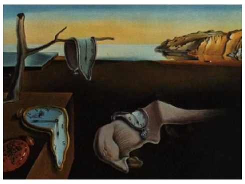

Journal of Statistical Software

MMMMMM YYYY, Volume~VV, Issue~II. *http://www.jstatsoft.org/*

Spatio-Temporal Data in R

Edzer Pebesma
Institute for Geoinformatics
University of Münster

Abstract

This document describes classes and methods designed to deal with different types of spatio-temporal data in R implemented in the R package **spacetime**, and provides examples for analyzing them. It builds upon the classes and methods for spatial data taken from package **sp**, and all temporal classes supported by package **xts**. The goal is to cover a number of useful representations for spatio-temporal sensor data, and results from predicting (spatial and/or temporal interpolation or smoothing), aggregating, or subsetting them. The goals of this package are to explore how spatio-temporal data can be sensibly represented in classes, and to find out which analysis and visualisation methods are useful and feasible for the classes implemented. It reuses existing classes, methods, and functions present in packages for spatial data (**sp**) and time series data (**zoo** and **xts**). Coercion to the appropriate reduced spatial and temporal classes is provided, as well as to `data.frame` objects in the long, time-wide and space-wide formats. We discuss when it is appropriate to store temporal data as just a start time or as a time interval. This document is the main reference for the R package `spacetime`, and is available (in updated form) as a vignette in this package.

Keywords:~Time series analysis, spatial data, spatio-temporal statistics, GIS.

1. Introduction

Spatio-temporal data are abundant, and easily obtained. Examples are satellite images of parts of the earth, temperature readings for a number of nearby stations, election results for voting districts and a number of consecutive elections, GPS tracks for people or animals possibly with additional sensor readings, disease outbreaks or volcano eruptions.

Schabenberger and Gotway (2004) argue that analysis of spatio-temporal data often happens *conditionally*, meaning that either first the spatial aspect is analysed, after which the temporal aspects are analysed, or reversed, but not in a joint, integral modelling approach, where space and time are not separated. As a possible reason they mention the lack of good software, data classes and methods to handle, import, export, display and analyse such data. This R (R Development Core Team 2011) package is a start to fill this gap.

Spatio-temporal data are often relatively abundant in either space, or time, but not in both. Satellite imagery is typically very abundant in space, giving lots of detail in high spatial resolution for large areas, but relatively sparse in time. Analysis of repeated images over time may further be hindered by difference in light conditions, errors in georeferencing resulting in spatial mismatch, and changes in obscured areas due to changed cloud coverage. On the other side, data from fixed sensors give often very detailed signals over time, allowing for elaborate modelling, but relatively little detail in space because a very limited number of sensors is available. The cost of an in situ sensor network typically depends primarily on its spatial density; the choice of the temporal resolution with which the sensors register signals may have little effect on total cost.

Although for example Botts, Percivall, Reed, and Davidson (2007) describe a number of open standards that allow the interaction with sensor data (describing sensor characteristics, requesting observed values, planning sensors, and processing raw sensed data to predefined events), the available statistical or GIS software for this is in an early stage, and scattered. This paper describes an attempt to combine available infrastructure in the R statistical environment to a set of useful classes and methods for manipulating, plotting and analysing spatio-temporal data. A number of case studies from different application areas will illustrate its use.

The paper is structured as follows. Section 2 describes how spatio-temporal data are usually recorded in tables. Section 3 describes a number of useful spatio-temporal layouts. Section 4 introduces classes and methods for data, based on these layouts. Section 5 presents a number of useful graphs for spatio-temporal data, and implementations for these. Section 6 discusses the spatial and temporal footprint, or support, of data, and how time intervals are dealt with in practice. Section 7 presents a number of worked examples, some of which include statistical analysis on the spatio-temporal data. Section 8 points to further material, including vignettes in package **spacetime**[1] on spatio-temporal overlay and aggregation, and on using proxy data sets to PostgreSQL tables that are too large to fit in memory for R. Section 9 finishes with a discussion.

2. How spatio-temporal data are recorded in tables

For reasons of simplicity, spatio-temporal data often come in the form of single tables. If this is the case, they come in one of three forms:

time-wide where different columns reflect different moments in time,

space-wide where different columns reflect different measurement locations or areas, or

[1] This paper is available (in updated form) as vignette from this package, which implements the classes and methods for spatio-temporal data described here.

long formats where each record reflects a single time and space combination.

Alternatively, they may be stored in different, related tables, which is more typical for relational data bases, or in tree structures which is typical for xml files. We will now illustrate the different single-table formats with simple examples.

2.1. Time-wide format

Spatio-temporal data for which each location has data for each time can be provided in two so-called **wide formats**. An example where a single column refers to a single moment or period in time is found in the North Carolina Sudden Infant Death Syndrome (sids) data set, which is in the **time-wide format**:

```
R> library("foreign")
R> read.dbf(system.file("shapes/sids.dbf", package="maptools"))[1:5,c(5,9:14)]

         NAME BIR74 SID74 NWBIR74 BIR79 SID79 NWBIR79
1        Ashe  1091     1      10  1364     0      19
2   Alleghany   487     0      10   542     3      12
3       Surry  3188     5     208  3616     6     260
4   Currituck   508     1     123   830     2     145
5 Northampton  1421     9    1066  1606     3    1197
```

where **columns** refer to a particular **time**: `SID74` contains to the infant death syndrome cases for each county at a particular time period (1974-1978).

2.2. Space-wide format

The Irish wind data (Haslett and Raftery 1989), for which the first six records and 9 of the stations (abbreviated by `RPT`, `VAL`, ...) are shown by

```
R> data("wind", package = "gstat")
R> 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
```

are in **space-wide format**: each *column* refers to another wind measurement **location**, and the rows reflect a single time period; wind was reported as daily average wind speed in knots (1 knot = 0.5418 m/s).

2.3. Long format

Finally, panel data are shown in **long form**, where the full spatio-temporal information is held in a single column, and other columns denote location and time. In the `Produc` data set (Baltagi 2001), a panel of 48 observations from 1970 to 1986 available in the **plm** package (Croissant and Millo 2008), the first five records and nine columns are

```
R> data("Produc", package = "plm")
R> Produc[1:5,1:9]
```

```
    state year     pcap     hwy   water    util       pc   gsp    emp
1 ALABAMA 1970 15032.67 7325.80 1655.68 6051.20 35793.80 28418 1010.5
2 ALABAMA 1971 15501.94 7525.94 1721.02 6254.98 37299.91 29375 1021.9
3 ALABAMA 1972 15972.41 7765.42 1764.75 6442.23 38670.30 31303 1072.3
4 ALABAMA 1973 16406.26 7907.66 1742.41 6756.19 40084.01 33430 1135.5
5 ALABAMA 1974 16762.67 8025.52 1734.85 7002.29 42057.31 33749 1169.8
```

where the first two columns denote space and time (the default assumption for package **plm**), and e.g., `pcap` reflects private capital stock.

