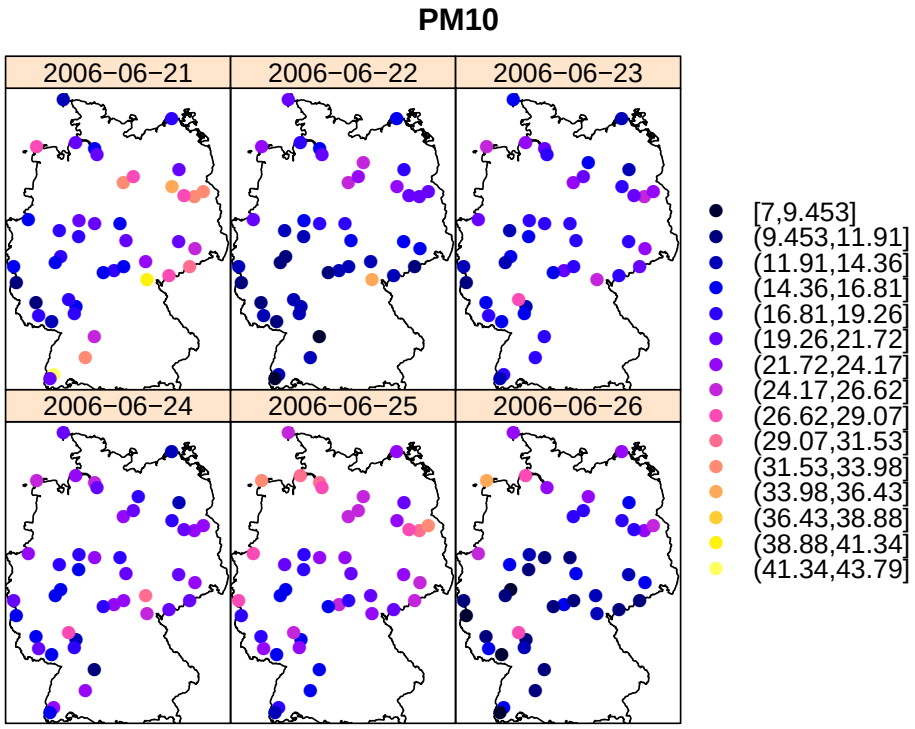
```
rr.t <- r5to10[,"2006-06-21/2006-06-26"]
class(rr.t)

[1] "STFDF"
attr(,"package")
[1] "spacetime"

class(rr.t@time)

[1] "xts" "zoo"
```

TASK 40 : Display the point observations as a coloured post-plot for the six days. •

A time-series of maps contained in a STFDF object can be plotted by the `stplot` function of the `spacetime` package. By default (optional argument `mode` as `"xy"`), one map is produced for each time step.

To help with the geographic visualization, we add the boundary of Germany with the optional `sp.layout` argument to the `spplot` method, which is called by the `stplot` function of the `spacetime` package.

```
stplot(rr.t, col.regions=bpy.colors(64),
       sp.layout=list(list("sp.polygons",de.boundary)))
```



TASK 41 : Create an interpolation grid over the entire bounding box, and just for the portion (partially) covered by Germany. •

The STF method creates a full space-time grid; it requires two arguments: the space and time components:

- For the space component, we already have the Germany grid created in §5;

- for the time component, we have the `@time` slot of the restricted STFDF object; these are the dates for which we want to interpolate over space.

```
rr.t <- as(rr.t,"STSDF")
de.bbox.grid <- STF(sp=as(de.bbox.grid,"SpatialPoints"), time=rr.t@time)
de.grid <- STF(sp=as(de.grid,"SpatialPoints"), time=rr.t@time)
```

> **Note:** The STSDF class is used for for spatio-temporal data with partial (not complete) space-time grids. It differs from class STFDF for full space-time grids by maintaining an index table showing which nodes in the incomplete grid have observations. Since the distances between stations are not regular, we must use STSDF for the observations.

TASK 42 : Predict over the 6-day space-time grid of Germany by kriging with the three fitted space-time models. •

The `krigeST` of the `gstat` package predicts by spatio-temporal kriging, using fitted variogram models as in the previous sections, the original space-time data as an ST object (argument `data`), and the space-time locations of points to be predicted, also as an ST object (argument `newdata`).

> **Note:** The `krigeST` function was designed to exploit separability of spatio-temporal covariance models, and so works most efficiently with full spatio-temporal grids, i.e., of class STFDF. However, here the observations are of class STSDF because the spatial grid is not regular.

```
k.de.met <- krigeST(PM10~1, data=rr.t, newdata=de.grid,
                    modelList=vgmf.metric)
gridded(k.de.met@sp) <- TRUE
```

```
k.de.sep <- krigeST(PM10~1, data=rr.t, newdata=de.grid,
                    modelList=vgmf.sep)
gridded(k.de.sep@sp) <- TRUE
```

```
k.de.sum.metric <- krigeST(PM10~1, data=rr.t, newdata=de.grid,
                    modelList=vgmf.sum.metric)
gridded(k.de.sum.metric@sp) <- TRUE
```

TASK 43 : Display the resulting maps on the same visual scale. •

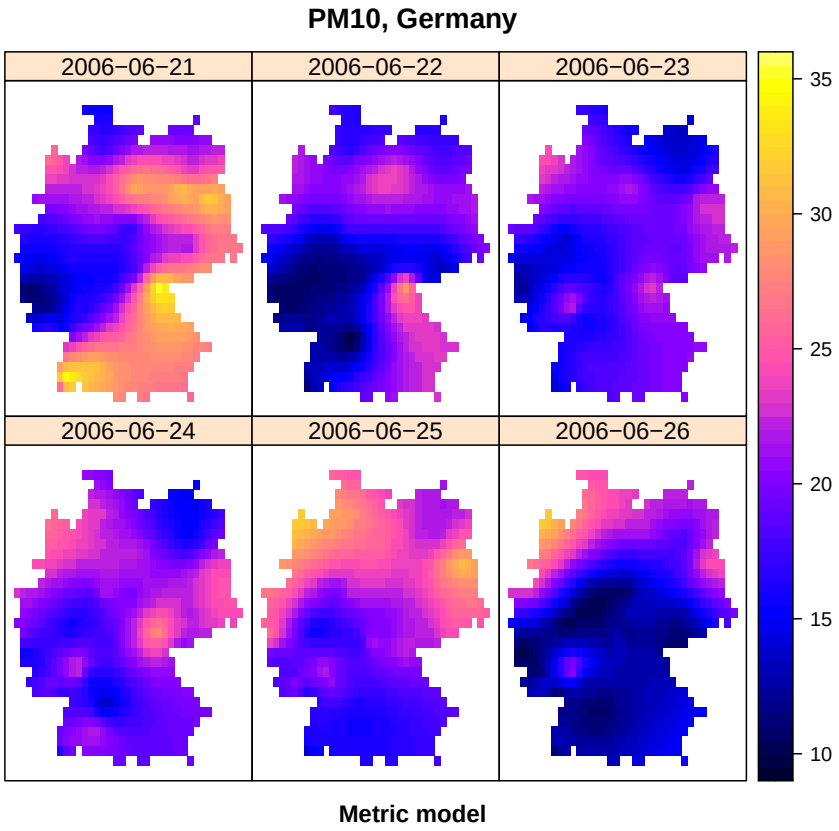
The `stplot` function of the `spacetime` package create trellis plot for ST objects. By default, each time step's result is plotted in a separate panel.

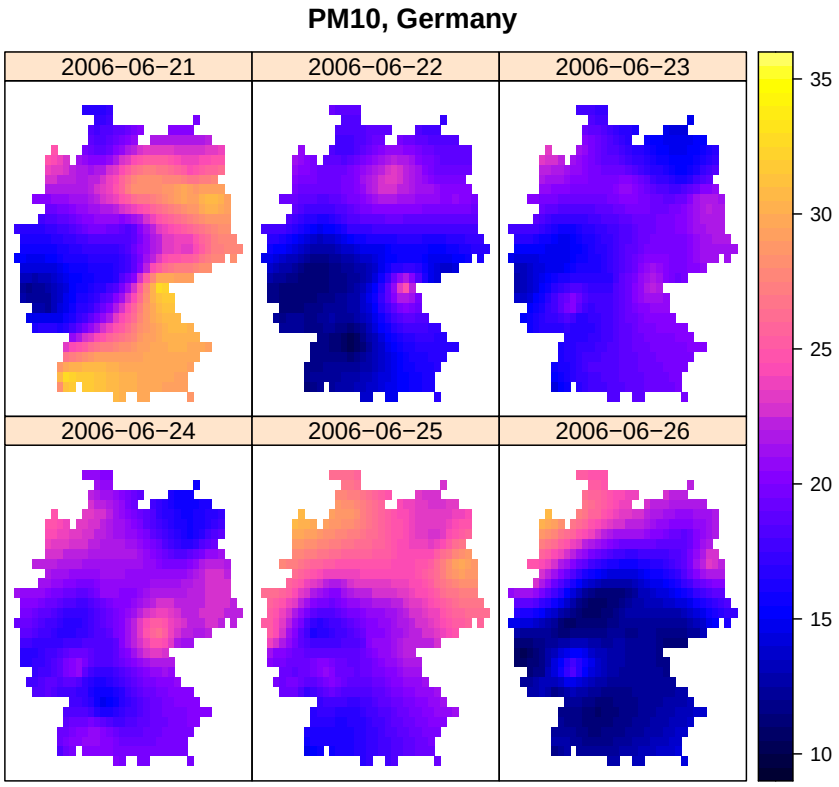
To show the three predictions on the same visual scale, we compute their common maximum and minimum prediction using the `floor` and `ceiling` functions:

```
plot.zlim <- seq(floor(min(k.de.met$var1.pred,
                           k.de.sep$var1.pred,
                           k.de.sum.metric$var1.pred)),
                 ceiling(max(k.de.met$var1.pred,
                             k.de.sep$var1.pred,
                             k.de.sum.metric$var1.pred)),
                 by = 0.5)
```

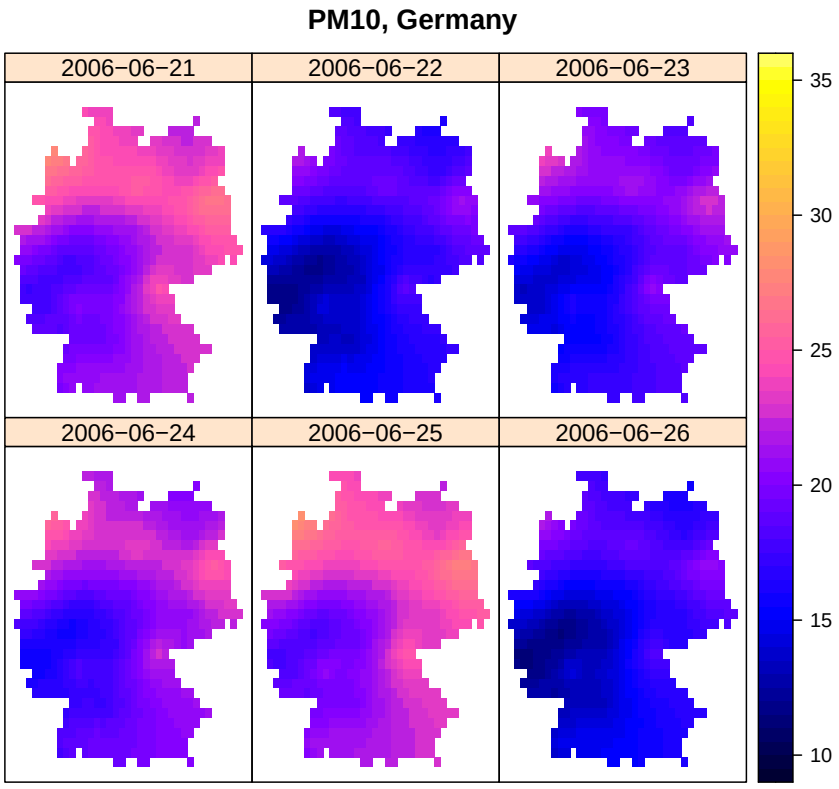
```
stplot(k.de.met, main="PM10, Germany",
       sub="Metric model",
```

```
       col.regions=bpy.colors(length(plot.zlim)),
       at=plot.zlim)
stplot(k.de.sep, main="PM10, Germany",
       sub="Separable space-time model",
       col.regions=bpy.colors(length(plot.zlim)),
       at=plot.zlim)
stplot(k.de.sum.metric, main="PM10, Germany",
       sub="Sum-metric space-time model",
       col.regions=bpy.colors(length(plot.zlim)),
       at=plot.zlim)
```





