Spatio-temporal overlay and aggregation

ifgi
Institute for Geoinformatics
University of Münster

52north
exploring horizons

Edzer Pebesma

January 25, 2021

Abstract

The so-called "map overlay" is not very well defined and does not have a simple equivalent in space-time. This paper will explain how the `over` method for combining two spatial features (and/or grids), defined in package `sp` and extended in package `rgeos`, is implemented for spatio-temporal objects in package `spacetime`. It may carry out the numerical spatio-temporal overlay, and can be used for aggregation of spatio-temporal data over space, time, or space-time.

Contents

1 Introduction

The so-called *map overlay* is a key GIS operation that does not seem to have a very sharp definition. The `over` vignette in package `sp` (Pebesma, 2012) comments on what paper (visual) overlays are, and discusses the `over` and `aggregate` methods for spatial data.

In the ESRI ArcGIS tutorial (ESRI, 2012), it can be read that

> *An overlay operation is much more than a simple merging of linework; all the attributes of the features taking part in the overlay are carried through, as shown in the example below, where parcels (polygons) and flood zones (polygons) are overlayed (using the Union tool) to create a new polygon layer. The parcels are split where they are crossed by the flood zone boundary, and new polygons created. The FID_flood value indicates whether polygons are outside (-1) or inside the flood zone, and all polygons retain their original land use category values.*

It later on mentions *raster overlays*, such as the addition of two (matching) raster layers (so, potentially the whole of map algebra functions, where two layers are involved).

In the open source arena, with no budgets for English language editing, the Grass 7.0 documentation mentions the following:

> `v.overlay` *allows the user to overlay two vector area maps. The resulting output map has a merged attribute-table. The origin column-names have a prefix (a_ and b_) which results from the ainput- and binput-map. [...] Operator defines features written to output vector map Feature is written to output if the result of operation 'ainput operator binput' is true. Input feature is considered to be true, if category of given layer is defined. Options: and, or, not, xor.*

2 Overlay with method `over`

We loosely define *map overlay* as

- an operation involving at least two maps
- asymmetric – *over*lay is different from *under*lay
- either a *visual* or a *numerical* activity.

The method `over`, as defined in package `sp`, provides a way to numerically combine two maps. In particular,

```
R> over(x, geometry(y))
```

returns an integer vector of length `length(x)` with `x[i]` the index of `y`, spatially corresponding to `x[i]`, so `x[i]=j` means that `x[i]` and `y[j]` match (have the same location, touch, or overlap/intersect etc.), or `x[i]=NA` if there is no match. If `y` has data values (attributes), then

```
R> over(x, y)
```

retrieves a `data.frame` with `length(x)` rows, where row `i` contains the attributes of `y` at the spatial location of `x[i]`, and NA values if there is no match.

If the relationship is more complex, e.g. a polygon or grid cell `x` containing more than one point of `y`, the command

```
R> over(x, y, returnList = TRUE)
```

returns a list of length `length(x)`, with each list element a numeric vector with all indices if `y` is geometry only, or else a data frame with all attribute table rows of `y` that spatially matches `x[i]`.

3 Spatio-temporal overlay with method `over`

Package `spacetime` adds `over` methods to those defined for spatial data in package `sp`:

```
R> library(sp)
R> library(spacetime)
R> showMethods(over)

Function: over (package sp)
x="ST", y="STS"
x="STF", y="STF"
x="STF", y="STFDF"
x="STF", y="STI"
x="STF", y="STIDF"
x="STF", y="STSDF"
x="STI", y="STF"
x="STI", y="STFDF"
x="STI", y="STI"
x="STI", y="STIDF"
x="STI", y="STSDF"
x="STS", y="STF"
x="STS", y="STFDF"
x="STS", y="STI"
x="STS", y="STIDF"
x="STS", y="STSDF"
x="Spatial", y="Spatial"
x="SpatialGrid", y="SpatialGrid"
x="SpatialGrid", y="SpatialGridDataFrame"
x="SpatialGrid", y="SpatialPixels"
x="SpatialGrid", y="SpatialPixelsDataFrame"
x="SpatialGrid", y="SpatialPoints"
x="SpatialGrid", y="SpatialPointsDataFrame"
x="SpatialGrid", y="SpatialPolygons"
x="SpatialGrid", y="SpatialPolygonsDataFrame"
x="SpatialPoints", y="SpatialGrid"
x="SpatialPoints", y="SpatialGridDataFrame"
x="SpatialPoints", y="SpatialPixels"
x="SpatialPoints", y="SpatialPixelsDataFrame"
x="SpatialPoints", y="SpatialPoints"
x="SpatialPoints", y="SpatialPointsDataFrame"
x="SpatialPoints", y="SpatialPolygons"
x="SpatialPoints", y="SpatialPolygonsDataFrame"
x="SpatialPolygons", y="SpatialGrid"
x="SpatialPolygons", y="SpatialGridDataFrame"
x="SpatialPolygons", y="SpatialPoints"
x="SpatialPolygons", y="SpatialPointsDataFrame"
x="SpatialPolygonsDataFrame", y="SpatialPoints"
    (inherited from: x="SpatialPolygons", y="SpatialPoints")
x="xts", y="xts"
```