None of these examples has strongly *referenced* spatial or temporal information: it is from the data alone not clear that the number `1970` refers to a year, or that `ALABAMA` refers to a state, and where this state is. Section 7 shows for each of these three cases how the data can be converted into classes with strongly referenced space and time information.

3. Space-time layouts

In the following we will use the word spatial *feature* (Herring 2011) to denote a spatial entity. This can be a particular spatial point (location), a line or set of lines, a polygon or set of polygons, or a pixel (grid or raster cell). For a particular feature, one or more measurements are registered at particular moments in time.

Four layouts of space-time data will be discussed next. Two of them reflect lattice layouts, one that is efficient when a particular spatial feature has data values for more than one time point, and one that is most efficient when all spatial feature have data values at each time point. Two others reflect irregular layouts, one of them specializes to trajectories.

3.1. Spatio-temporal full grids

A full space-time grid of observations for spatial features (points, lines, polygons, grid cells)[2] $s_i, i = 1, ..., n$ and observation time $t_j, j = 1, ..., m$ is obtained when the full set of $n \times m$ set of observations z_k is stored, with $k = 1, ..., nm$. We choose to cycle spatial features first, so observation k corresponds to feature s_i, $i = ((k-1)\ \%\ n) + 1$ and with time moment t_j, $j = ((k-1)/n) + 1$, with / integer division and % integer division remainder (modulo). The t_j are assumed to be in time order.

In this data class (top left in figure 1), for each spatial feature, the same temporal sequence of data is sampled. Alternatively one could say that for each moment in time, the same set of spatial entities is sampled. Unsampled combinations of (space, time) are stored in this class, but are assigned a missing value `NA`.

[2] note that neither spatial features nor time points need to follow a regular layout

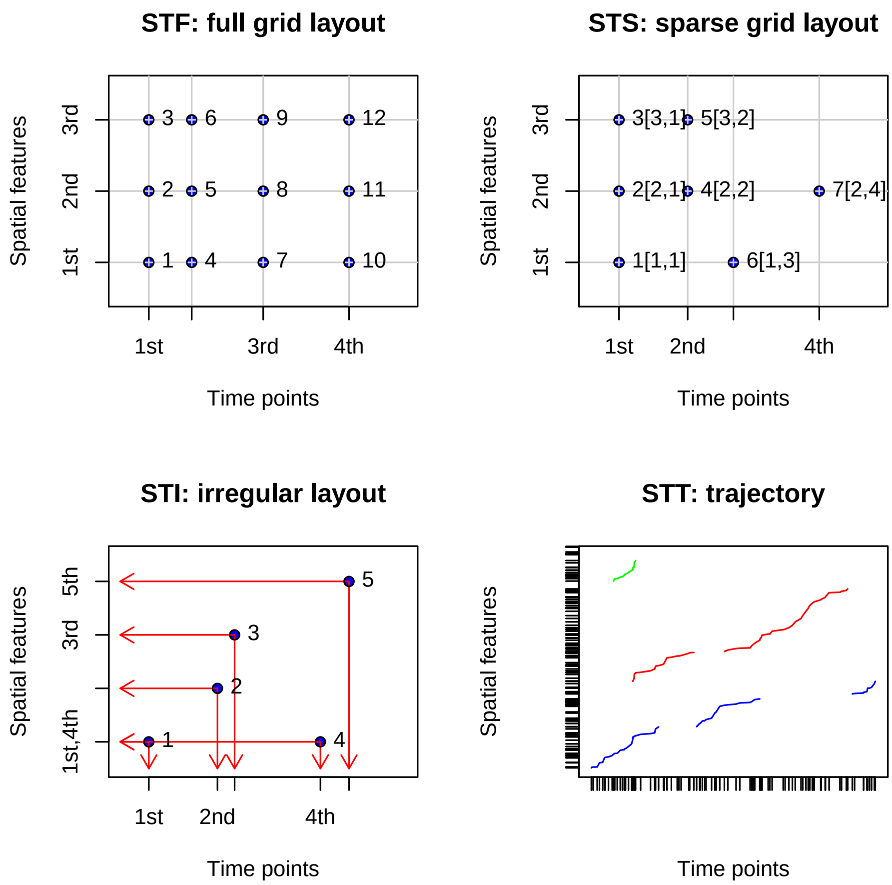

Figure 1: Space-time layouts: (i) the top-left: full grid (STF) layout stores all space-time combinations; (ii) top-right: the sparse grid (STS) layout stores only the non-missing space-time combinations on a lattice; (iii) bottom-left: the irregular (STI) layout: each observation has its spatial feature and time stamp stored, in this example, spatial feature 1 is stored twice – the fact that observations 1 and 4 have the same feature is not registered; (iv) bottom right: simple trajectories (STT), plotted against a common time axis

3.2. Spatio-temporal sparse grids

A sparse grid has the same general layout, with measurements laid out on a space time grid (top right in figure 1), but instead of storing the full grid, only non-missing valued observations z_k are stored. For each k, an index $[i, j]$ is stored that refers which spatial feature i and time point j the value belongs to. Storing data this way may be efficient if full space-time lattices have many missing values, or if a limited set of spatial features each have different time instances (times of crime cases for a set of administrative regions), or if for a set of times the set of spatial features varies (locations of crimes registered per year, or spatially misaligned remote sensing images).

3.3. Spatio-temporal irregular data

Space-time irregular data cover the case where time and space points of measured values have no apparent organisation: for each measured value the spatial feature and time point is stored, as in the long format. This is equivalent to the (maximally) sparse grid where the index for observation k is $[k, k]$, and hence can be dropped. For these objects, $n = m$ equals the number of records. Spatial features and time points need not be unique, but are replicated in case they are not.

3.4. Spatio-temporal trajectories

Trajectories cover the case where sets of (irregular) space-time points are grouped, and form a trajectory. The grouping may be simple (e.g. the trajectories of two persons on a single day), nested (for several objects, a set of trajectories representing different trips) or complex (e.g. with objects that split, merge, or disappear).

4. Classes and methods for spatio-temporal data

The different layouts, or types, of spatio-temporal data discussed in the previous section have been implemented in the **spacetime** R package, along with methods for import, export, coercion, selection, and visualisation.

4.1. Classes

The classes for the different layouts are shown in figure 2. Similar to the classes of package **sp** (Bivand, Pebesma, and Gomez-Rubio 2008), the classes all derive from a base class `ST` which is not meant to represent actual data. The first order derived classes specify particular spatio-temporal geometries (i.e., only the spatial and temporal information), the second order derived classes augment each of these with actual data, in the form of a `data.frame`.