Separable space−time model



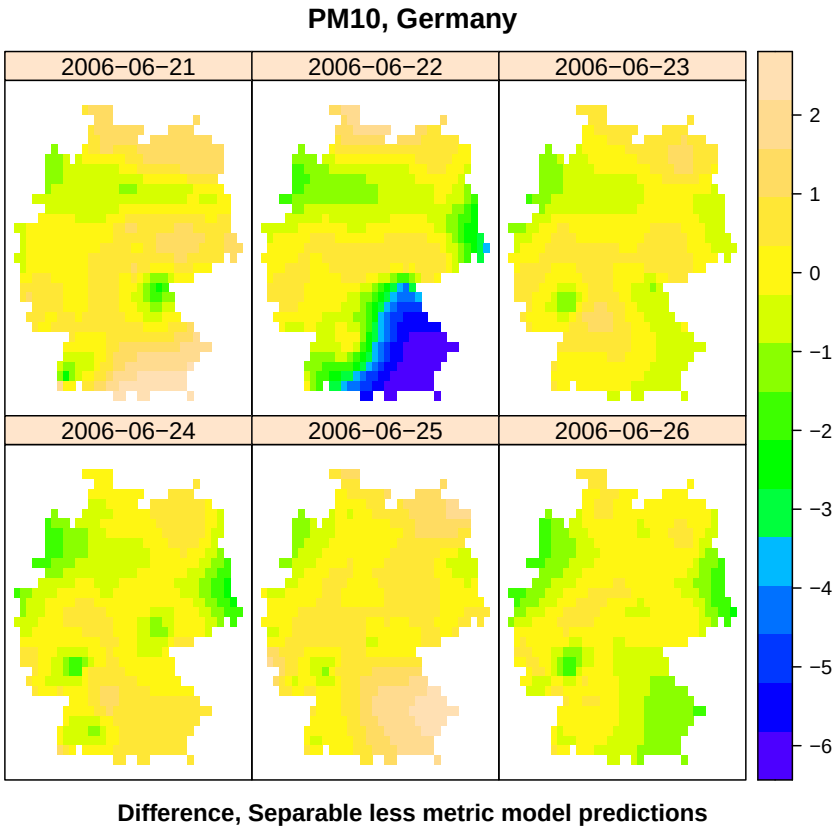
Sum−metric space−time model

TASK 44 : Compare the model predictions. •

For each combination we summarize the differences non-spatially, and then display maps of the differences.

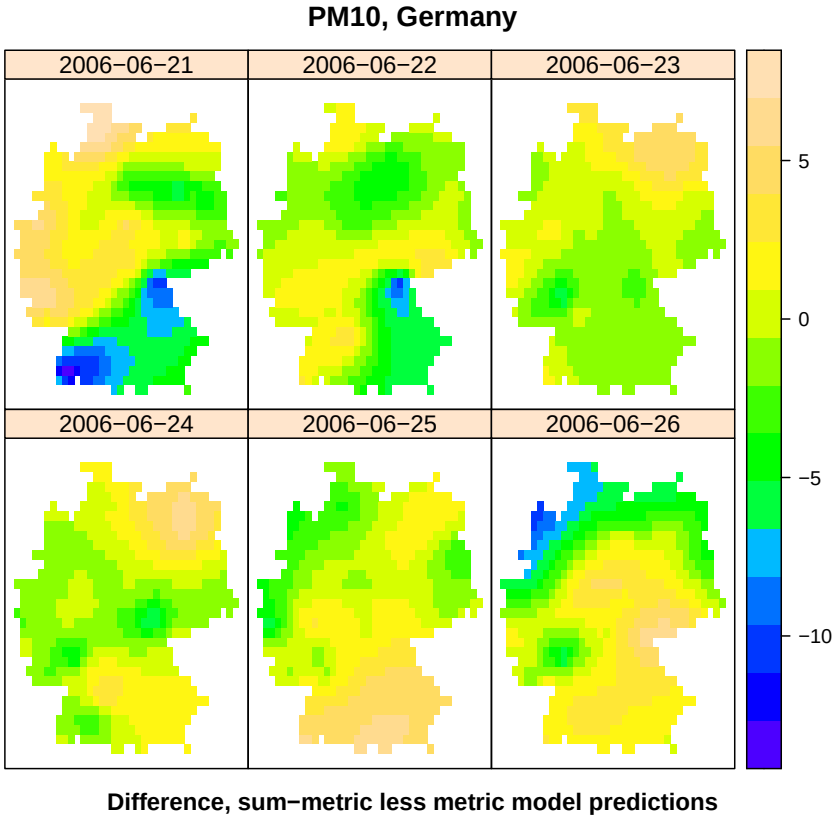
First, metric vs. separable:

```
k.de.diff <- k.de.sep
k.de.diff$var1.pred <- (k.de.sep$var1.pred - k.de.met$var1.pred)
summary(k.de.diff$var1.pred)

    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-6.42541 -0.44205  0.07891 -0.08387  0.50698  2.79610

stplot(k.de.diff, main="PM10, Germany",
       sub="Difference, Separable less metric model predictions",
       col.regions=topo.colors(64))
```



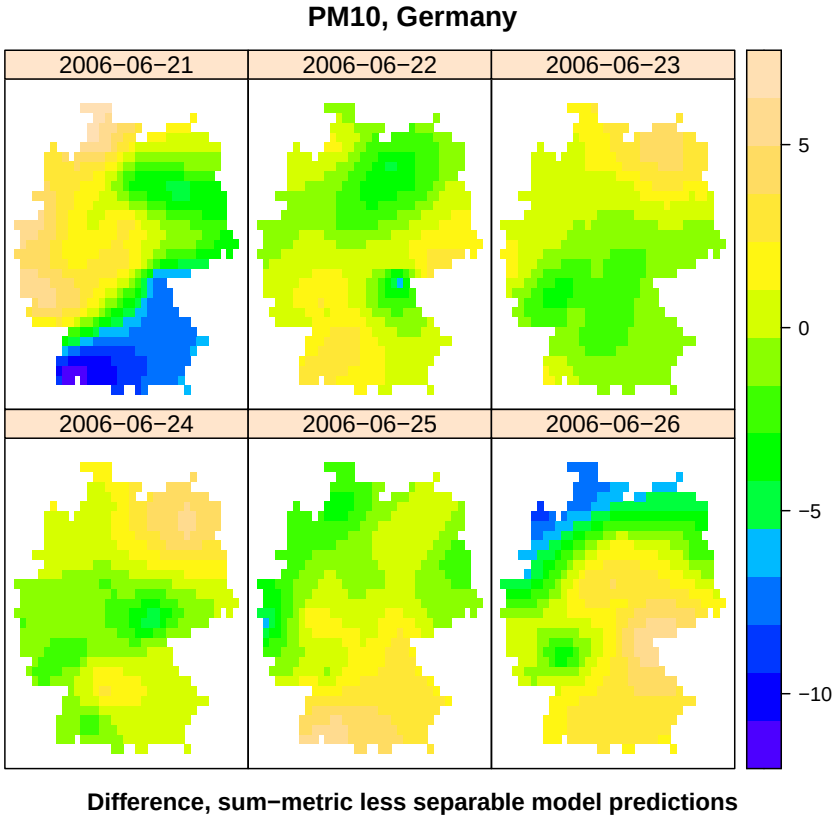
Second, metric vs. sum-metric:

```
k.de.diff <- k.de.sep
k.de.diff$var1.pred <- (k.de.sum.metric$var1.pred - k.de.met$var1.pred)
summary(k.de.diff$var1.pred)

    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-14.1622  -1.6661  -0.1393  -0.1859   1.7087   8.4570

stplot(k.de.diff, main="PM10, Germany",
       sub="Difference, sum-metric less metric model predictions",
       col.regions=topo.colors(64))
```



Finally, sum-metric vs. separable:

```
k.de.diff <- k.de.sep
k.de.diff$var1.pred <- (k.de.sum.metric$var1.pred - k.de.sep$var1.pred)
summary(k.de.diff$var1.pred)

     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
-12.02653  -1.47391   0.05176  -0.10200   1.71715   7.55837

stplot(k.de.diff, main="PM10, Germany",
       sub="Difference, sum-metric less separable model predictions",
       col.regions=topo.colors(64))
```



Q29 : *Describe the principal differences between the three predictions.* *Jump to A29* •

We can see all three predictions, and the observation values, for any one day, by extracting single day kriging predictions.

TASK 45 : Combine the three predictions for 25-June-2006 into the same data frame portion of a `SpatialPixelsDataFrame` object. •

The kriging predictions are of class `STFDF`, but if only one day is selected with the `[]` operator, the result is a `SpatialPixelsDataFrame` object, with only one variable: the kriging prediction. Predictions from other `SpatialPixelsDataFrames` can similarly be extracted, and their single variables, found in the `data` slot, can be added as new columns using the `cbind` "column bind" function.

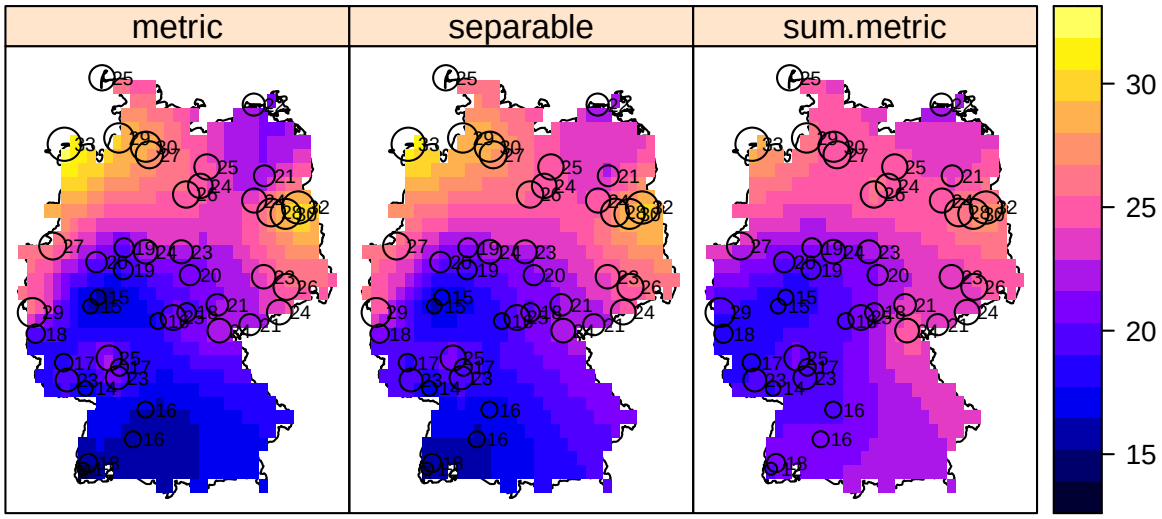
```
class(k.de.met)

[1] "STFDF"
attr(,"package")
[1] "spacetime"

k.de.day <- k.de.met[,"2006-06-25"]
k.de.day@data <- cbind(k.de.day@data, k.de.sep[,"2006-06-25"]@data )
k.de.day@data <- cbind(k.de.day@data, k.de.sum.metric[,"2006-06-25"]@data )
names(k.de.day@data) <- c("metric","separable","sum.metric")
summary(k.de.day)
```

```
Object of class SpatialPixelsDataFrame
Coordinates:
           min  max
coords.x1  5.5 15.5
coords.x2 47.0 55.5
Is projected: FALSE
proj4string :
[+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0]
Number of points: 729
Grid attributes:
          cellcentre.offset cellsize cells.dim
coords.x1               6.0     0.25        37
coords.x2              47.5     0.25        30
Data attributes:
     metric        separable       sum.metric
 Min.   :13.87   Min.   :14.80   Min.   :17.81
 1st Qu.:18.15   1st Qu.:19.16   1st Qu.:20.75
 Median :21.66   Median :22.15   Median :22.85
 Mean   :21.67   Mean   :22.22   Mean   :22.55
 3rd Qu.:24.73   3rd Qu.:25.04   3rd Qu.:24.39
 Max.   :31.88   Max.   :30.83   Max.   :28.11
```

TASK 46 : Display the three kriging predictions side-by-side on the same scale, with the day's point observations overlaid as a post-plot. •

If the `spplot` method is called for a `SpatialPixelsDataFrame` with more than one variable, unless otherwise specified (with the `zcol` argument) it displays a separate panel for each variable, on the same scale. The `sp.layout` argument allows extra elements to be displayed on top of the grid; here we ask for points and a polygon.

```
rr.day <- rr.t["2006-06-25"]
spplot(k.de.day, col.regions=bpy.colors(64),
       sp.layout=list(
           list("sp.points", rr.day, pch=1, col="black",
                cex=2*rr.day$PM10/max(rr.day$PM10)),
           list("sp.polygons",de.boundary, col="black"),
           list("sp.text",coordinates(rr.day),
                round(rr.day$PM10), pos=4,
                col="black", cex=0.6)
           )
       )
```



Q30 : *Which of the three variogram models best fits the selected day's point observations? Why?* *Jump to A30* •

Challenge: Repeat the kriging predictions and comparisons of this section with other dates. Do you reach the same conclusions about which spatio-temporal covariance structure is most appropriate for this dataset?