3.1 Time intervals or time instances?

When computing the overlay

```
R> over(x, y)
```

A space-time feature matches another space-time feature when their spatial locations match (coincide, touch, intersect or overlap), and when their temporal properties match. For temporal properties, it is crucial whether time is a time interval, or a time instance. When all `endTime` values are equal to the `time` times, time is considered instance. When one or more `endTime` values are larger than `time`, time is considered to reflect intervals.

Suppose we have two time sequences, $T : t_1, t_2, ..., t_n$ and $U : u_1, u_2, ..., u_m$. Both are ordered: $t_i \le t_{i+1}$.

Both T and U can reflect time *instances* or time *intervals*. In case they reflect time *instances*, an observation at t_i takes place at the time instance t_i, and has an unregistered (possibly ignorable) duration. In case they reflect time *intervals*, an observation "at" t_i takes place during, or is representatitive for, the time interval $t_i \le t < t_{i+1}$. (The last time interval t_n is obtained by adopting the one-but-last time interval duration: $t_n \le t < t_n + (t_n - t_{n-1})$).

We define the time (instance or interval) pair $\{t_i, u_j\}$ to match if

for T instance, U instance:

$$t_i = u_j$$

for T interval, U instance

$$t_i \le u_j < t_{i+1}$$

for T instance, U interval

$$u_j \le t_i < u_{j+1}$$

for T interval, U interval

$$\exists t : t_i \le t < t_{i+1} \land u_j \le t < u_{j+1}$$

which can be rephrased as the negation of $t_{i+1} \le u_j \lor t_i \ge u_{j+1}$ (where $\lor$ denotes 'or'), or alternatively expressed as

$$t_{i+1} > u_j \land t_i < u_{j+1}$$

where $\land$ denotes 'and'.

All these conditions fail for intervals having zero width (empty intervals), i.e. the case where T is interval and for some i, $t_{i+1} - t_i = 0$ or the case where U is interval and for some j, $u_{j+1} - u_j = 0$.

4 Aggregating spatio-temporal data

The `aggregate` method for a `data.frame` is defined as

```
R> aggregate(x, by, FUN, ..., simplify = TRUE)
```

where `x` is the `data.frame` to be aggregated, `by` indicates how groups of `x` are formed, `FUN` is applied to each group, and `simplify` indicates whether the output should be simplified (to vector), or remain a `data.frame`. The `...` are passed to `FUN`, e.g. passing `na.rm=TRUE` is useful when `FUN` is `mean` and missing values need to be ignored.

For spatio-temporal data, the `x` argument needs to be of class `STFDF`, `STSDF` or `STIDF`. The `by` argument needs to specify an aggregation medium: time, space, or space-time.

4.1 Example data: PM10

Air quality example data are loaded by

```
R> data(air)
R> rural = STFDF(stations, dates, data.frame(PM10 = as.vector(air)))
R> class(rural)
```

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

```
R> class(DE_NUTS1)
```

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

it provides PM10 daily mean values (taken from AirBase - the European Air quality dataBase), for Germany, 1998-2009, where only stations classified as *rural background* were selected. The object `DE_NUTS1` contains NUTS-1 level state boundaries for Germany, downloaded from GADM.