To store temporal information, we chose to use objects of class `xts` in package **xts** (Ryan and Ulrich 2011) for time, because

- it extends the functionality of package **zoo** (Zeileis and Grothendieck 2005),
- it supports several basic types to represent time or date: `Date`, `POSIXct`, `timeDate`, `yearmon`, and `yearqtr`,

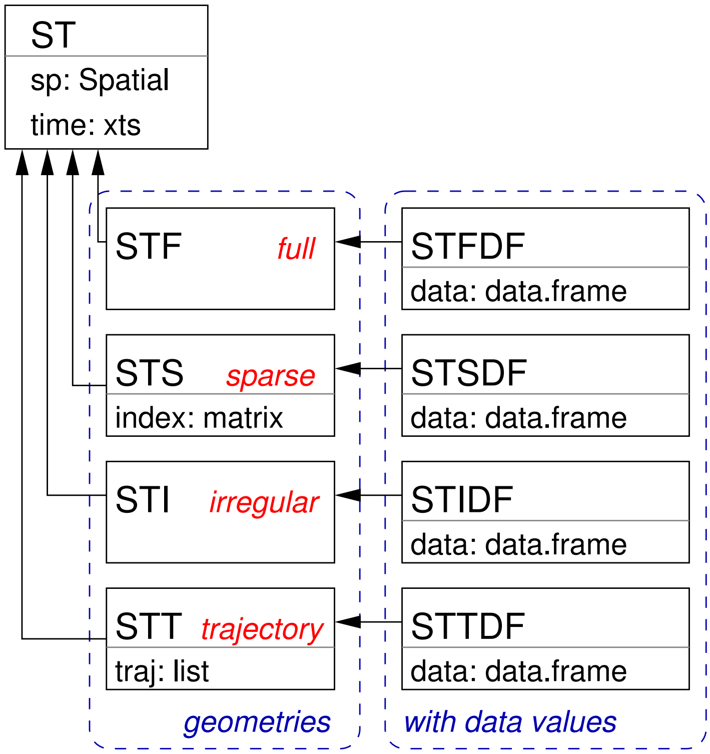

Figure 2: classes for spatio-temporal data in package **spacetime**. Arrows denote inheritance, lower side of boxes list slot name and type

- it has good tools for *aggregation* over time using arbitrary aggregation functions, essentially deriving this from package `zoo` (Zeileis and Grothendieck 2005).
- it has a flexible syntax to select time periods that adheres to ISO 8601[3].

An overview of the different time classes in R is found in Ripley and Hornik (2001). Further advice on which classes to use is found in Grothendieck and Petzoldt (2004).

For spatial interpolation, we used the classes deriving from `Spatial` in package **sp** (Pebesma and Bivand 2005; Bivand *et˜al.* 2008) because

- they are the dominant set of classes in R for dealing with spatial data,
- they are interfaced to key external libraries through packages **rgdal** and **rgeos**, and
- they provide a single interface to dealing with points, lines, polygons and grids.

We do not use `xts` or `Spatial` objects to *store* spatio-temporal data values, but we use `data.frame` to store data values. For purely temporal information the `xts` objects can be used, and for purely spatial information the `sp` objects can be used. These will be recycled appropriately when coercing to a long format `data.frame`.

[3]`http://en.wikipedia.org/wiki/ISO_8601`

method	what it does
`stConstruct`	Creates STFDF or STIDF objects from single or multiple tables
`[[`, `$`, `$<-`	Selects or replaces data values
`[`, `[[`, `$`	Select spatial and/or temporal sections
`as`	coerces to other spatio-temporal objects, `xts`, `Spatial`, `matrix`, or `data.frame`
`stplot`	creates spatio-temporal plots, see section 5
`over`	overlay: retrieves index or data values of one object at the locations/times of another
`aggregate`	aggregates data values over particular spatial, temporal, or spatio-temporal domains

Table 1: methods for spatio-temporal data in package **spacetime**

The spatial features supported by package **sp** are two-dimensional for lines and polygons, but may be higher (three-) dimensional for spatial points, pixels and grids.

4.2. Methods

The main methods for spatio-temporal data implemented in packages **spacetime** are listed in table 1.

4.3. Creation

Construction of spatio-temporal objects essentially needs specification of the spatial, the temporal, and the data values. The documentation of `stConstruct` contains examples of how this can be done from long, space-wide, and time-wide tables, or from shapefiles. A simple toy example for a full grid layout with three spatial points and four time instances is given below. First, the spatial object is created:

```
R> sp = cbind(x = c(0,0,1), y = c(0,1,1))
R> row.names(sp) = paste("point", 1:nrow(sp), sep="")
R> library(sp)
R> sp = SpatialPoints(sp)
```

Then, the time points are defined:

```
R> time = as.POSIXct("2010-08-05", tz = "GMT")+3600*(10:13)
```

next, a data.frame with the data values is created:

```
R> m = c(10,20,30) # means for each of the 3 point locations
R> values = rnorm(length(sp)*length(time), mean = rep(m, 4))
R> IDs = paste("ID",1:length(values), sep = "_")
R> mydata = data.frame(values = signif(values, 3), ID=IDs)
```

and finally, the `STFDF` object is created:

```
R> library(spacetime)
R> stfdf = STFDF(sp, time, mydata)
```

4.4. Overlay and aggregation

Aggregation of data values to a more coarse spatial or temporal form (e.g., to a coarser grid, aggregating points over administrative regions, aggregating daily data to monthly data, or aggregation along an irregular set of space-time points) can be done using the method `aggregate`. To obtain the required aggregation predicate, i.e. the grouping of observations in space-time, the method `over` is implemented for objects deriving from `ST`. Grouping can be done based on spatial, temporal, or spatio-temporal predicates. This effectively provides an spatio-temporal equivalent to what is known in geographic information science as the *spatial overlay*.

4.5. Space and time selection with `[`

The idea behind the `[` method for classes in `sp` was that objects would behave as much as possible similar to a matrix or `data.frame`. For a `data.frame`, a construct like `a[i,j]` selects row(s) `i` and column(s) `j`. For objects deriving from `Spatial`, rows were taken as the spatial features (points, lines, polygons, pixels) and columns as the data variables[4].

For the spatio-temporal data classes described here, `a[i,j,k]` selects spatial features `i`, temporal instances `j`, and data variable(s) `k`. Unless `drop=FALSE` is added to such a call, selecting a single time or single feature results in an object that is no longer spatio-temporal, but either *snapshot* of a particular moment, or *history* at a particular feature (Galton 2004).

Similar to selection on spatial objects in **sp** and time series objects in **xts**, space and time indices can be defined by index or boolean vectors, but also by higher-level expressions such as spatial areas and time periods. For instance, the selection

```
R> air_quality[Germany, "2008::2009", "PM10"]
```

would select the `PM10` measurements for the years 2008-9, lying in `Germany`, when `Germany` is a `Spatial` object (polygons, or a grid) that defines Germany.