Challenge: Compute the spatio-temporal variogram for just the selected six-day period, and/or for a two-week period including the selected six days. How different are the fitted variogram models from the models fitted from the two-year dataset? Repeat the kriging prediction but using this model derived from the "time-local" space-time empirical variogram. How different are the predictions? Is it necessary to use models derived from time-local space-time empirical variogram (a moving time window) or can a single model be used? Does this vary by season?

12 Empirical orthogonal functions

Another way to model space-time structure are with Empirical Orthogonal Functions (EOF). These are mathematically the same as PCA (Principal Component Analysis). They are used to find the spatial or temporal patterns of variability of a space-time variable, i.e., spatial patterns over time and temporal patterns over space.

> **Note:** PCA is a data reduction technique. It finds a new set of variables, equal in number to the original set, where these so-called synthetic variables are uncorrelated. In addition, the first synthetic variable represents as much of the common variation of the original variables as possible, the second variable represents as much of the residual variation as possible, and so forth. This technique thus reveals the structure of the data as uncorrelated components. The transformation itself and the synthetic variables produced by it can be interpreted by the analyst to understand the underlying processes. In terms of mathematics, the vector space made up of the original variables is projected onto another space such that the projected variables are orthogonal, with descending variances. The mathematics are well-explained in many texts, e.g., [4, 12].

We use the `wind` dataset (wind speeds at some stations in the Republic of Ireland) provided with the `gstat` package and used as an example in Pebesma [13]. Wind speed is reported as daily average wind speed in knots[11] at 12 stations in the Republic of Ireland. With so few stations, it is impossible to compute spatial variograms, so we turn to another method, namely EOF, to analyze them.

TASK 47 : Load the Irish wind data and examine its structure, convert to a spatial object. •

> **Note:** To see the objects added from the `wind` dataset, we save the names of the objects in the workspace before and after loading, with the `ls` "list objects" command. We then find the newly-defined objects with the `setdiff` "set difference" function.

[11] 1 knot = 0.5418 m s^{-1}

```
tmp1 <- ls()
data("wind", package="gstat")
tmp2 <- ls()
setdiff(tmp2,tmp1)

[1] "tmp1"     "wind"     "wind.loc"

rm(tmp1, tmp2)
str(wind)

'data.frame': 6574 obs. of  15 variables:
 $ year : int  61 61 61 61 61 61 61 61 61 61 ...
 $ month: int  1 1 1 1 1 1 1 1 1 1 ...
 $ day  : int  1 2 3 4 5 6 7 8 9 10 ...
 $ RPT  : num  15 14.7 18.5 10.6 13.3 ...
 $ VAL  : num  14.96 16.88 16.88 6.63 13.25 ...
 $ ROS  : num  13.2 10.8 12.3 11.8 11.4 ...
 $ KIL  : num  9.29 6.5 10.13 4.58 6.17 ...
 $ SHA  : num  13.96 12.62 11.17 4.54 10.71 ...
 $ BIR  : num  9.87 7.67 6.17 2.88 8.21 4.5 8.33 7.29 6.79 6.54 ...
 $ DUB  : num  13.67 11.5 11.25 8.63 11.92 ...
 $ CLA  : num  10.25 10.04 8.04 1.79 6.54 ...
 $ MUL  : num  10.83 9.79 8.5 5.83 10.92 ...
 $ CLO  : num  12.58 9.67 7.67 5.88 10.34 ...
 $ BEL  : num  18.5 17.54 12.75 5.46 12.92 ...
 $ MAL  : num  15 13.8 12.7 10.9 11.8 ...

wind[1:6, 1:12]

  year month day   RPT   VAL   ROS   KIL   SHA  BIR   DUB   CLA   MUL
1   61     1   1 15.04 14.96 13.17  9.29 13.96 9.87 13.67 10.25 10.83
2   61     1   2 14.71 16.88 10.83  6.50 12.62 7.67 11.50 10.04  9.79
3   61     1   3 18.50 16.88 12.33 10.13 11.17 6.17 11.25  8.04  8.50
4   61     1   4 10.58  6.63 11.75  4.58  4.54 2.88  8.63  1.79  5.83
5   61     1   5 13.33 13.25 11.42  6.17 10.71 8.21 11.92  6.54 10.92
6   61     1   6 13.21  8.12  9.96  6.67  5.37 4.50 10.67  4.42  7.17
```

The `wind` table is in so-called **space-wide** format: each column (field in the data frame) refers to one location, and the rows refer to a single time period. This is thus a matrix where each cell is a space-time combination. It can be summarized column-wise per location and row-wise per day.

For the first location the summary statistics are:

```
summary(wind[,"RPT"])

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   0.67    8.12   11.71   12.36   15.92   35.80
```

And for the first day:

```
wind[1,1:3]

  year month day
1   61     1   1

summary(t(wind[1,4:15]))

       1
 Min.   : 9.29
 1st Qu.:10.69
 Median :13.42
 Mean   :13.10
 3rd Qu.:14.98
 Max.   :18.50
```

TASK 48 : Examine the structure of the station locations object, and convert to a spatial object. •

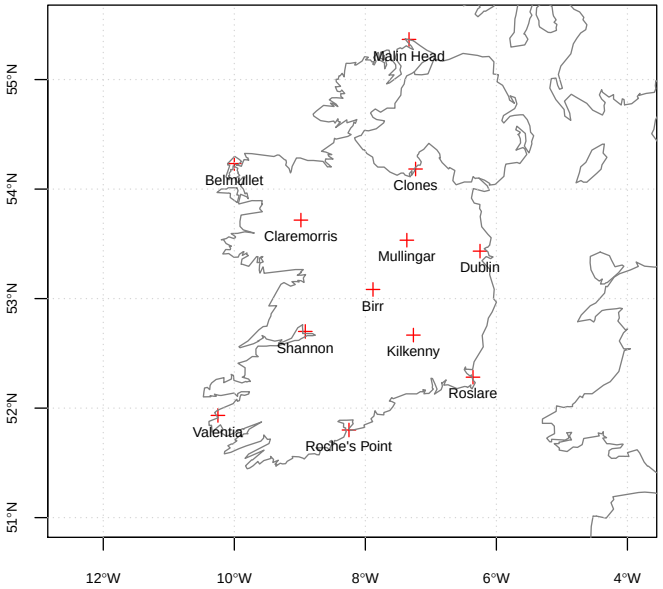
The coördinates are given by the `wind.loc` object also loaded with the `data(wind)` command. These must be converted from a formatted string to numeric decimal degrees, using the `as.character` function to convert a factor to a string, and then the `char2dms` function to interpret this as decimal degrees. Finally, the `coordinates` and `proj4string` commands are used to designate these as coördinates and specify the coördinate system.

```
str(wind.loc)

'data.frame': 12 obs. of  5 variables:
 $ Station  : Factor w/ 12 levels "Belmullet","Birr",..: 12 1 3 11 9 2 8 7 6 4 ...
 $ Code     : Factor w/ 12 levels "BEL","BIR","CLA",..: 12 1 3 11 10 2 8 7 6 4 ...
 $ Latitude : Factor w/ 12 levels "51d48'N","51d56'N",..: 2 11 9 5 1 6 8 12 4 10 ...
 $ Longitude: Factor w/ 12 levels "10d00'W","10d15'W",..: 2 1 12 11 10 9 8 7 6 5 ...
 $ MeanWind : num  5.48 6.75 4.32 5.38 6.36 3.65 4.38 8.03 3.25 4.48 ...

wind.loc$y <- as.numeric(char2dms(as.character(wind.loc[["Latitude"]])))
wind.loc$y[1]

[1] 51.93333

wind.loc$x <- as.numeric(char2dms(as.character(wind.loc[["Longitude"]])))
coordinates(wind.loc) <- ~x+y
proj4string(wind.loc) <- "+proj=longlat +datum=WGS84"
```

TASK 49 : Display a map of the stations in geographic context. •

Again the country boundaries are found in the `worldHires` dataset of the `mapdata` package; here we just plot the whole database, which is automatically restricted to the window defined in the initial `plot` command.

```
plot(wind.loc, xlim = c(-11,-5.4), ylim = c(51,55.5), axes=T, col="red",
       cex.axis =.7)
map("worldHires", add=TRUE, col = grey(.5))
grid()
text(coordinates(wind.loc), pos=1, label=wind.loc$Station, cex=.7)
```



Note: The map is geometrically correct because the projection has been specified. At this latitude 1° latitude is only a bit less than 2° longitude.

TASK 50 : Make a space-time object with both locations and wind speeds. •

The constructor `stConstruct` creates objects of various ST classes from long or wide tables. Here we have a time-wide table. The `space` argument gives the set of columns that contain the spatial variable, here columns 4 through 15. The `time` argument gives the time stamp of each observation. The `SpatialObj` argument gives the name of the time-wide table.

The station names in the time-wide table must be matched with those in the station location dataframe, using the `match` function. The time is given in the object only as two-digit years, so 1900 must be added.

```
stations <- 4:15 # columns 4..15 are the station fields
names(wind[stations])

 [1] "RPT" "VAL" "ROS" "KIL" "SHA" "BIR" "DUB" "CLA" "MUL" "CLO" "BEL"
[12] "MAL"

as.character(wind.loc$Code)

 [1] "VAL" "BEL" "CLA" "SHA" "RPT" "BIR" "MUL" "MAL" "KIL" "CLO" "DUB"
[12] "ROS"

wind.loc <- wind.loc[match(names(wind[stations]),
                           wind.loc$Code),]
row.names(wind.loc) <- wind.loc$Station
wind$time <- ISOdate(wind$year+1900,
```

```
                      wind$month, wind$day, 0)
st.space <- list(values <- names(wind)[stations])
wind.st <- stConstruct(wind[stations],
                       space=st.space, time=wind$time,
                       SpatialObj=wind.loc)
```

This is a long time series, so to save computation time we restrict to the first two years.