4.2 Spatial aggregation

To aggregate *completely* over space, we can coerce the data to a matrix and apply a function to the rows:

```
R> x = as(rural[,"2008"], "xts")
R> apply(x, 1, mean, na.rm=TRUE)[1:5]
```

```
2008-01-01 2008-01-02 2008-01-03 2008-01-04 2008-01-05
  17.34950   16.06945   25.60065   27.24141   24.03417
```

A more refined spatial aggregation of time series can be obtained by grouping them to the state ("Bundesland") level. Here, states are passed as a `SpatialPolygons` object:

```
R> dim(rural[,"2008"])
```

```
    space      time variables
       70       366         1
```

```
R> x = aggregate(rural[,"2008"], DE_NUTS1, mean, na.rm=TRUE)
R> dim(x)
```

```
   space      time variables
      13       366         1
```

```
R> summary(x)
```

```
Object of class STFDF
 with Dimensions (s, t, attr): (13, 366, 1)
[[Spatial:]]
Object of class SpatialPolygonsDataFrame
Coordinates:
        min      max
x  5.871619 15.03811
y 47.269858 55.05653
Is projected: FALSE
proj4string : [+proj=longlat +datum=WGS84 +no_defs]
Data attributes:
      ID_0     ISO         NAME_0        ID_1
 Min.   :60   DEU:13   Germany:13   Min.   :753.0
 1st Qu.:60                         1st Qu.:756.0
 Median :60                         Median :761.0
 Mean   :60                         Mean   :760.5
 3rd Qu.:60                         3rd Qu.:764.0
 Max.   :60                         Max.   :768.0

    NAME_1                               VARNAME_1 NL_NAME_1
 Length:13          Bavaria                   :1   NA's:13
 Class :character   Hesse                     :1
 Mode  :character   Lower Saxony              :1
                    Mecklenburg-West Pomerania:1
                    North Rhine-Westphalia    :1
                    (Other)                   :3
                    NA's                      :5
     HASC_1    CC_1      TYPE_1   ENGTYPE_1    VALIDFR_1    VALIDTO_1
 DE.BE  :1   NA's:13   Land:13   State:13   Unknown:13   Present:13
 DE.BR  :1
 DE.BW  :1
 DE.BY  :1
 DE.HE  :1
 DE.MV  :1
 (Other):7
 REMARKS_1   Shape_Leng       Shape_Area
 NA's:13   Min.   : 2.631   Min.   :0.1172
           1st Qu.:14.529   1st Qu.:2.1541
           Median :16.891   Median :2.6645
           Mean   :18.068   Mean   :3.3126
           3rd Qu.:24.519   3rd Qu.:4.3832
           Max.   :32.255   Max.   :8.6561

[[Temporal:]]
     Index              timeIndex
```

```
 Min.   :2008-01-01   Min.   :3653  
 1st Qu.:2008-04-01   1st Qu.:3744  
 Median :2008-07-01   Median :3836  
 Mean   :2008-07-01   Mean   :3836  
 3rd Qu.:2008-09-30   3rd Qu.:3927  
 Max.   :2008-12-31   Max.   :4018  
[[Data attributes:]]
      PM10       
 Min.   : 2.181  
 1st Qu.: 9.933  
 Median :13.755  
 Mean   :15.023  
 3rd Qu.:18.371  
 Max.   :68.750  
 NA's   :366     
```

```
R> stplot(x, mode = "tp")
```

the result of which is shown in figure 1, which was created by

```
R> stplot(x, mode = "tp", par.strip.text = list(cex=.5))
```

An aggregation for all stations selected within a single area is obtained by using the country boundary DE, and aggregating the observations within Germany for each moment in time:

```
R> x = aggregate(rural[,"2008"], DE, mean, na.rm=TRUE)
R> class(x)

[1] "xts" "zoo"

R> plot(x[,"PM10"])
```

the plot of which is shown in figure 2.

4.3 Temporal aggregation

To aggregate *completely* over time, we can coerce the data to a matrix and apply a function to the columns:

```
R> x = as(rural[,"2008"], "xts")
R> apply(x, 2, mean, na.rm=TRUE)[1:5]

 DESH001  DENI063  DEUB038  DEBE056  DEBE062 
     NaN 18.41594      NaN 20.76446      NaN 
```

Aggregating values *temporally* is done by passing a character string or a function to the by argument. For monthly data, we will first select those stations that have measured (non-NA) values in 2008,

```
R> sel = which(!apply(as(rural[,"2008"], "xts"), 2, function(x) all(is.na(x))))
R> x = aggregate(rural[sel,"2008"], "month", mean, na.rm=TRUE)
R> stplot(x, mode = "tp")
```