4.6. Coercion to long and wide tables

Spatio-temporal data objects can be coerced to the corresponding purely spatial objects. Objects of class `STFDF` will be represented in time-wide form, where only the first (selected) data variable is retained:

```
R> xs1 = as(stfdf, "Spatial")
R> class(xs1)

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

[4]a convention that was partially broken for class `SpatialGridDataFrame`, where `a[i,j,k]` could select the k-th data variable of the spatial grid selection with spatial grid row(s) `i` and column(s) `j`, *unless* the length of `i` equals the number of grid cells.

```
R> xs1

       coordinates X2010.08.05.10.00.00 X2010.08.05.11.00.00
point1      (0, 0)                 11.0                 11.2
point2      (0, 1)                 19.5                 19.9
point3      (1, 1)                 29.5                 28.3
       X2010.08.05.12.00.00 X2010.08.05.13.00.00
point1                 10.4                 10.4
point2                 19.6                 19.8
point3                 29.1                 31.8
```

as time values are difficult to retrieve from these column names, this object gets the proper time values as an attribute:

```
R> attr(xs1, "time")

[1] "2010-08-05 10:00:00 GMT" "2010-08-05 11:00:00 GMT"
[3] "2010-08-05 12:00:00 GMT" "2010-08-05 13:00:00 GMT"
```

Objects of class STSDF or STIDF will be represented in long form, where time is added as additional column:

```
R> x = as(stfdf, "STIDF")
R> xs2 = as(x, "Spatial")
R> class(xs2)

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

R> xs2[1:4,]

  coordinates values   ID                time
1      (0, 0)   11.0 ID_1 2010-08-05 10:00:00
2      (0, 1)   19.5 ID_2 2010-08-05 10:00:00
3      (1, 1)   29.5 ID_3 2010-08-05 10:00:00
4      (0, 0)   11.2 ID_4 2010-08-05 11:00:00
```

5. Graphs of spatio-temporal data

5.1. stplot: panels, space-time plots, animation

The `stplot` method can create a few specialized plot types for the classes in the `spacetime` package. They are:

multi-panel plots In this form, for each time step (selected) a map is plotted in a separate panel, and the strip above the panel indicates what the panel is about. The panels share x- and y-axis, no space needs to be lost by separating white space, and a common legend is used. Three types are implemented for STFDF data:

- x and y axis denote space, an example for gridded data is shown in figure 3. The `stplot` is a wrapper around `spplot` in package `sp`, and inherits most of its options.
- x and y denote time and value; one panel for each spatial feature, colors may indicate different variables (`mode="tp"`)
- x and y denote time and value; one panel for each variable, colors may denote different stations (`mode="ts"`)

space-time plots space-time plots show data in a space-time cross-section, with e.g., space on the x-axis and time on the y-axis. (See also figure 1.)

Hovmöller plots (Hovmöller 1949) are an example of these for full space-time lattices, i.e. objects of class `STFDF`. To obtain such a plot, the arguments `mode` and `scaleX` should be considered; some special care is needed when only the x- or y-axis needs to be plotted instead of the spatial index (1...n); details are found in the `stplot` documentation. An example of a Hovmöller-style plot with station index along the x-axis and time along the y-axis is obtained by

```
R> scales=list(x=list(rot = 45))
R> stplot(w, mode = "xt", scales = scales, xlab = NULL)
```

and shown in figure 4. Note that the y-axis direction is opposite to that of regular Hovmöller plots.

animated plots Animation is another way of displaying change over time; a sequence of `spplots`, one for each time step, is looped over when the parameter `animate` is set to a positive value (indicating the time in seconds to pause between subsequent plots).

Time series plots

Time series plots are a fairly common type of plot in R. Package `xts` has a plot method that allows univariate time series to be plotted. Many (if not most) plot routines in R support time to be along the x- or y-axis. The plot in figure 5 was generated by using package **lattice** (Sarkar 2008), and uses a colour palette from package **RColorBrewer** (Neuwirth 2011):