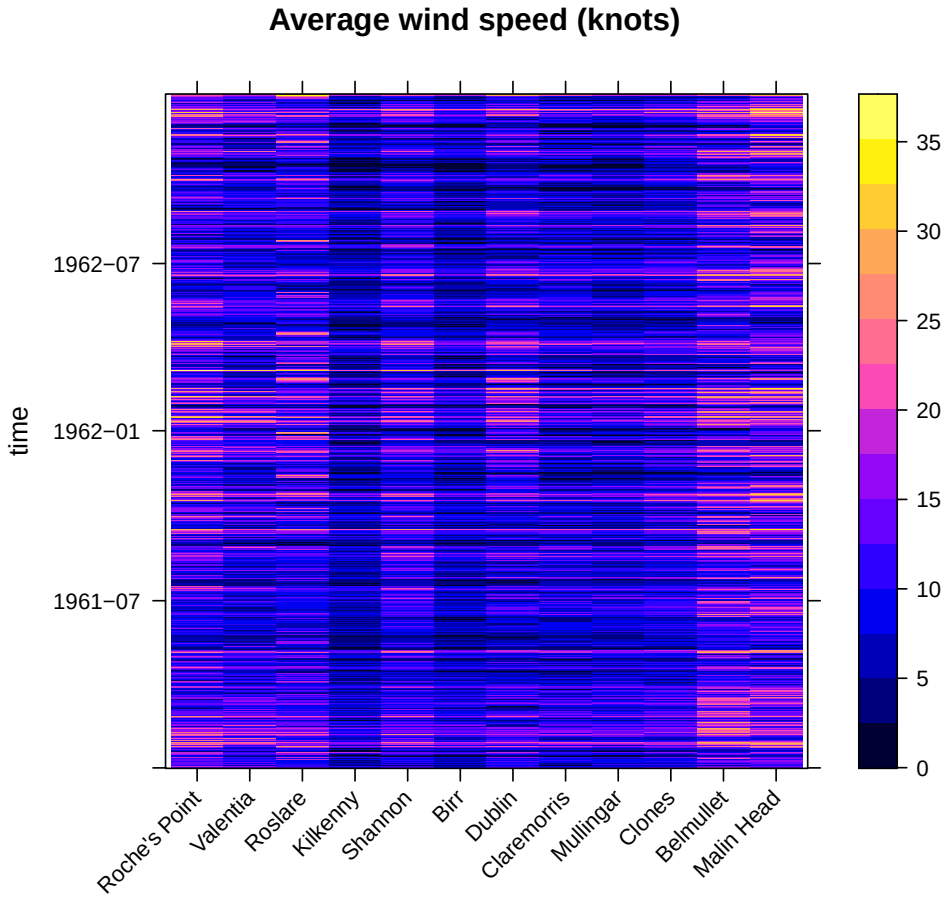
```
wind.st <- wind.st[,1:730]
summary(wind.st)

Object of class STFDF
 with Dimensions (s, t, attr): (12, 730, 1)
[[Spatial:]]
Object of class SpatialPointsDataFrame
Coordinates:
     min      max
x -10.25 -6.25000
y  51.80 55.36667
Is projected: FALSE
proj4string :
[+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0]
Number of points: 12
Data attributes:
        Station       Code             Latitude         Longitude
 Belmullet  :1   BEL    :1   51d48'N       :1   10d00'W      :1
 Birr       :1   BIR    :1   51d56'N       :1   10d15'W      :1
 Claremorris:1   CLA    :1   52d16'56.791"N:1   6d15'W       :1
 Clones     :1   CLO    :1   52d40'N       :1   6d21'25.056"W:1
 Dublin     :1   DUB    :1   52d42'N       :1   7d14'W       :1
 Kilkenny   :1   KIL    :1   53d05'N       :1   7d16'W       :1
 (Other)    :6   (Other):6   (Other)       :6   (Other)      :6
    MeanWind
 Min.   :3.250
 1st Qu.:4.365
 Median :5.215
 Mean   :5.261
 3rd Qu.:6.090
 Max.   :8.030

[[Temporal:]]
     Index                       timeIndex
 Min.   :1961-01-01 00:00:00   Min.   :  1.0
 1st Qu.:1961-07-02 06:00:00   1st Qu.:183.2
 Median :1961-12-31 12:00:00   Median :365.5
 Mean   :1961-12-31 12:00:00   Mean   :365.5
 3rd Qu.:1962-07-01 18:00:00   3rd Qu.:547.8
 Max.   :1962-12-31 00:00:00   Max.   :730.0
[[Data attributes:]]
     values
 Min.   : 0.00
 1st Qu.: 6.34
 Median : 9.59
 Mean   :10.34
 3rd Qu.:13.33
 Max.   :37.63

dim(wind.st)

    space      time variables
       12       730         1
```

TASK 51 : Display a space-time 2D plot of the wind speed per station. •

The `stplot` function which was used above (§11) to show the results of spatio-temporal kriging can also show the space-time relations on a single plot if called with argument `mode="xt"` with the spatial unit (here, the station) on the x-axis, time on the y-axis, and the attribute value shown as a colour ramp with a legend.

> **Note:** The `scales` optional argument to the `xyplot` method of the `lattice` package is a list determining how the x- and y-axes (tick marks and labels) are drawn; here we specify rotation of the labels on the x-axis, because otherwise they would overlap.

```
stplot(wind.st, mode = "xt", scales = list(x=list(rot = 45)),
       xlab = NULL, col.regions=bpy.colors(64),
       main="Average wind speed (knots)")
```



Q31 : *Which station has the lowest wind speed on most days? Which is the most consistent? Which is the most variable? Does there appear to be temporal dependence at most stations? Is it the same at all stations?*
Jump to A31 •

TASK 52 : Compute the spatial and temporal EOF. •

The `eof` function computes the EOF, using the standard `prncmp` PCA function. The EOF are computed in either the spatial or temporal dimension, depending on the `how` argument. By default, it computes *unstandardized* PCs (i.e., using covariances instead of correlations), which seems correct since the predictors are in the same units of measure.

> **Note:** In many applications *standardized* PCA is used, because the variables are on different measurement scale.

The optional `returnEOF` argument specifies what is returned; if `TRUE` (the default), the eigenvectors (EOFs) are returned for each station as objects of class `xts` or one of the `Spatial` classes; if `FALSE`, an object of class `prcomp` "principal components result" as returned by the `prcomp` "compute principal components" function is returned; this can be processed in the usual way for PCA. For example, screeplots (§12.2) and biplots (§12.3) can be produced and interpreted.

The *spatial* EOF use the time-series to characterize stations; this is a matrix with one row for each time (here, day) and one column for each station. The *temporal* EOF use the stations to characterize the time series: this is a matrix with one row for each station and one column for each observation time (here, day).

12.1 Computing EOF

First, the PC scores themselves, for spatial EOF:

```
wind.eof.1 <- eof(wind.st, how="spatial")
str(wind.eof.1) # returns SpatialPointsDataFrame

Formal class 'SpatialPointsDataFrame' [package "sp"] with 5 slots
  ..@ data       :'data.frame': 12 obs. of  12 variables:
  .. ..$ EOF1 : num [1:12] -0.328 -0.268 -0.248 -0.237 -0.299 ...
  .. ..$ EOF2 : num [1:12] -0.437 -0.2402 -0.5292 -0.1518 -0.0689 ...
  .. ..$ EOF3 : num [1:12] 0.1028 0.465 -0.4128 -0.0591 0.2317 ...
  .. ..$ EOF4 : num [1:12] -0.2038 0.0793 0.6178 -0.1114 -0.3543 ...
  .. ..$ EOF5 : num [1:12] -0.50741 -0.21311 0.1473 0.00966 0.10637 ...
  .. ..$ EOF6 : num [1:12] 0.143 0.249 -0.146 -0.272 -0.188 ...
  .. ..$ EOF7 : num [1:12] -0.57081 0.54492 0.12815 -0.00239 0.30717 ...
  .. ..$ EOF8 : num [1:12] 0.172 -0.423 0.134 -0.324 0.504 ...
  .. ..$ EOF9 : num [1:12] -0.02751 -0.22467 0.00571 0.47679 0.11222 ...
  .. ..$ EOF10: num [1:12] 0.0382 0.0811 -0.0815 0.3328 -0.4195 ...
  .. ..$ EOF11: num [1:12] -0.1144 -0.065 -0.0681 0.5851 0.2567 ...
  .. ..$ EOF12: num [1:12] 0.0232 -0.0545 -0.1319 0.2134 -0.2769 ...
  ..@ coords.nrs : num(0)
  ..@ coords     : num [1:12, 1:2] -8.25 -10.25 -6.36 -7.27 -8.92 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:12] "Roche.s.Point" "Valentia" "Roslare" "Kilkenny" ...
  .. .. ..$ : chr [1:2] "x" "y"
  ..@ bbox       : num [1:2, 1:2] -10.25 51.8 -6.25 55.37
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:2] "x" "y"
  .. .. ..$ : chr [1:2] "min" "max"
  ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
  .. .. ..@ projargs: chr "+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0"
```

The spatial EOF is a `SpatialPointsDataFrame` object, with one row per PC, the fields being the principal component scores for each station.

Second, the PC scores for the temporal EOF:

```
wind.eof.2 <- eof(wind.st, how="temporal")
str(wind.eof.2) # returns xts

An 'xts' object on 1961-01-01/1962-12-31 containing:
  Data: num [1:730, 1:12] -0.0369 -0.0417 -0.0317 -0.0312 -0.0275 ...
 - attr(*, "dimnames")=List of 2
  ..$ : NULL
  ..$ : chr [1:12] "EOF1" "EOF2" "EOF3" "EOF4" ...
  Indexed by objects of class: [POSIXct,POSIXt] TZ: GMT
  xts Attributes:
 NULL

summary(wind.eof.2[,1:2])

     Index                          EOF1                EOF2
 Min.   :1961-01-01 00:00:00   Min.   :-0.09914   Min.   :-0.125571
 1st Qu.:1961-07-02 06:00:00   1st Qu.:-0.04397   1st Qu.:-0.031517
 Median :1961-12-31 12:00:00   Median :-0.03270   Median :-0.006705
 Mean   :1961-12-31 12:00:00   Mean   :-0.03388   Mean   :-0.007155
 3rd Qu.:1962-07-01 18:00:00   3rd Qu.:-0.02323   3rd Qu.: 0.018004
 Max.   :1962-12-31 00:00:00   Max.   : 0.01234   Max.   : 0.086714
```

The temporal EOF is an `xts` "extended time series" object, indexed by the time series, with the twelve PC scores for each time as the columns.

Specifying `returnEOFs` to be `FALSE` causes a `prcomp` object to be returned.

```
eof.sp <- eof(wind.st, how="spatial", returnEOFs=FALSE)
eof.t <- eof(wind.st, how="temporal", returnEOFs=FALSE)
```

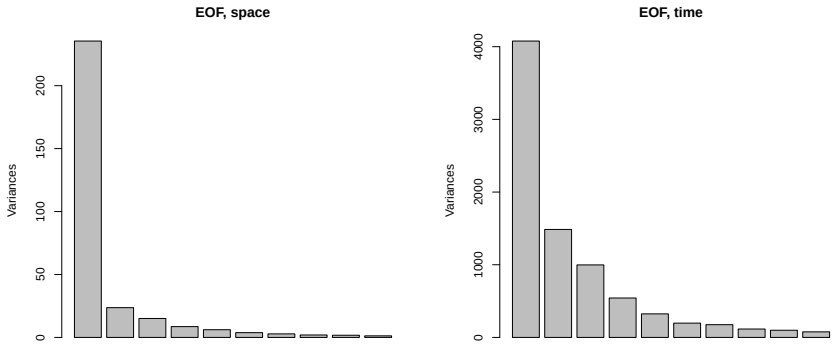
12.2 Interpreting EOF: the scree plot

TASK 53 : Print and graphically represent the cumulative proportion of variance explained by each PC. •

The proportion of the total variance explained by each PC, and the cumulative proportion explained, is shown in the summary of the `prcomp` object, and can be graphically represented by a so-called **scree plot**, which is the default for the `plot` method applied to objects of this class. The spatial PCA has 12 PCs, one for each station; the temporal PCA also has only 12 PCs.

```
summary(eof.sp)

Importance of components:
                          PC1     PC2    PC3     PC4     PC5     PC6
Standard deviation     15.3443 4.86305 3.8810 2.93129 2.47623 1.94827
Proportion of Variance  0.7785 0.07819 0.0498 0.02841 0.02027 0.01255
Cumulative Proportion   0.7785 0.85666 0.9065 0.93487 0.95514 0.96769
                           PC7     PC8     PC9    PC10    PC11   PC12
Standard deviation     1.68166 1.39482 1.32008 1.12165 1.07328 0.9198
Proportion of Variance 0.00935 0.00643 0.00576 0.00416 0.00381 0.0028
Cumulative Proportion  0.97704 0.98347 0.98923 0.99339 0.99720 1.0000

summary(eof.t)

Importance of components:
                          PC1     PC2     PC3      PC4      PC5
Standard deviation     63.8558 38.5457 31.5853 23.29147 18.00348
Proportion of Variance  0.4999  0.1822  0.1223  0.06651  0.03974
Cumulative Proportion   0.4999  0.6821  0.8044  0.87091  0.91065
```

```
                           PC6      PC7      PC8     PC9     PC10
Standard deviation     14.02701 13.24553 10.75123 9.94581 8.75978
Proportion of Variance  0.02412  0.02151  0.01417 0.01213 0.00941
Cumulative Proportion   0.93477  0.95628  0.97045 0.98258 0.99199
                          PC11      PC12
Standard deviation     8.08425 2.716e-14
Proportion of Variance 0.00801 0.000e+00
Cumulative Proportion  1.00000 1.000e+00

plot(eof.sp, main="EOF, space")
plot(eof.t, main="EOF, time")
```