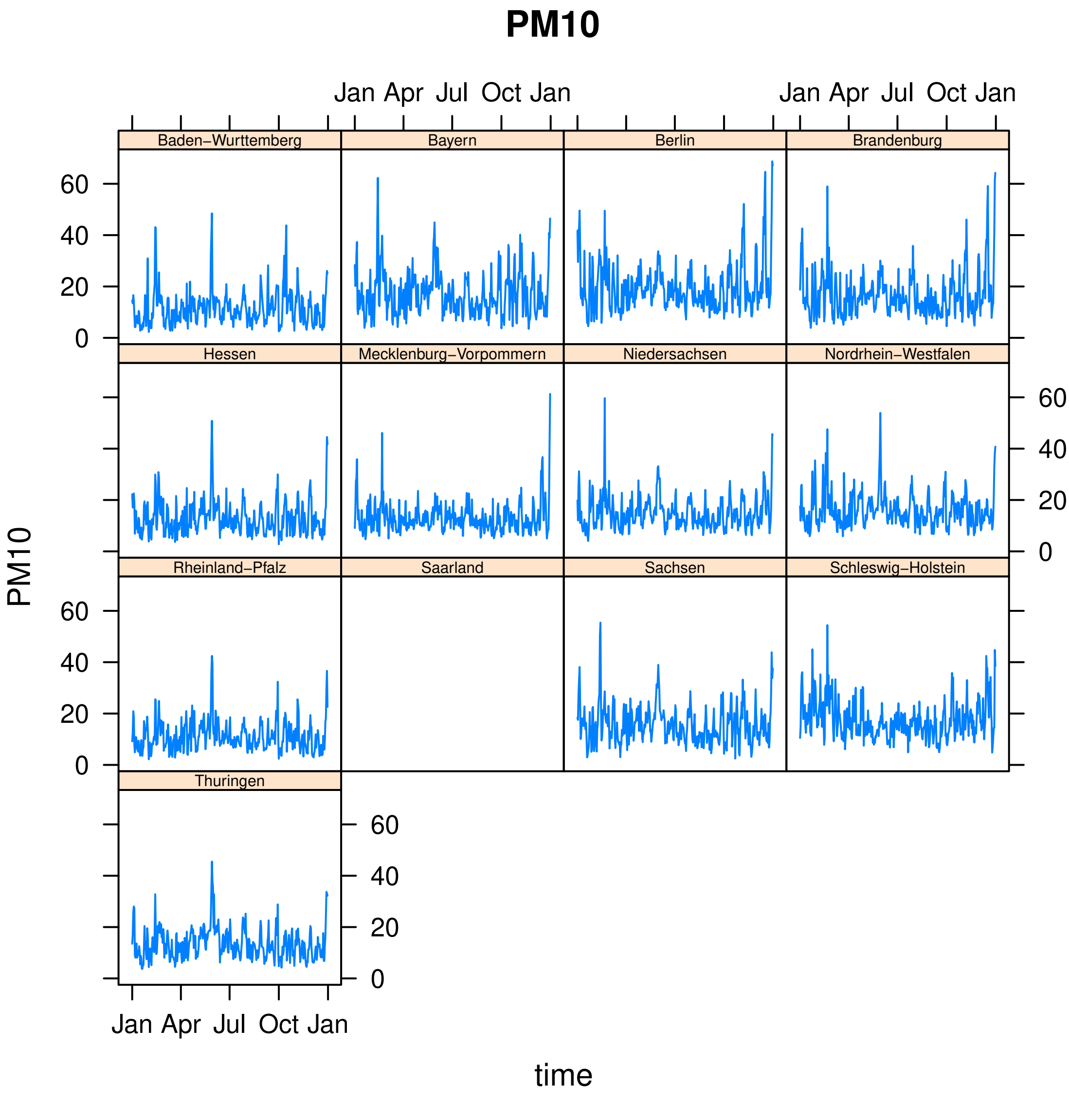

Figure 1: Daily PM10 values, aggregated (averaged) over states

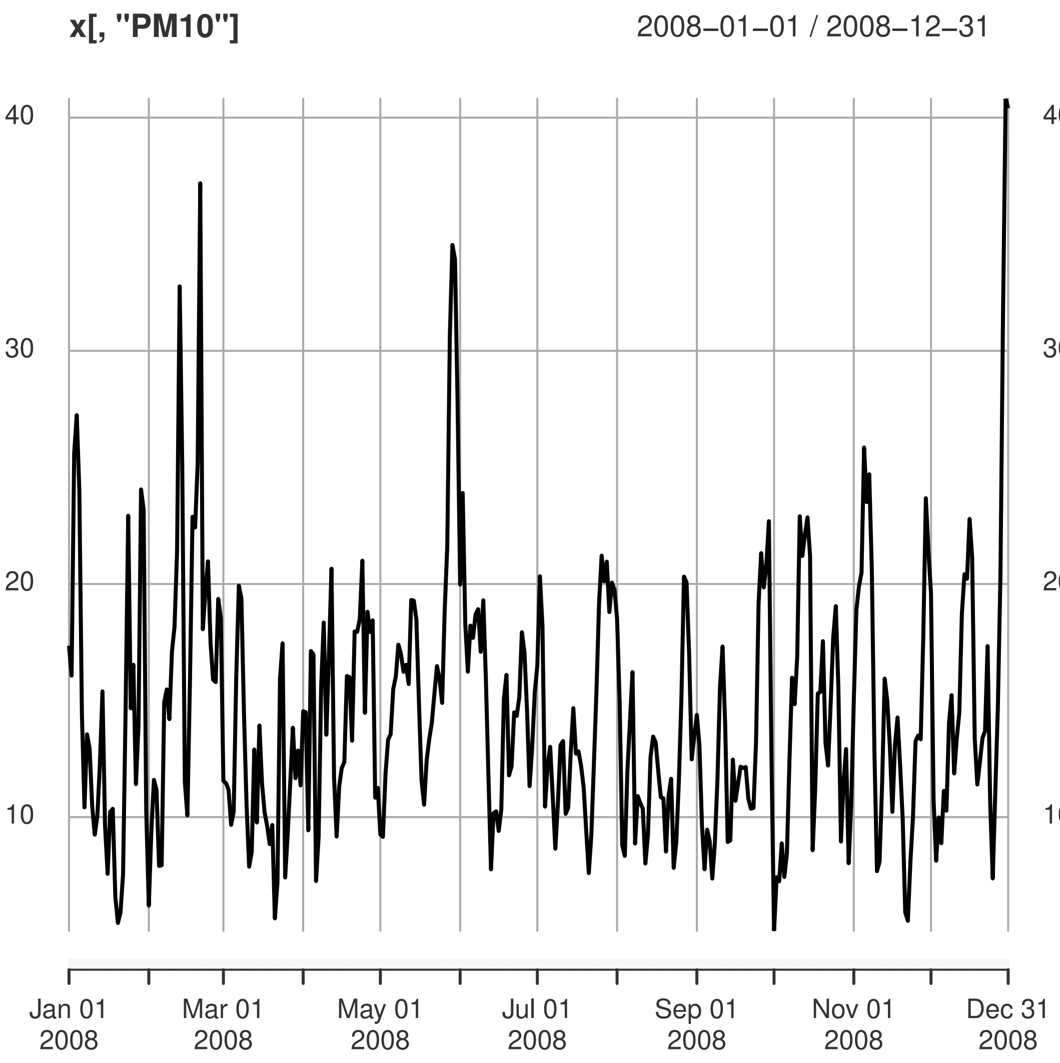

Figure 2: Time series plot of daily rural background PM10, averaged over Germany

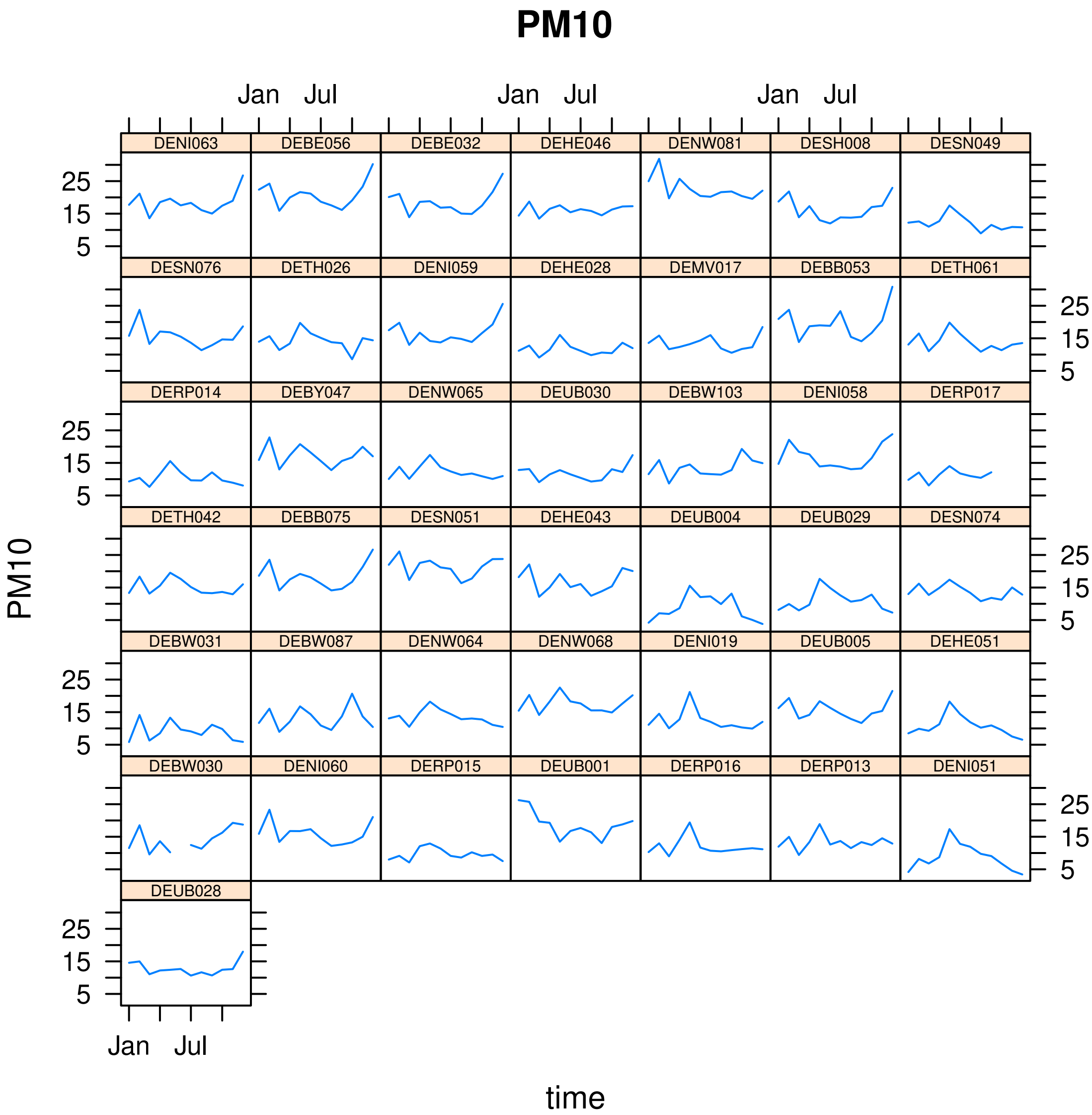

Figure 3: Monthly averaged PM10 values, for those rural background stations in Germany having measured values

shown in figure 3

The strings that can be passed are e.g. "year", but also "3 days". See ?cut.Date for possible values. Aggregation using this way is only possible if the time index is of class Date or POSIXct.

An alternative is to provide a function for temporal aggregation. The function as.yearqtr from package zoo transforms dates to quarters, and hence allows aggregation *to* quarterly values, in this example medians:

```
R> library(zoo)
R> x = aggregate(rural[sel,"2005::2011"], as.yearqtr, median, na.rm=TRUE)
R> stplot(x, mode = "tp")
```

shown in figure 4. Aggregating to monthly values is obtained by function as.yearmon, aggregating to years by creating the function

```
R> as.year <- function(x) as.numeric(floor(as.yearmon(x)))
```

Further information can be found in ?aggregate.zoo, which is the function used to do the processing.

4.4 Spatio-temporal aggregation

Aggregation over spatio-temporal volumes can be done by passing an object inheriting from ST to the by argument:

```
R> DE.years = STF(DE, as.Date(c("2008-01-01", "2009-01-01")))
R> aggregate(rural[,"2008::2009"], DE.years, mean, na.rm=TRUE)
```

```
               PM10 timeIndex
2008-01-01 14.56871         1
2009-01-01 15.23961         2
```

4.5 Time intervals

If all data concern time instances (endTime equals time), then time instances are being matched, else overlapping time intervals are being matched. In case intervals are being matched, empty intervals are never matched.