```
R> library("lattice")
R> library("RColorBrewer")
R> b = brewer.pal(12, "Set3")
R> par.settings = list(superpose.symbol = list(col = b, fill = b),
+          superpose.line = list(col = b),
+          fontsize = list(text=9))
R> stplot(w, mode = "ts",  auto.key=list(space="right"),
+          xlab = "1961", ylab = expression(sqrt(speed)),
+          par.settings = par.settings)
```

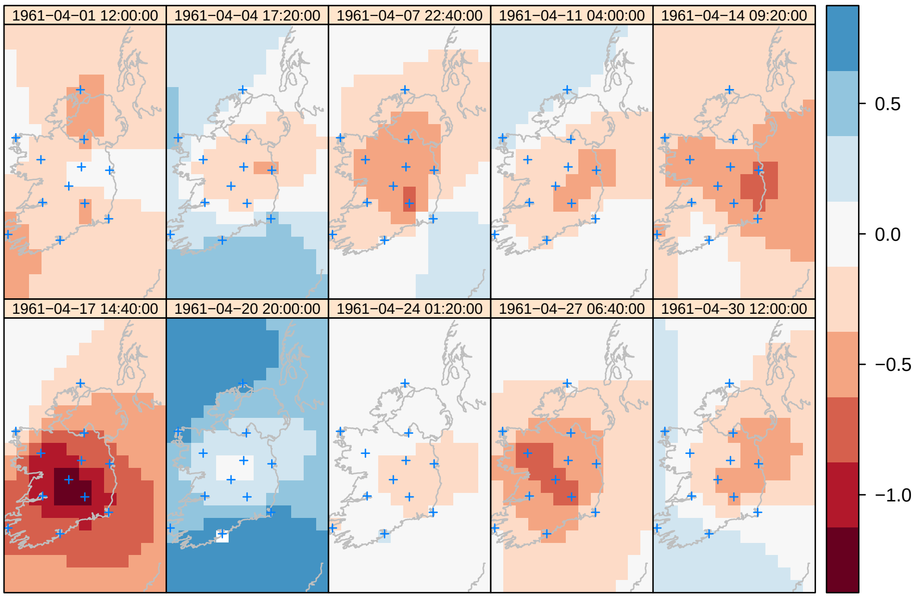

Figure 3: Space-time interpolations of wind (square root transformed, detrended) over Ireland using a separable product covariance model, for 10 time points regularly distributed over the month for which daily data was considered (April, 1961).

6. Spatial footprint or support, time intervals

6.1. Time periods or time instances

Data structures for time series data in R have, explicitly or implicitly, for each record a time stamp, not a time interval. The implicit assumption seems to be (i) the time stamp is a moment, (ii) this indicates either the real moment of measurement / registration, or the start of the interval over which something is aggregated (summed, averaged, maximized). For financial "Open, high, low, close" data, the "Open" and "Close" refer to the values at the moments the stock exchange opens and closes, meaning time instances, whereas "high" and "low" are aggregated values – the minimum and maximum price over the time interval between opening and closing times.

Package `lubridate` (Grolemund and Wickham 2011) allows one to explicitly define and compute with time intervals (e.g., Allen (1983)). It does not provide structures to attach these

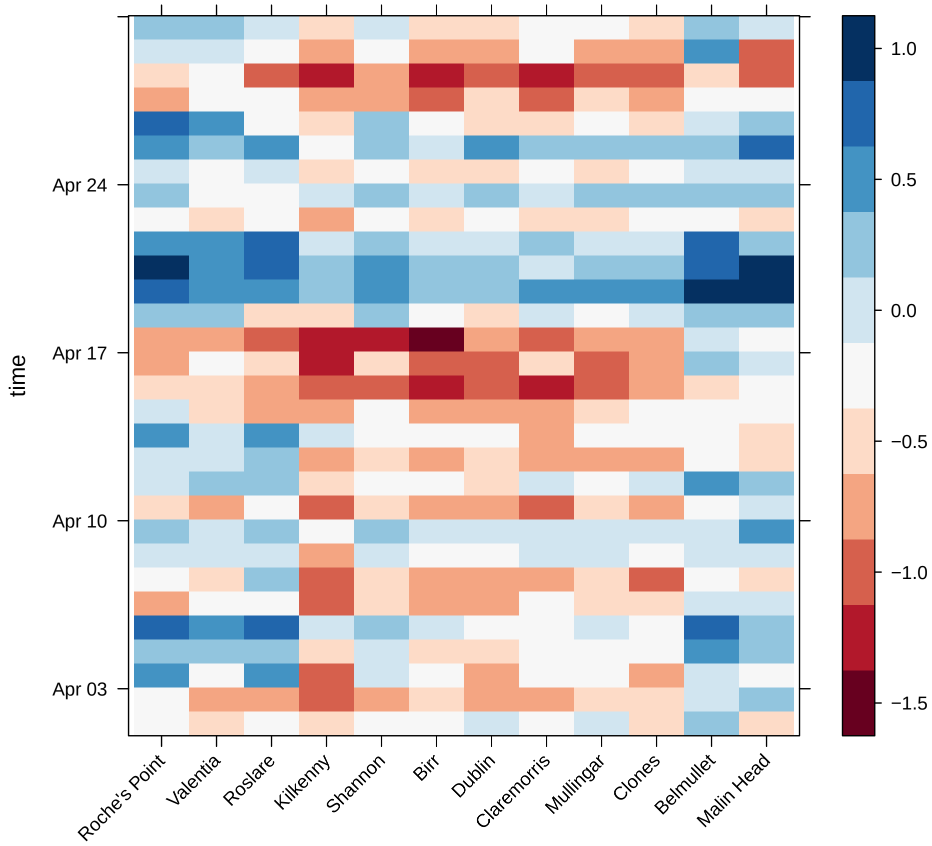

Figure 4: Space-time (Hovmöller) plot of wind station data.

intervals to time series data, or else to use them in `xts` objects.

According to ISO 8601:2004, a time stamp like "2010-05" refers to *the full* month of May, 2010, and so reflects a time period rather than a moment. As a selection criterion, `xts` will include everything inside the following interval:

```
R> .parseISO8601('2010-05')

$first.time
[1] "2010-05-01 CEST"

$last.time
[1] "2010-05-31 23:59:59 CEST"
```

and this syntax lets one define, unambiguously, yearly, monthly, daily, hourly or minute intervals, but not e.g.˜10- or 30-minute intervals. For a particular interval, the full specification is needed:

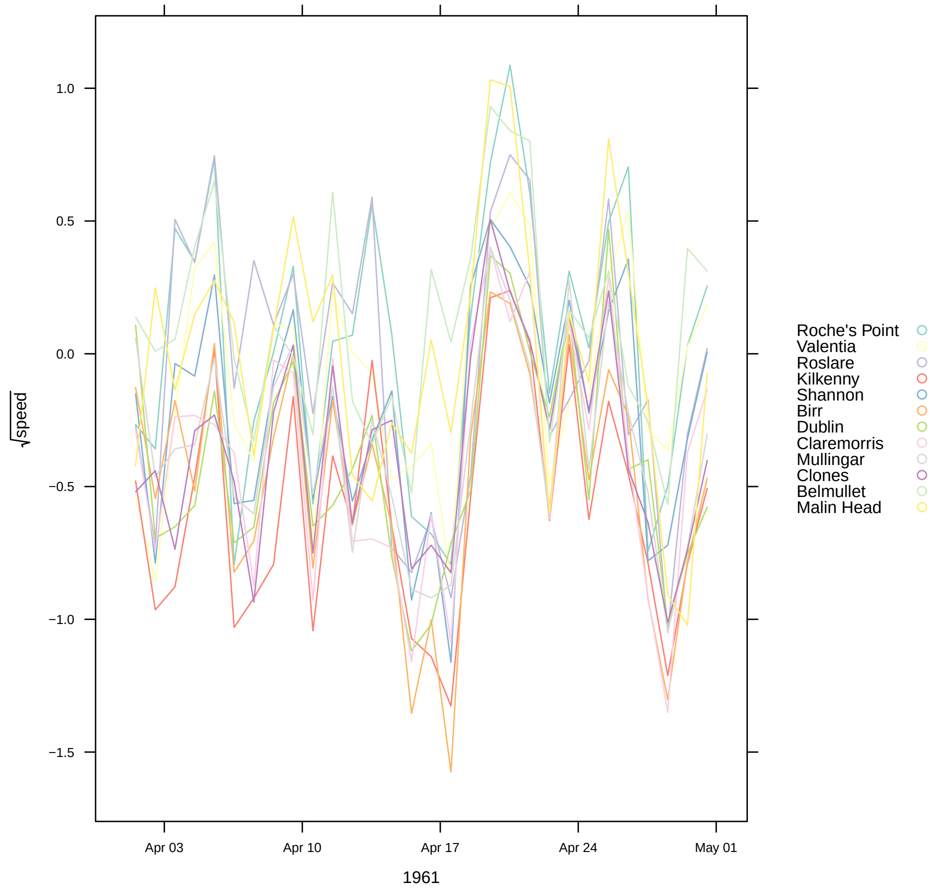

Figure 5: Time series plot of daily wind speed at 12 stations, used for interpolation in figure 3.