Q32 : *Which EOF has the largest variances? Why?* *Jump to A32* •

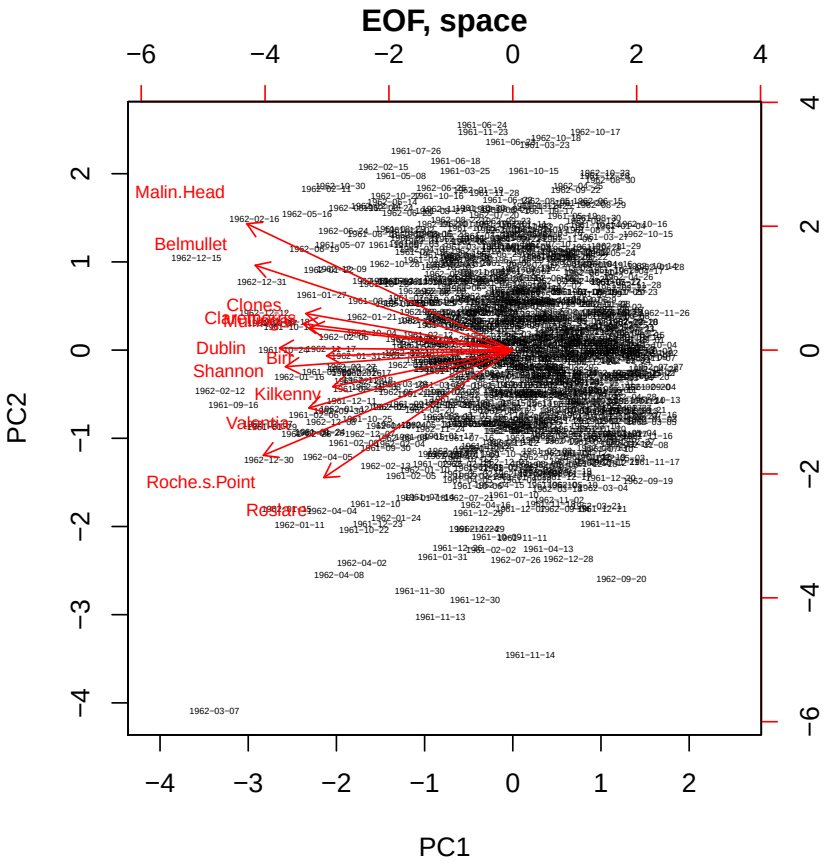
Q33 : *In which of the EOF is most of the variance explained by the first component? In which does it require more components to explain the variance?* *Jump to A33* •

12.3 Interpreting EOF: the biplot

Principal components can be graphically represented by so-called **biplots**, which show the PC scores of any two PCs (by default the first two, i.e., the two with the most explanation of variance) of observations, along with vectors showing how correlated each predictor variable is with each of the two PCs. We specify the `pc.biplot` argument to be `TRUE`, to produce the biplot as proposed by Gabriel [5], which scales the scores of the observations and rotations by the components' standard deviations (eigenvalues).

First the spatial EOF, where the time-series was used to summarize behaviour over space:

```
biplot(eof.sp, pc.biplot=T, main="EOF, space", cex=c(0.3, 0.7))
```



In a biplot, the first set of points (shown in black) are the observations, and the second set (shown in red) are the variables. Here there are only twelve variables, i.e., the spatial locations, but 730 observations, i.e., the dates.

Biplots can be interpreted in several ways:

1. The **orientation** (direction) of the vector, with respect to the PC space. The more a vector, which represents an original variable, is parallel to a PC axis (PC1 at the bottom, PC2 at left), the more it contributes to that PC.

2. The **length** in the space defined by the displayed PCs; the longer the vector, the more variability of this variable is represented by the two displayed PCs.

3. The **angles between vectors** of different variables show their correlation in the space spanned by the two displayed PC's: small angles represent high positive correlation, right angles represent lack of correlation, opposite angles represent high negative correlation.

   The scaled values of the eigenvectors are shown as the top (PC1)

and right (PC2) axes.

Because of the scaling applied, the inner (vector) product between two variables approximates their covariance (if standardized, their correlation). Thus two vectors (here, representing dates) that are almost parallel will have a very high covariance, and two orthogonal vectors will have no covariance.

4. The **location of observations** in the biplot space, defined by their scores scaled by the standard deviation of each PC, i.e., its eigenvalue: this shows the relation of observations to each other and which observations are unusual. These values are shown on the bottom (PC1) and left (PC2) axes.

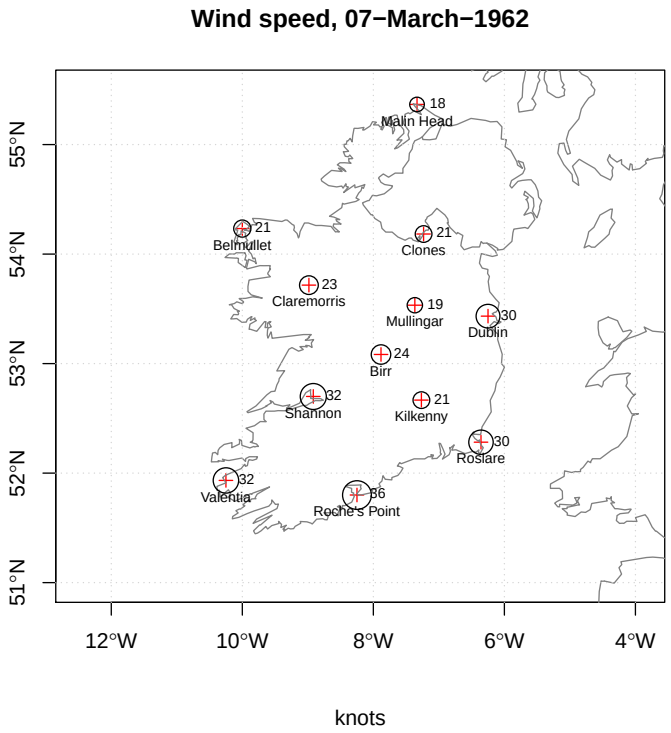
   Because of the scaling applied, the distances between observations plotted in this space approximate the Mahalanobis distance between them; this is a common multivariate measure of similarity between observations.

Here we see the 12 stations all contributing to PC1, which represents the overall windspeed over all days; thus on average all of Ireland has a similar wind speed on a given day. PC2 differentiates Malin Head and Belmullet (the two stations on the NW coast, where the Atlantic storms first reach Ireland) from Roches' Point and Roslare form another group; these are (the two stations on the SE coast); the other stations hardly contribute to PC2. PC1 is easily interpreted as overall windspeed (higher at negative values of PC1). PC2 may be interpreted as consistency of wind speed: more consistent in the SE (negative values of PC2).

The PC scores of the individual dates form a dense cloud. To the left (negative values of PC1) are days with higher windspeed; note the denser cloud, but closer to the origin, for lower windspeed days. Some outliers do not fit the pattern. For example 1962-03-07 had both high windspeed and strong trend.

TASK 54 : Display the wind speeds over Ireland for 1962-03-07 as a postplot, with text showing the speed. •

```
tmp <- wind.st[,"1962-03-7"]
plot(wind.loc, xlim = c(-11, -5.4), ylim = c(51, 55.5), axes = T, col = "red",
     main="Wind speed, 07-March-1962",
     sub="knots")
map("worldHires", add = TRUE, col = grey(0.5))
text(coordinates(wind.loc), pos = 1, label = wind.loc$Station, cex = 0.7)
grid()
points(coordinates(wind.loc), cex = 3*tmp$values/max(tmp$values))
text(coordinates(wind.loc), pos = 4, label=round(tmp$values), cex = 0.7)
```

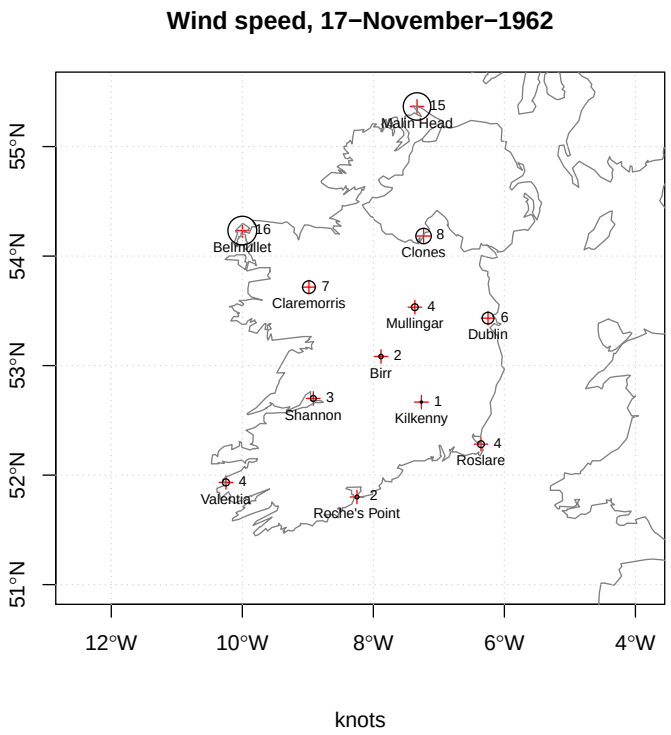


Notice the much stronger winds in the S and E, quite a contrast the usual pattern with Atlantic winds coming from the NW.

Contrast this with the most extreme contrasting date, in the upper-right of the plot, 1962-10-17.

TASK 55 : Display the wind speeds over Ireland for 1962-10-17 as a postplot, with text showing the speed. •

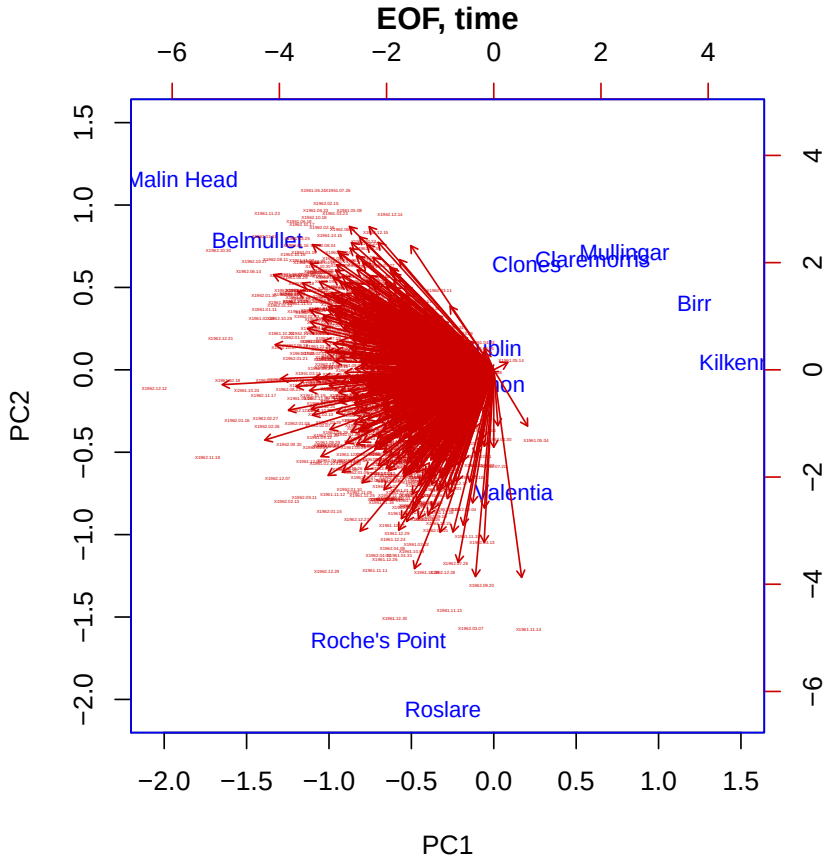
```
tmp <- wind.st[,"1962-10-17"]
plot(wind.loc, xlim = c(-11, -5.4), ylim = c(51, 55.5), axes = T, col = "red",
     main="Wind speed, 17-November-1962",
     sub="knots")
map("worldHires", add = TRUE, col = grey(0.5))
text(coordinates(wind.loc), pos = 1, label = wind.loc$Station, cex = 0.7)
grid()
points(coordinates(wind.loc), cex = 3*tmp$values/max(tmp$values))
text(coordinates(wind.loc), pos = 4, label=round(tmp$values), cex = 0.7)
```



Here the overall windspeed is much lower, and the contrast is from high in the N to light in the S.