References

- ESRI (2012) ESRI ArcGIS Tutorial. http://webhelp.esri.com/arcgisdesktop/9.2/index.cfm?TopicName=Overlay_analysis
- Pebesma, 2012. Map overlay and spatial aggregation in sp. Vignette in package sp, https://cran.r-project.org/web/packages/sp/vignettes/over.pdf

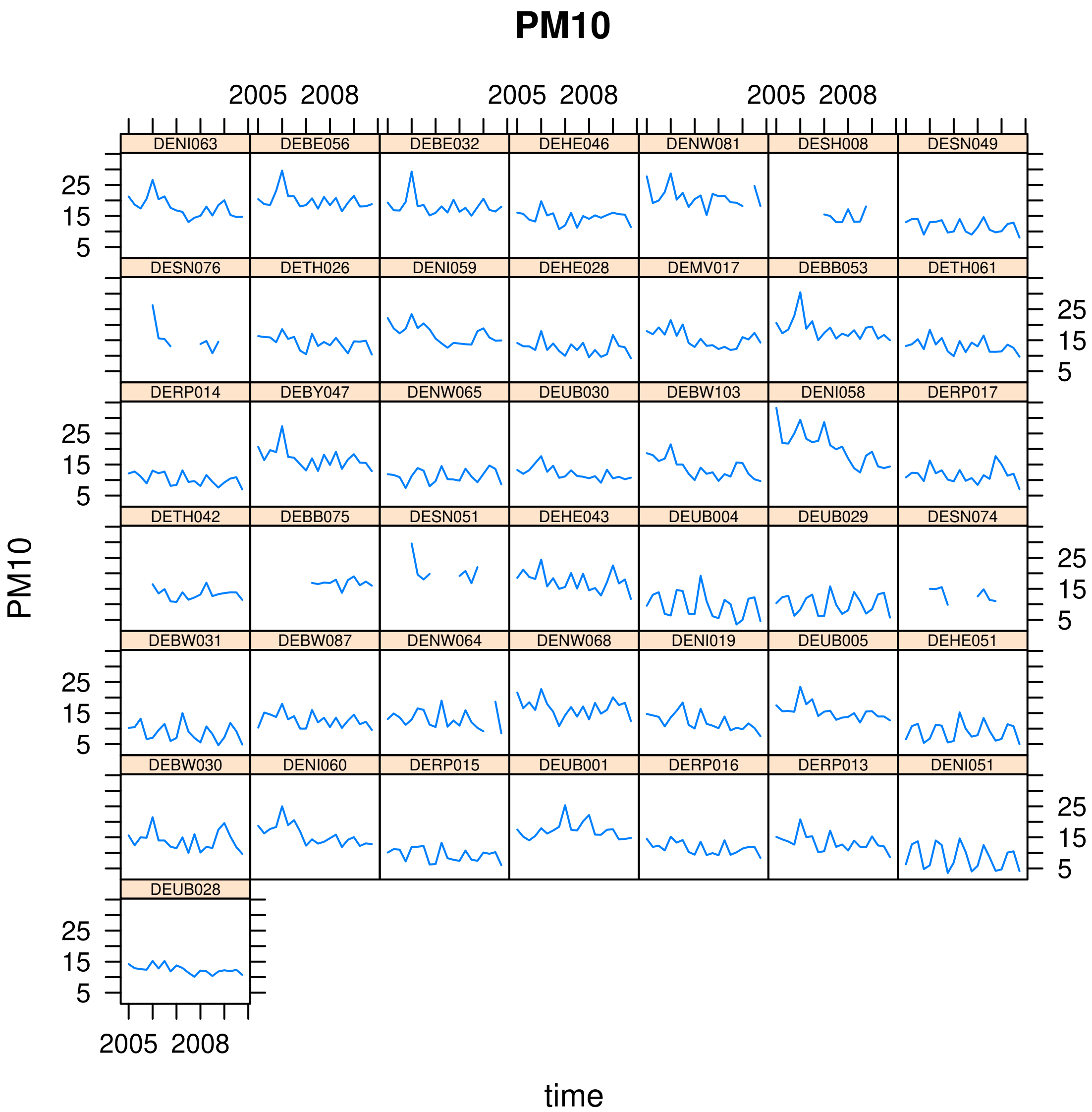

Figure 4: PM10 values, averaged to quarterly medians of daily averages