```
R> .parseISO8601('2010-05-01T13:30/2010-05-01T13:39')

$first.time
[1] "2010-05-01 13:30:00 CEST"

$last.time
[1] "2010-05-01 13:39:59 CEST"
```

When matching two sequences of time (figure 6) in order to overlay or aggregate, it matters whether each of the sequences reflect instances, one of them reflects time intervals and the other instances, or both reflect time intervals. All of the three cases are accomodated for in package **spacetime**. By default, objects deriving from `STF` or `STS` are considered to reflect interval time, and objects deriving from `STI` or `STT` instance time. As only start times of time intervals are given, the package makes the assumption that the last interval (for which no end time instance is present) has the same length as the one-but-last interval.

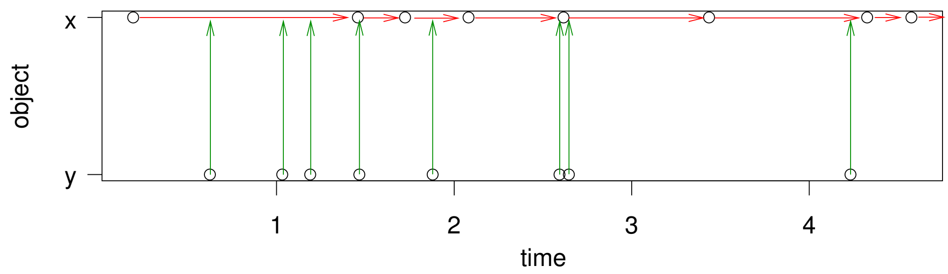

Figure 6: Matching two time sequences, assuming `x` reflects time intervals, and `y` reflects time instances. Note the interval extended beyond the last point of `x`.

6.2. Spatial support

All examples above work with spatial points, i.e., data having a point support. The assumption of data having points support is implicit for `SpatialPoints` features. For polygons, the assumption will be that values reflect aggregates (e.g. sums, or averages) over the polygon. For gridded data, it is ambiguous whether the value at the grid cell centre is meant (e.g. for DEM data) or an aggregate over the grid cell (typical for remote sensing imagery). The `Spatial*` objects of package **sp** have no *explicit* information about the spatial support.

7. Worked examples

This Section shows how existing data in various formats can be converted into ST classes, and how they can be analysed and/or visualised.

7.1. North Carolina SIDS

As an example, the North Carolina Sudden Infant Death Syndrome (sids) data (Symons, Grimson, and Yuan 1983) in package **maptools** (Lewin-Koh, Bivand, contributions~by Edzer J.~Pebesma, Archer, Baddeley, Bibiko, Dray, Forrest, Friendly, Giraudoux, Golicher, Rubio, Hausmann, Hufthammer, Jagger, Luque, MacQueen, Niccolai, Short, Stabler, and Turner 2011) will be used; they are sparse in time (aggregated to 2 periods of unequal length, according to the documentation in package `spdep`), but have polygons in space. First, we will prepare the spatial data:

```
R> library("maptools")
R> fname = system.file("shapes/sids.shp", package="maptools")[1]
R> nc = readShapePoly(fname, proj4string=CRS("+proj=longlat +datum=NAD27"))
```

then, we construct the time sequence:

```
R> time = as.POSIXct(strptime(c("1974-01-01", "1979-01-01"), "%Y-%m-%d"),
+           tz = "GMT")
```

and finally we construct the data values table (in long form):

```
R> data = data.frame(
+          BIR = c(nc$BIR74, nc$BIR79),
+          NWBIR = c(nc$NWBIR74, nc$NWBIR79),
+          SID = c(nc$SID74, nc$SID79))
```

These three components are put together by function `STFDF`:

```
R> nct = STFDF(sp = as(nc, "SpatialPolygons"), time = time, data = data)
```

7.2. Panel data

The panel data discussed in Section 2 are imported as a full spatio-temporal `data.frame` (`STFDF`), and linked to the proper state polygons of maps. We obtain the states polygons by:

```
R> library("maps")
R> states.m = map('state', plot=FALSE, fill=TRUE)
R> IDs <- sapply(strsplit(states.m$names, ":"), function(x) x[1])
R> library("maptools")
R> states = map2SpatialPolygons(states.m, IDs=IDs)
```

we obtain the time points by:

```
R> yrs = 1970:1986
R> time = as.POSIXct(paste(yrs, "-01-01", sep=""), tz = "GMT")
```

We obtain the data table (already in long format) by

```
R> library("plm")
R> data("Produc")
```

When combining all this information, we do not need to reorder states because `states` and `Produc` order states alphabetically. We need to deselect District of Columbia, which is not present in `Produc` table (record 8):

```
R> # deselect District of Columbia, polygon 8, which is not present in Produc:
R> Produc.st = STFDF(states[-8], time, Produc[order(Produc[2], Produc[1]),])

R> library(RColorBrewer)
R> stplot(Produc.st[,,"unemp"], yrs, col.regions = brewer.pal(9, "YlOrRd"),cuts=9)
```

the plot of which is shown in Figure 7.

Time and state were not removed from the data table on construction; printing these data after coercion to `data.frame` can then be used to verify that time and state were matched correctly.

The routines in package **plm** can be used on the data, when back transformed to a `data.frame`, when `index` is used to specify which variables represent space and time (the first two columns from the `data.frame` no longer contain state and year). For instance, to fit a panel linear model for gross state products (gsp) to private capital stock (pcap), public capital (pc), labor input (emp) and unemployment rate (unemp), we get

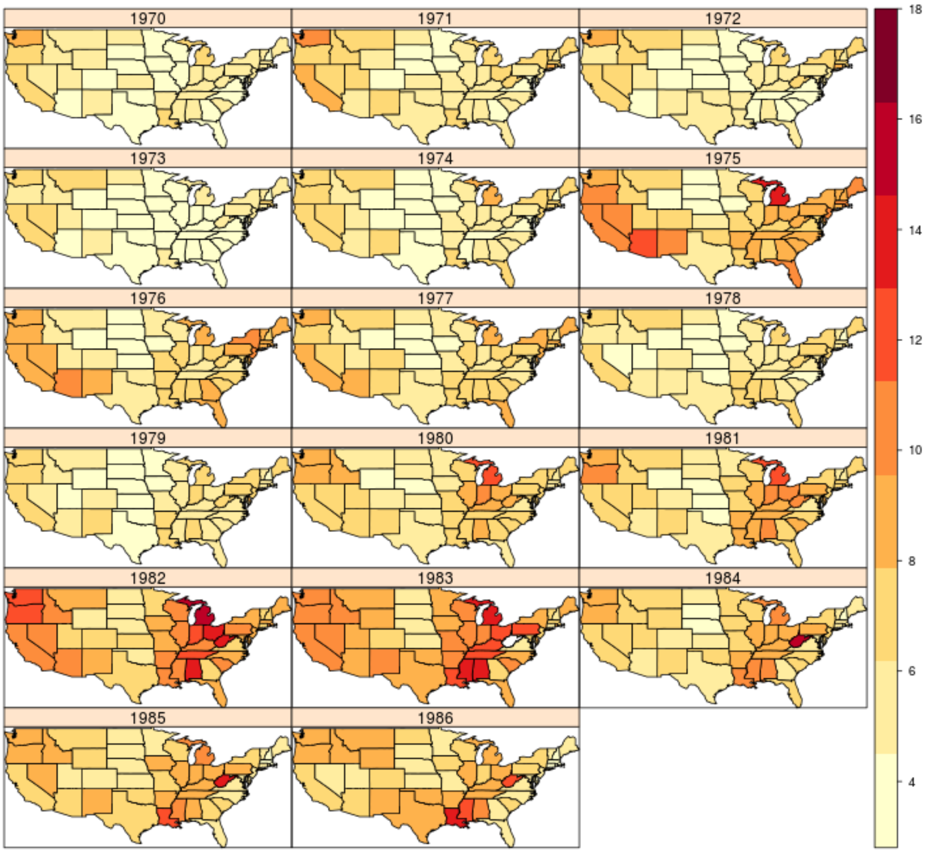

Figure 7: Unemployment rate per state, over the years 1970-1986