Second, the temporal EOF, where the 730 time periods are summarized by their behaviour at the twelve stations.

```
biplot(eof.t, pc.biplot=T, main="EOF, time", cex=c(0.8, 0.2), arrow.len=0.05, col=c("blue","r
```



Challenge: View the biplots for PC3 and PC4 (hint:see optional argument `choices` to `biplot.princomp`) for both space and time EOF, and attempt to interpret their meaning.

13 Answers

A1 : *Object `DE_NUTS1` is of class SpatialPolygonsDataFrame; these are polygons representing statistical district boundaries. Object `rural` is of class STFDF.* *Return to Q1* •

A2 : *The `PM10` attributes in the data frame has 306810 observations; this is the product of the number of points 70 and the number of times 4383. Thus this is a full space-time grid – each location is measured at each time, although some observations may be missing (`NA`).* *Return to Q2* •

A3 : *This is shown in the `proj4string` slot of the `SpatialPoints` object: CRS 4326 in the EPSG database; this is WGS84 geographic coördinates.* *Return*

to Q3 •

A4 *:*

1. *No long-term trend;*
2. *Perhaps an annual cycle, with more PM10 in the summer months;*
3. *Several spikes, especially an extreme value near 01-Jan-2003.*

Return to Q4 •

A5 *: 17 stations were removed (see the dimensions of the two multiple time series).* *Return to Q5* •

A6 *: There are three dimensions: (1) space, (2) time, (3) attributes at a full grid of space and time.*

The ***spatial*** *information is of class* `SpatialPoints` *and just gives the location and CRS of the observations.*

The ***temporal*** *information is an index of class* `POSIXct`*, giving the date of each observation.*

The ***attributes*** *are what is measured at each observation, here PM10 concentration, one observation for each time at each station; here many are missing, i.e., no observation for the given date at the given station.* *Return to Q6* •

A7 *: The temporal autocorrelation is shown for 30 days; by definition it is 1 at lag 0. It decreases to effectively zero after ten days; see the confidence intervals around* $\rho = 0$ *shown by the blue dashed lines. Even after one day the autocorrelation is only about +0.65; this shows that air pollution at this station is not persistent.* *Return to Q7* •

A8 *: The partial autocorrelation is about +0.65 at a lag of +1 day; this is the same as the autocorrelation for lag +1. For longer lags it is not provable different from 0, i.e., all of the autocorrelation can be explained by the single previous days values. This implies that the process has a persistence of one day with a fairly strong one-day correlation, which manifests itself in apparent autocorrelations at longer lags.* *Return to Q8* •

A9 *: The off-diagonals of the graph matrix show the cross-correlations. We consider the upper-right off-diagonal: the forward cross-correlation, i.e., time lag is from the first to the second-listed station, so the second station is later.*

The longest lag with significant cross-correlation is between stations `063` *and* `065` *at about 11 days; these also have a significant positive cross-correlation around lags 14-18. These stations also have the largest lag-0 (instantaneous) cross-correlation, about +0.81.*

Stations `068` *and* `081` *also have strong lag-0 cross-correlation, with significant positive cross-correlation through day 15.*

The weakest cross-correlation is between stations 081 and 064; lag-0 cross-correlation only about 0.48 and significant cross-correlation only till lag 6. *Return to Q9* •

A10 : *The lower-left off-diagonal is the backwards cross-correlation. These can differ from reverse of the corresponding forward cross-correlation. For example, stations 081 and 063 have a forward significant cross-correlation till lag 10 but backward only until lag -4. That is, knowing the PM10 on a given day at station 081 still gives information about the PM10 at station 063 10 days in the future; but only 4 days in the past. Station 081 (Bocholt) is directly to the west of station 063; the strong forward temporal cross-correlation is likely due to the prevailing wind direction.* *Return to Q10* •

A11 : *Maximum: Stations 063 and 064: 228 km apart.*

Minimum: Stations 063 and 068 are only 63 km apart. The strongest cross-correlation does not seem to be between the pair with minimum separation. The cross-correlation between the furthest pair is not the weakest, but is definitely weaker than that for the closest pair. *Return to Q11* •

A12 : *The distribution is strongly right-skewed. Almost all the values are below 50 mg kg^{-1}. The single maximum is over twice the next-highest value.* *Return to Q12* •

A13 : *The maximum PM10 was 269.079, observed at station DEMV017 (in Mecklenburg-Vorpommern, northeastern Germany) on 2009-03-22.* *Return to Q13* •

A14 : *This value seems unlikely, although physically possible, e.g., from a forest fire. The previous day is moderately high (upward trend) and is higher than other peaks in the same time period. However, the following day has again very low PM10. Perhaps PM10 from a fire was then washed out of the air by a rainstorm?* *Return to Q14* •

A15 : *The neighbouring points all have much lower values and are fairly consistent; this value is quite unlikely.* *Return to Q15* •

A16 : *There is strong spatial correlation, e.g., clusters low values in the NE, high values in the centre and SW.* *Return to Q16* •

A17 : *The variogram is fairly consistent, considering that is has only a few stations – note the numbers of point-pairs in each variogram bin. It shows a clear structure and a good model: no nugget, increase in semivariance consistent with an exponential model to an effective sill at about 57 (mg kg^{-1})^2 at a*

computed range of about 434 km. *Return to Q17* •

A18 : *The highest kriging prediction variances are in the SE (Bavaria, away from observation stations). The highest coefficients of variation, however, are found in the NE and centre E, where the predictions are low relative to the uncertainty.* *Return to Q18* •

A19 : *The empirical variogram is well-structured and shows a clear dependence to about 319 km. An exponential model seems appropriate. The nugget is zero, this is consistent with air mixing at short range, assuming good measurement technique. The effective sill is about 60 (mg kg^{-1})^2 . This model is not much different than the single-date model of the previous section, except that the fitted range is only about half of that for the single-date variogram. Although the model forms, sills and nuggets are similar, the discrepancy in ranges brings into question the assumption of similar structure on all dates.* *Return to Q19* •

A20 : *The two maps are very similar; no PM10 prediction differs by more than 2.5 µg m^{-3}, which is an order of magnitude smaller than health limits. Thus using a temporally-lumped variogram model to interpolate on one day seems justified in this case. The higher predictions by the lumped model are in the E and NE, this is because of the reduced range of that model, so the relatively higher measurements in that region have a bit more weight.* *Return to Q20* •

A21 : *The magnitude of the prediction standard deviations is similar, but the lumped model in this case has smaller areas of low standard deviations near observation points. This is because of the longer range of the single-day variogram.* *Return to Q21* •

A22 : *At 143 km average separation and day 5, the semivariance* $\gamma = 88.24$ *(mg kg^{-1})^2 .* *Return to Q22* •

A23 : *1. The averaged spatial dependence at lag 0 (same day) is well-structured, with a low nugget and increasing to about 60 (mg kg^{-1})^2.*

2. As the time lags increase, (1) the nugget increases, (2) the total sill increases, (3) the partial sill (total sill less nugget) however decreases – there is less spatial structure. By 10 days lag there is almost no spatial structure.

3. There is an interesting anomaly at the 20 km separation: at all lags after the first (i.e., from two days onward) there is better correlation at 20 km than at 0 km. *Return to Q23* •

A24 : *The model has a nugget of about 7 (mg kg^{-1})^2, a partial sill of about 98 (mg kg^{-1})^2 at an effective range of about 921 km.*

The anisotropy ratio is 160.2, i.e., one day's lag is equivalent to about 160 km spatial separation. This is well above the estimated starting value, 62.8. *Return*

to Q24 •

A25 : *The model fits the empirical variogram at lag 0 fairly well; it smooths the fluctuations inherent in the empirical variogram from few stations. However it fits quite poorly at longer time lags – the model shows a smoother time decay with space than evidenced by the empirical variogram. We conclude that time does not just act as another dimension with the same structure as space, differing only in units.* *Return to Q25* •

A26 : *The separable model fits much better. We see the spatial structure for lags up to about five days, with the increasing nugget, thus decreasing amount of spatial dependence, similarly to the empirical variogram. Neither model deals with the short-range anomaly (non-monotonic) from lag 2 on. Neither model reproduces the anomalies around 150 and 250 km.* *Return to Q26* •

A27 : *The sum-metric model better represents the decay in spatial dependence with increasing time lag.* *Return to Q27*
•

A28 : *The sum-metric model has the best fit, followed by the separable model; the metric model has the worst fit. Thus the sum-metric covariance model seems to best represent the spatio-temporal process.* *Return to Q28* •

A29 : *The metric and separable models only differ by a few $\mu g\ m^{-3}$, whereas the sum-metric model differs from both by a substantial amount, higher by up to 10 $\mu g\ m^{-3}$ and lower by up to 8 $\mu g\ m^{-3}$. Large differences are seen especially in N Germany on 25-June-2006. The sum-metric model allows for interaction between the spatial and temporal dependence.* *Return to Q29* •

A30 : *On this date both the separable and metric models better capture the very high values in the NW. The sum-metric model best identifies the anomaly near Frankfurt in the centre-SW. It has smaller patches of high and low values than the other two models, which have longer ranges.* *Return to Q30* •

A31 : *Kilkenny has the lowest speeds on almost all days; it also seems the most consistent. Main Head and Roche's Point appear to be the most variable. There is strong temporal dependence, shown by same colours vertically, for several days to a week at most stations, but this seems variable among stations.* *Return to Q31* •

A32 : *The variances are much larger in PC's computed over time than for those computed over space, for this two-year subset. That is, there is much more variability over time, summarized in the 12 stations, than in space, summarized over the long time series.* *Return to Q32*

•

A33 : *In the temporal EOF, the screeplot shows that the first PC has most of the variance, almost 80%. In the spatial EOF, stations are fairly distinct, as shown by the screeplot and cumulative proportion of variance. The 12 stations's behaviour is summarized at the 95% level by the first seven PCs. Only five PCs are needed to account for 95% of the variation among days over the whole time period.* *Return to Q33* •

A Creating space-time objects

In this section we show how to create space-time objects of class `STFDF`, that can then be analyzed as in the previous sections.

TASK 56 : Load tabular data representing a space-time object. •

An example tabular dataset has been created, see §B; this is the daily depth to water table of several sites in the US state of North Carolina in 2013.

```
load("./ds/NCwater/example.RData")
str(ds.nc)

'data.frame': 17113 obs. of  7 variables:
 $ site: Factor w/ 48 levels "335334078352106",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ date: POSIXlt, format: "2012-12-28" ...
 $ mean: num  5.43 5.14 4.94 4.9 4.86 4.84 4.83 4.84 4.84 4.82 ...
 $ long: num  -78.6 -78.6 -78.6 -78.6 -78.6 ...
 $ lat : num  33.9 33.9 33.9 33.9 33.9 ...
 $ alt : num  47.3 47.3 47.3 47.3 47.3 ...
 $ z   : num  41.9 42.1 42.3 42.4 42.4 ...
```

This dataset is a so-called **long** table: each entry has both its coördinates (spatial reference) and its time stamp, here a date in POSIX format, as well as attributes measured at each location at each time, here mean water depth below land surface `mean`, land surface elevation `alt`, and water surface elevation above sea level `z`. Further, each site is identified with a code in field `site`.

TASK 57 : Convert to a `spacetime` object. •

The `stConstruct` function creates this spacetime objects from various formats, including a long table. We need to inform the function which fields in the table are the coördinates and attributes and which field contains the time stamps.