```
R> zz <- plm(log(gsp) ~ log(pcap) + log(pc) + log(emp) + unemp,
+           data = as.data.frame(Produc.st), index = c("state","year"))
```

(the output of `summary(zz)` is left out for breveity).

7.3. Interpolating Irish wind

This worked example is a modified version of the analysis presented in `demo(wind)` of package **gstat** (Pebesma 2004). This demo is rather lengthy and reproduces much of the original analysis in Haslett and Raftery (1989). Here, we will reduce the intermediate plots and focus on the use of spatio-temporal classes.

First, we will load the wind data from package `gstat`. It has two tables, station locations in a `data.frame`, called `wind.loc`, and daily wind speed in `data.frame` `wind`. We now convert character representation (such as `51d56'N`) to proper numerical coordinates, and convert the station locations to a `SpatialPointsDataFrame` object. A plot of these data is shown in figure 8.

```
R> library("gstat")
R> data("wind")
R> wind.loc$y = as.numeric(char2dms(as.character(wind.loc[["Latitude"]])))
R> wind.loc$x = as.numeric(char2dms(as.character(wind.loc[["Longitude"]])))
R> coordinates(wind.loc) = ~x+y
R> proj4string(wind.loc) = "+proj=longlat +datum=WGS84"
```

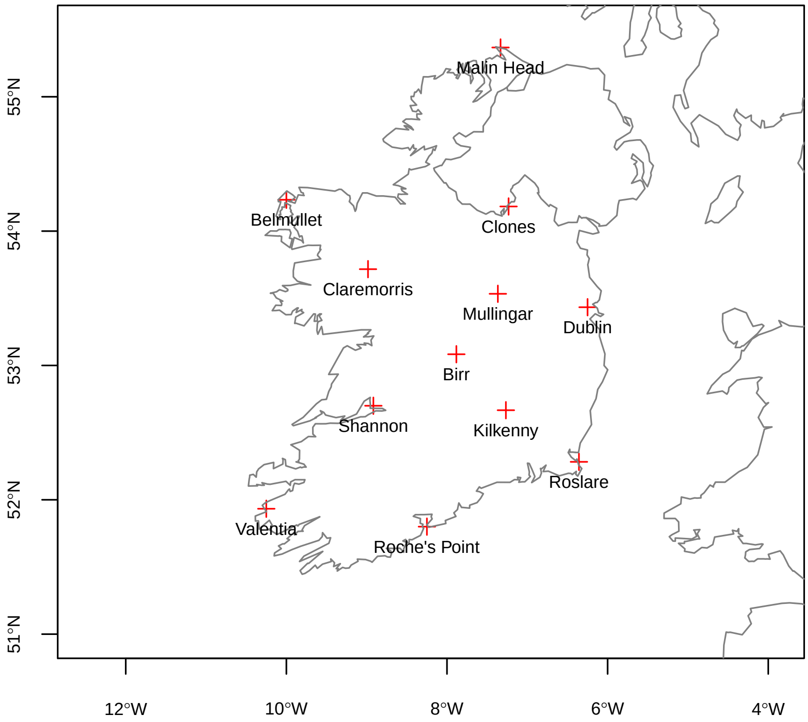

Figure 8: Station locations for Irish wind data.

The first thing to do with the wind speed values is to reshape these data. Unlike the North Carolina SIDS data of Section 7.1, for we have few spatial and many time points, and so the data in `data.frame wind` come in space-wide form with stations time series in columns:

```
R> wind[1:3,]

  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
```

```
    CLO   BEL   MAL
1 12.58 18.50 15.04
2  9.67 17.54 13.83
3  7.67 12.75 12.71
```

We will recode the time columns to an appropriate time data structure,

```
R> wind$time = ISOdate(wind$year+1900, wind$month, wind$day)
R> wind$jday = as.numeric(format(wind$time, '%j'))
```

and then subtract a smooth time trend of daily means (not exactly equal, but similar to the trend removal in the original paper):

```
R> stations = 4:15
R> windsqrt = sqrt(0.5148 * as.matrix(wind[stations])) # knots -> m/s
R> Jday = 1:366
R> windsqrt = windsqrt - mean(windsqrt)
R> daymeans = sapply(split(windsqrt, wind$jday), mean)
R> meanwind = lowess(daymeans ~ Jday, f = 0.1)$y[wind$jday]
R> velocities = apply(windsqrt, 2, function(x) { x - meanwind })
```

Next, we will match the wind data to its location, by connecting station names to location coordinates, and create a spatial points object:

```
R> wind.loc = wind.loc[match(names(wind[4:15]), wind.loc$Code),]
R> pts = coordinates(wind.loc[match(names(wind[4:15]), wind.loc$Code),])
R> rownames(pts) = wind.loc$Station
R> pts = SpatialPoints(pts, CRS("+proj=longlat +datum=WGS84"))
```

then, we project the longitude/latitude coordinates and country boundary to UTM zone 29, using `spTransform` in package **rgdal** (Keitt, Bivand, Pebesma, and Rowlingson 2011) for coordinate transformation:

```
R> library("rgdal")
R> utm29 = CRS("+proj=utm +zone=29 +datum=WGS84")
R> pts = spTransform(pts, utm29)
```

And now we can construct the spatio-temporal object from the from space-wide table with velocities:

```
R> w = stConstruct(velocities, space = list(values = 1:ncol(velocities)),
+          time = wind$time, SpatialObj = pts)
R> class(w)

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

For plotting purposes, we can obtain country boundaries from package **maps**:

```
R> library("maptools")
R> m = map2SpatialLines(
+          map("worldHires", xlim = c(-11,-5.4), ylim = c(51,55.5), plot=F))
R> proj4string(m) = "+proj=longlat +datum=WGS84"
R> m = spTransform(m, utm29)
```