> **Note:** The `stConstruct` function takes the coördinate fields in their order in the source dataframe, not in their order in the argument list; therefore they must be re-ordered in the dataframe before using the function.

```
stds <- stConstruct(ds.nc,
                    space=c("long","lat"),
                    time="date",
                    interval=FALSE)
## this CRS was found by searching the EPSG database
## http://www.epsg-registry.org/
proj4string(stds) <- CRS("+init=EPSG:4269")
str(stds)

Formal class 'STIDF' [package "spacetime"] with 4 slots
  ..@ data   :'data.frame': 17113 obs. of  4 variables:
  .. ..$ site: Factor w/ 48 levels "335334078352106",..: 1 2 3 4 5 6 7 8 9 10 ...
  .. ..$ mean: num [1:17113] 5.43 13.06 9.03 43.68 23.48 ...
  .. ..$ alt : num [1:17113] 47.3 28.1 28.1 28.1 28.3 ...
  .. ..$ z   : num [1:17113] 41.85 15 19.03 -15.6 4.78 ...
  ..@ sp     :Formal class 'SpatialPoints' [package "sp"] with 3 slots
  .. .. ..@ coords     : num [1:17113, 1:2] -78.6 -78.2 -78.2 -78 -78 ...
  .. .. .. ..- attr(*, "dimnames")=List of 2
```

```
  .. .. .. .. ..$ : chr [1:17113] "2" "369" "736" "1103" ...
  .. .. .. .. ..$ : chr [1:2] "long" "lat"
  .. .. ..@ bbox       : num [1:2, 1:2] -83.9 33.9 -76.3 36.3
  .. .. .. ..- attr(*, "dimnames")=List of 2
  .. .. .. .. ..$ : chr [1:2] "long" "lat"
  .. .. .. .. ..$ : chr [1:2] "min" "max"
  .. .. ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
  .. .. .. .. ..@ projargs: chr "+init=EPSG:4269 +proj=longlat +datum=NAD83 +no_defs +ellps=G
  ..@ time   :An 'xts' object on 2012-12-28/2013-12-27 containing:
  Data: int [1:17113, 1] 1 366 731 1096 1461 1824 2189 2554 2882 3247 ...
 - attr(*, "dimnames")=List of 2
  ..$ : NULL
  ..$ : chr "timeIndex"
  Indexed by objects of class: [POSIXlt,POSIXt] TZ: UTC
  xts Attributes:
 NULL
  ..@ endTime: POSIXct[1:17113], format: "2012-12-28" ...
```

The object has slots for the attributes (`@data`), spatial reference (`@sp`), time reference (`@time`), and ending time of the series (`@endTime`).

However, this is of class `STIDF`, "unstructured spatio-temporal data", with full space/time grid, where each entry has both a location and a timestamp. Thus the spatial coördinates are repeated for each time point. In this structure the `sp` and `time` slots need to have the same number of records, and are simply matched by order to identify the location and time of a particular event. The `data` slot has the same number of records and is matched accordingly. There is no need for repeat observations at the same location, nor for a single observation at a point at a single time.

If the grid is regular, the spacetime data have recurrent observations for fixed spatial entities, and each spatial entity (here, points) has the same number of observations. In this case, the data structure should be converted to a `SpatialPoints` object with just the unique coördinates, i.e., a full grid but with only the unique values for the space and time components, linked through their regular structure, as follows. The `sp` and `time` slots typically have different number of records. The assumption every *spatial* record has a *time series* of data of length `nrow(time)`, and that the `STFDF` slot has `length(sp) * nrow(time)` records, space cycling fastest. So if the `STIDF` full grid is set up this way, the conversion to `STFDF` will be correct.

```
stds <- as(stds, "STFDF")
class(stds)

[1] "STFDF"
attr(,"package")
[1] "spacetime"

str(stds)

Formal class 'STFDF' [package "spacetime"] with 4 slots
  ..@ data   :'data.frame': 14965 obs. of  4 variables:
  .. ..$ site: Factor w/ 48 levels "335334078352106",..: 1 2 3 6 7 8 9 10 11 12 ...
  .. ..$ mean: num [1:14965] 5.43 13.06 9.03 3.12 50.03 ...
  .. ..$ alt : num [1:14965] 47.3 28.1 28.1 28 54.5 ...
  .. ..$ z   : num [1:14965] 41.85 15 19.03 24.88 4.51 ...
  ..@ sp     :Formal class 'SpatialPoints' [package "sp"] with 3 slots
  .. .. ..@ coords     : num [1:41, 1:2] -78.6 -78.2 -78.2 -78 -78.1 ...
```

```
  .. .. .. ..- attr(*, "dimnames")=List of 2
  .. .. .. .. ..$ : chr [1:41] "2" "369" "736" "1103" ...
  .. .. .. .. ..$ : chr [1:2] "long" "lat"
  .. .. ..@ bbox       : num [1:2, 1:2] -83.9 33.9 -76.3 36.3
  .. .. .. ..- attr(*, "dimnames")=List of 2
  .. .. .. .. ..$ : chr [1:2] "long" "lat"
  .. .. .. .. ..$ : chr [1:2] "min" "max"
  .. .. ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
  .. .. .. .. ..@ projargs: chr "+init=EPSG:4269 +proj=longlat +datum=NAD83 +no_defs +ellps=G
  ..@ time   :An 'xts' object on 2012-12-28/2013-12-27 containing:
  Data: int [1:365, 1] 1 2 3 4 5 6 7 8 9 10 ...
 - attr(*, "dimnames")=List of 2
  ..$ : NULL
  ..$ : chr "timeIndex"
  Indexed by objects of class: [POSIXlt,POSIXt] TZ: UTC
  xts Attributes:
 NULL
  ..@ endTime: POSIXct[1:365], format: "2012-12-29" ...

summary(stds)

Object of class STFDF
 with Dimensions (s, t, attr): (41, 365, 4)
[[Spatial:]]
Object of class SpatialPoints
Coordinates:
           min       max
long -83.91389 -76.27528
lat   33.89278  36.30833
Is projected: FALSE
proj4string :
[+init=EPSG:4269 +proj=longlat +datum=NAD83 +no_defs
+ellps=GRS80 +towgs84=0,0,0]
Number of points: 41
[[Temporal:]]
     Index              timeIndex
 Min.   :2012-12-28   Min.   :  1
 1st Qu.:2013-03-29   1st Qu.: 92
 Median :2013-06-28   Median :183
 Mean   :2013-06-28   Mean   :183
 3rd Qu.:2013-09-27   3rd Qu.:274
 Max.   :2013-12-27   Max.   :365
[[Data attributes:]]
              site            mean              alt
 335334078352106:  365   Min.   : -0.400   Min.   :  12.0
 335629078115406:  365   1st Qu.:  7.218   1st Qu.:  85.0
 335629078115407:  365   Median : 17.795   Median : 555.0
 335631078003606:  365   Mean   : 24.428   Mean   : 694.9
 335849078054301:  365   3rd Qu.: 35.233   3rd Qu.:1009.0
 (Other)        :12735   Max.   :101.120   Max.   :3148.3
 NA's           :  405   NA's   :493       NA's   :405
       z
 Min.   :   4.19
 1st Qu.:  71.92
 Median : 467.82
 Mean   : 670.38
 3rd Qu.: 990.92
 Max.   :3146.35
 NA's   :493
```

For future use, this is saved as an R data file:

```
save(stds, file="./ds/NCwater/NCwater.RData")
```

B Creating space-time long format data frames from online data sources

The long table imported in §A was prepared from data provided by the United States Geological Survey (USGS). It is typical of the vast amount of freely-available data now available for download. It also presents the typical problems of how to find information, how to combine it, and how to format it. This appendix shows the details of such a procedure for the example long table; it should give some ideas on how to do this in general, for the USGS and other data providers.

The USGS has a wealth of groundwater (and other) data, freely-available for search and download[12]. These are rich sources of space-time data; however, preparing them for analysis in R (or other programs) is tedious and requires careful attention. In this appendix I show how I accessed the USGS groundwater site, selected dates, downloaded it, and re-formatted into an R data frame with the observation date, station coördinates, an the groundwater level for each day at each station.

I selected the state of North Carolina, Water Level/Flow Parameter "Depth to water level, ft below land surface", and the previous 365 days from 28-December-2013, downloaded as tab-separated data with dates in YYYY-MM-DD format. The automatically-created file was named `dv`, which I modified to `dv.txt`. The following R code shows how I processed it to obtain the R dataset used in this tutorial.

There are two files: the water levels and the site information. The water levels file uses the site and date as key; the site is labelled by approximate geographic coördinates and a sequence number. The exact coördinates are found in the site information file. But to download this one must prepare a text file of the sites of interest.

The water levels file looks like[13]:

[12] http://waterdata.usgs.gov/

[13] Some comment lines have been removed for clarity.

```
# Data for the following 92 site(s) are contained in this file
#    USGS 335334078352102 BR-116 CALABASH RS NR CALABASH, NC (BLACK CREEK)
#    USGS 335334078352106 BR-123 CALABASH RS NR CALABASH, NC (SURFICIAL)
...
#
# Data provided for site 335334078352106
#  DD parameter statistic Description
#  01   72019     00001   Depth to water level, feet below land surface (Maximum)
#  01   72019     00002   Depth to water level, feet below land surface (Minimum)
#  01   72019     00003   Depth to water level, feet below land surface (Mean)
#
# Data-value qualification codes included in this output:
#     A  Approved for publication -- Processing and review completed.
#     P  Provisional data subject to revision.
#
agency_cd site_no datetime 01_72019_00001 01_72019_00001_cd 01_72019_00002
01_72019_00002_cd 01_72019_00003 01_72019_00003_cd
5s 15s 20d 14n 10s 14n 10s 14n 10s
USGS 335334078352106 2012-12-28 5.45 A 5.40 A 5.43 A
USGS 335334078352106 2012-12-29 5.40 A 4.94 A 5.14 A
...
```

Read in the water levels file, remove extra headers, fix data types, and save the revised information as an R data file:

The `read.table` function is used to read text files into R; it has many options to match different text file formats. Here we specify the comment character with `comment.char` `read.table` will skip lines beginning with that character. We also specify that there is a header line, and to fill incomplete rows.

```
## comment lines begin with `#'
## some stations have fewer fields (only mean, not min and max)
##    so fill incomplete records
ds.nc <- read.table("./ds/NCwater/dv.txt", comment.char="#",
                    fill=T, header=T)
## easier field names
names(ds.nc) <-
    c("agency","site","date","min","minA","max","maxA","mean","meanA")
## remove header lines for all stations
length(ix <- which(ds.nc$agency == "agency_cd"))

[1] 47

ds.nc <- ds.nc[-ix,]
length(ix <- which(ds.nc$agency == "5s"))

[1] 48

ds.nc <- ds.nc[-ix,]
## type conversions
ds.nc$site <- as.character(ds.nc$site)
ds.nc$min <- as.numeric(as.character(ds.nc$min))
ds.nc$max <- as.numeric(as.character(ds.nc$max))
ds.nc$mean <- as.numeric(as.character(ds.nc$mean))
## if mean is missing, copy the first field "min" to "mean"
## because in this case the first data field is indeed the mean
length(ix <- which(is.na(ds.nc$mean)))

[1] 9107
```

```
ds.nc[ix, "mean"] <- ds.nc[ix, "min"]
## how many records still have no mean water level?
## leave these as NA
length(ix <- which(is.na(ds.nc$mean)))

[1] 106

## keep only fields we need
ds.nc <- ds.nc[,c("site","date","mean")]
str(ds.nc)

'data.frame': 17113 obs. of  3 variables:
 $ site: chr  "335334078352106" "335334078352106" "335334078352106" "335334078352106" ...
 $ date: Factor w/ 367 levels "2012-12-28","2012-12-29",..: 1 2 3 4 5 6 7 8 9 10 ...
 $ mean: num  5.43 5.14 4.94 4.9 4.86 4.84 4.83 4.84 4.84 4.82 ...

length(unique(ds.nc$site))

[1] 48

## write the sites to a text file
write.table(unique(ds.nc$site), file="./ds/NCwater/sites.txt", quote=F,
            col.names=F, row.names=F)
```

Note that all points are on the NAD83 datum (equivalent to WGS84).

Now use this file at the USGS site to get the site information[14]: site coördinates and altitude (to correct the water depths measured from the land surface), as well as the datum. Specify "Site-description information displayed in tab-separated format – saved to file". This file is automatically named `inventory`; we rename it to `inventory.txt`. It looks like[15]:

```
# US Geological Survey
# URL: http://nwis.waterdata.usgs.gov/nc/nwis/inventory
#
#  site_no         -- Site identification number
#  station_nm      -- Site name
#  dec_lat_va      -- Decimal latitude
#  dec_long_va     -- Decimal longitude
#  coord_acy_cd    -- Latitude-longitude accuracy
#  dec_coord_datum_cd -- Decimal Latitude-longitude datum
#  alt_va          -- Altitude of Gage/land surface
#  alt_acy_va      -- Altitude accuracy
#  alt_datum_cd    -- Altitude datum
#
# there are 48 sites matching the search criteria.
#
site_no station_nm dec_lat_va dec_long_va coord_acy_cd dec_coord_datum_cd alt_va
alt_acy_va alt_datum_cd
15s 50s 16n 16n 1s 10s 8s 3s 10s
335334078352106 BR-123 CALABASH RS NR CALABASH, NC (SURFICIAL) 33.89277778
-78.58916670 S NAD83  47.28   1 NGVD29
335629078115406 BR-079 SUNSET HARBOR RS (PEEDEE) 33.94111110 -78.19861110 S NAD83
28.06   1 NGVD29
...
```

[14] See http://waterdata.usgs.gov/nwis/si, accessed from the main page under the "Site Information" button; then select "File of Site Numbers" and upload the just-created site list.

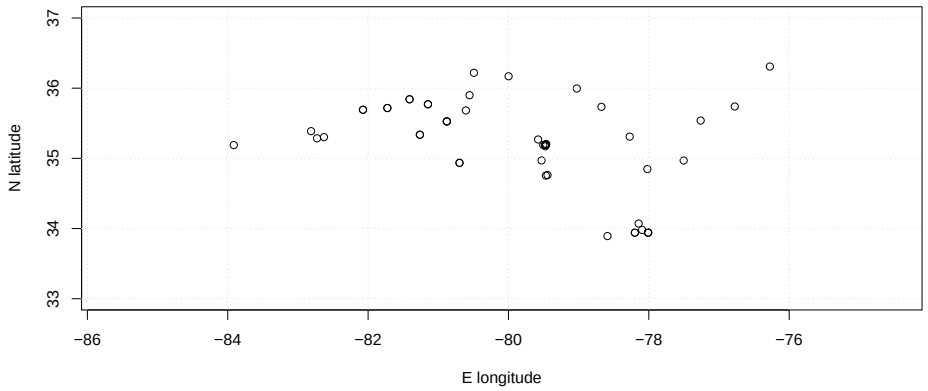
[15] Some comment lines have been removed for clarity

```
sites <- read.table("./ds/NCwater/inventory.txt",
                    sep="\t", comment.char="#", header=T)
names(sites)

[1] "site_no"            "station_nm"         "dec_lat_va"
[4] "dec_long_va"        "coord_acy_cd"       "dec_coord_datum_cd"
[7] "alt_va"             "alt_acy_va"         "alt_datum_cd"

## remove the header line with format specs
sites[1,]

  site_no station_nm dec_lat_va dec_long_va coord_acy_cd
1     15s        50s        16n         16n           1s
  dec_coord_datum_cd alt_va alt_acy_va alt_datum_cd
1                10s     8s         3s          10s

sites <- sites[-1,]
## view the first two sites
sites[1:2,]

          site_no                                      station_nm
2 335334078352106 BR-123 CALABASH RS NR CALABASH, NC (SURFICIAL)
3 335629078115406              BR-079 SUNSET HARBOR RS (PEEDEE)
   dec_lat_va  dec_long_va coord_acy_cd dec_coord_datum_cd alt_va
2 33.89277778 -78.58916670            S              NAD83  47.28
3 33.94111110 -78.19861110            S              NAD83  28.06
  alt_acy_va alt_datum_cd
2          1       NGVD29
3          1       NGVD29

## convert site numbers to strings to match data
sites$site_no <- as.character(sites$site_no)
unique(sites$dec_coord_datum_cd)

[1] NAD83
Levels: 10s NAD83

unique(sites$alt_datum_cd)

[1] NGVD29 NAVD88
Levels: 10s NAVD88 NGVD29
```

Convert the sites file to a `SpatialPointsDataFrame` object, with the USGS site number and station elevation as the attributes, and save as an R data object:

```
sites <- sites[,c("dec_long_va","dec_lat_va","site_no","alt_va")]
names(sites) <- c("long","lat","site","alt")
sites$long <- as.numeric(as.character(sites$long))
sites$lat <- as.numeric(as.character(sites$lat))
sites$alt <- as.numeric(as.character(sites$alt))
coordinates(sites) <- ~long + lat
proj4string(sites) <- CRS("+init=EPSG:4269")
plot(coordinates(sites), asp=1, xlab="E longitude", ylab="N latitude",
     ylim=c(33,37))
grid()
str(sites)

Formal class 'SpatialPointsDataFrame' [package "sp"] with 5 slots
  ..@ data       :'data.frame': 48 obs. of  2 variables:
  .. ..$ site: chr [1:48] "335334078352106" "335629078115406" "335629078115407" "335631078003
  .. ..$ alt : num [1:48] 47.3 28.1 28.1 28.1 28.3 ...
  ..@ coords.nrs : int [1:2] 1 2
  ..@ coords     : num [1:48, 1:2] -78.6 -78.2 -78.2 -78 -78 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:48] "2" "3" "4" "5" ...
  .. .. ..$ : chr [1:2] "long" "lat"
  ..@ bbox       : num [1:2, 1:2] -83.9 33.9 -76.3 36.3
```

```
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:2] "long" "lat"
  .. .. ..$ : chr [1:2] "min" "max"
  ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
  .. .. ..@ projargs: chr "+init=EPSG:4269 +proj=longlat +datum=NAD83 +no_defs +ellps=GRS80 +

save(sites, file="./ds/NCwater/sites.RData")
```



Now we can match the coördinates of the sites with the site numbers, and create coördinate and altitude fields in the attribute data; use the latter to get an elevation above sea level field:

```
unique(ix <- match(ds.nc$site, sites$site))

 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22
[23] 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44
[45] 45 46 47 48

ds.nc$long <- coordinates(sites)[ix,1]
ds.nc$lat <- coordinates(sites)[ix,2]
str(sites)

Formal class 'SpatialPointsDataFrame' [package "sp"] with 5 slots
  ..@ data       :'data.frame': 48 obs. of  2 variables:
  .. ..$ site: chr [1:48] "335334078352106" "335629078115406" "335629078115407" "335631078003
  .. ..$ alt : num [1:48] 47.3 28.1 28.1 28.1 28.3 ...
  ..@ coords.nrs : int [1:2] 1 2
  ..@ coords     : num [1:48, 1:2] -78.6 -78.2 -78.2 -78 -78 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:48] "2" "3" "4" "5" ...
  .. .. ..$ : chr [1:2] "long" "lat"
  ..@ bbox       : num [1:2, 1:2] -83.9 33.9 -76.3 36.3
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:2] "long" "lat"
  .. .. ..$ : chr [1:2] "min" "max"
  ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
  .. .. ..@ projargs: chr "+init=EPSG:4269 +proj=longlat +datum=NAD83 +no_defs +ellps=GRS80 +

ds.nc$alt <- sites@data[ix, "alt"]
ds.nc$z <- ds.nc$alt - ds.nc$mean
## the site is a factor
ds.nc$site <- as.factor(ds.nc$site)
str(ds.nc)

'data.frame': 17113 obs. of  7 variables:
 $ site: Factor w/ 48 levels "335334078352106",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ date: Factor w/ 367 levels "2012-12-28","2012-12-29",..: 1 2 3 4 5 6 7 8 9 10 ...
 $ mean: num  5.43 5.14 4.94 4.9 4.86 4.84 4.83 4.84 4.84 4.82 ...
```

```
 $ long: num  -78.6 -78.6 -78.6 -78.6 -78.6 ...
 $ lat : num  33.9 33.9 33.9 33.9 33.9 ...
 $ alt : num  47.3 47.3 47.3 47.3 47.3 ...
 $ z   : num  41.9 42.1 42.3 42.4 42.4 ...
```

The date field can be converted with the `as.POSIXlt` function:

```
ds.nc$date <- as.POSIXlt(ds.nc$date)
class(ds.nc$date)

[1] "POSIXlt" "POSIXt"

ds.nc$date[2] - ds.nc$date[1]

Time difference of 1 days

str(ds.nc)

'data.frame': 17113 obs. of  7 variables:
 $ site: Factor w/ 48 levels "335334078352106",..: 1 1 1 1 1 1 1 1 1 ...
 $ date: POSIXlt, format: "2012-12-28" ...
 $ mean: num  5.43 5.14 4.94 4.9 4.86 4.84 4.83 4.84 4.84 4.82 ...
 $ long: num  -78.6 -78.6 -78.6 -78.6 -78.6 ...
 $ lat : num  33.9 33.9 33.9 33.9 33.9 ...
 $ alt : num  47.3 47.3 47.3 47.3 47.3 ...
 $ z   : num  41.9 42.1 42.3 42.4 42.4 ...
```

The dataset is now a so-called **long** table: each entry has both its coördinates (spatial reference) and its time stamp, here a date.

TASK 58 : Save as an R data file, for later import with `load`. •

```
save(ds.nc, file="./ds/NCwater/example.RData")
```

References

[1] R. S. Bivand, E. J. Pebesma, and V. Gómez-Rubio. *Applied Spatial Data Analysis with R*. UseR! Springer, 2008. http://www.asdar-book.org/. 1

[2] John M. Chambers. *Programming with Data*. Springer, 1998. ISBN ISBN 0-387-98503-4. 5

[3] Noel A. C. Cressie and Christopher K. Wikle. *Statistics for spatio-temporal data*. Wiley series in probability and statistics. Wiley, 2011. ISBN 9780471692744. 1

[4] J. C. Davis. *Statistics and data analysis in geology*. John Wiley & Sons, New York, 3rd edition, 2002. 62

[5] K. R. Gabriel. The biplot graphic display of matrices with application to principal component analysis. *Biometrika*, 58(3):453–467, 1971. doi: 10.2307/2334381. 70

[6] Tilmann Gneiting, Marc G. Genton, and Peter Guttorp. *Geostatistical Space-Time Models, Stationarity, Separability and Full Symmetry*. Number TR475 in Department of Statistics, University of Washington. 2005. URL http://www.stat.washington.edu/research/reports/2005/tr475.pdf. 1

[7] P Goovaerts. *Geostatistics for natural resources evaluation*. Applied Geostatistics. Oxford University Press, New York; Oxford, 1997. 1

[8] Gabor Grothendieck and Thomas Petzoldt. R Help Desk: Date and time classes in R. *R News*, 4(1):29–32, 2004. 7

[9] Gerard B. M. Heuvelink and Daniel A. Griffith. Space-time geostatistics for geography: A case study of radiation monitoring across parts of Germany. *Geographical Analysis*, 42(2):161–179, 2010. doi: 10.1111/j.1538-4632.2010.00788.x. 48

[10] G. Jost, G. B. M. Heuvelink, and A. Papritz. Analysing the space-time distribution of soil water storage of a forest ecosystem using spatio-temporal kriging. *Geoderma*, 128(3-4):258–273, 2005. 48

[11] Phaedon C. Kyriakidis and André G. Journel. Geostatistical space–time models: A review. *Mathematical Geology*, 31(6):651–684, 1999. doi: 10.1023/A:1007528426688. 1

[12] P Legendre and L Legendre. *Numerical ecology*. Elsevier Science, 1998. 62

[13] Edzer Pebesma. spacetime: Spatio-temporal data in R. *Journal of Statistical Software*, 51(7), 2012. URL http://www.jstatsoft.org/v51/i07. 1, 62

[14] Brian D. Ripley and Kurt Hornik. Date-time classes. *R News*, 1(2): 8–11, 2001. 6, 7

[15] Robert H. Shumway and David S. Stoffer. *Time series analysis and its applications with R examples*. Springer texts in statistics. Springer, New York, 3rd ed edition, 2011. ISBN 9781441978653. 1

[16] R. Webster and M. A. Oliver. *Geostatistics for environmental scientists*. John Wiley & Sons Ltd., 2nd edition, 2008. 1

[17] Yihui Xie. `knitr`: Elegant, flexible and fast dynamic report generation with R, 2011. URL http://yihui.name/knitr/. Accessed 04-Mar-2016. 1

Index of R Concepts