For interpolation, we can define a grid over the area:

```
R> grd = SpatialPixels(SpatialPoints(makegrid(m, n = 300)),
+          proj4string = proj4string(m))
```

Next, we (arbitrarily) restrict observations to those of April 1961:

```
R> w = w[, "1961-04"]
```

and choose 10 time points from that period to form the spatio-temporal prediction grid:

```
R> n = 10
R> tgrd = xts(1:n, seq(min(index(w)), max(index(w)), length=n))
R> pred.grd = STF(grd, tgrd)
```

We will interpolate with a separable exponential covariance model, with ranges 750 km and 1.5 day:

```
R> v = list(space = vgm(0.6, "Exp", 750000), time = vgm(1, "Exp", 1.5 * 3600 * 24))
R> pred = krigeST(values ~ 1, w, pred.grd, v)
R> wind.ST = STFDF(grd, tgrd, data.frame(sqrt_speed = pred))
```

then creates the `STFDF` object with interpolated values, the results of which are shown in figure 3, created by

```
R> layout = list(list("sp.lines", m, col='grey'),
+          list("sp.points", pts, first=F, cex=.5))
R> stplot(wind.ST, col.regions=brewer.pal(11, "RdBu")[-c(10,11)],
+          at=seq(-1.375,1,by=.25),
+          par.strip.text = list(cex=.7), sp.layout = layout)
```

7.4. Calculation of EOFs

Empirical orthogonal functions from `STFDF` objects can be computed in spatial form (default):

```
R> eof.sp = EOF(wind.ST)
```

or in temporal form by:

```
R> eof.xts = EOF(wind.ST, "temporal")
```

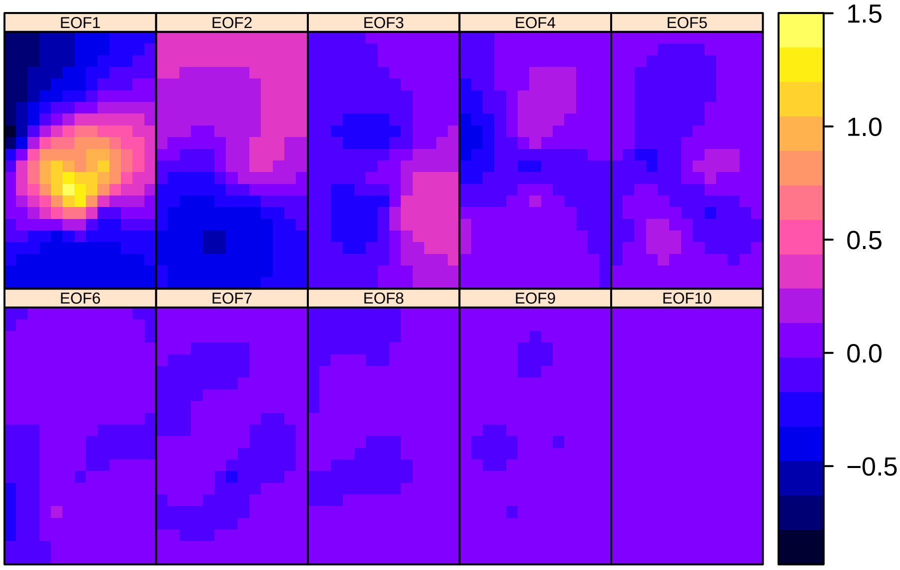

Figure 9: EOFs of space-time interpolations of wind over Ireland (for spatial reference, see figure 3), for the 10 time points at which daily data was chosen above (April, 1961).

the resulting object is of the appropriate `Spatial` subclass (`SpatialGridDataFrame`, `SpatialPolygonsDataFr` etc.) in the spatial form, or of class `xts` in the temporal form. Figure 9 shows the 10 spatial EOFs obtained from the interpolated wind data of figure 3.

7.5. Conversion from and to trip

Objects of class `trip` in package **trip** (Sumner 2010), meant to represent trajectories, extend objects of class `SpatialPointsDataFrame` by indicating in which data value columns time and trip ID are, in slot `TOR.columns`. To not lose this information (in particular, which column contains the IDs), we will extend class `STIDF` to retain this info.

The following example uses data from package **diveMove** (Luque 2007). It assumes that time in a trip object is ordered, as **xts** will order it otherwise.

We first prepare the trip object:

```
R> library("diveMove")
R> library("trip")
R> locs = readLocs(gzfile(system.file(file.path("data", "sealLocs.csv.gz"),
+           package="diveMove")), idCol=1, dateCol=2,
+           dtformat="%Y-%m-%d %H:%M:%S", classCol=3,
+           lonCol=4, latCol=5, sep=";")
R> ringy = subset(locs, id == "ringy" & !is.na(lon) & !is.na(lat))
R> coordinates(ringy) = ringy[c("lon", "lat")]
R> tr = trip(ringy, c("time", "id"))
```

Next, we convert it into an STTDF object, and plot it:

```
R> setAs("trip", "STTDF",
+         function(from) {
+                 from$burst = from[[from@TOR.columns[2]]]
+                 time = from[[from@TOR.columns[1]]]
+                 #from = from[order(time),]
+         STIbox = STI(SpatialPoints(t(bbox(from))), range(time))
+                 STT = new("STT", STIbox, traj = list(STI(geometry(from), time)))
+                 new("STTDF", STT, data = from@data)
+         }
+ )
R> x = as(tr, "STTDF")
R> m = map2SpatialLines(map("world",
+         xlim = c(-100,-50), ylim = c(40,77), plot=F))
R> proj4string(m) = "+proj=longlat +datum=WGS84"
R> plot(m, axes=TRUE, cex.axis =.7)
R> plot(x, add=TRUE, col = "red")
```

the resulting plot is shown in figure 10.

7.6. Trajectory data: `ltraj` in package adehabitatLT

Trajectory objects of class `ltraj` in package **adehabitatLT** (Calenge, Dray, and Royer-Carenzi 2008) are lists of bursts, sets of sequential, connected space-time points at which an object is registered. An example `ltraj` data set is obtained by[5]:

```
R> library("adehabitatLT")
R> data("puechabonsp")
R> locs = puechabonsp$relocs
R> xy = coordinates(locs)
R> da = as.character(locs$Date)
R> da = as.POSIXct(strptime(as.character(locs$Date),"%y%m%d"), tz = "GMT")
R> ltr = as.ltraj(xy, da, id = locs$Name)
R> foo = function(dt) dt > 100*3600*24
R> l2 = cutltraj(ltr, "foo(dt)", nextr = TRUE)
```

and these data, converted to STTDF can be plotted, as panels by `time` and `id` by

```
R> sttdf = as(l2, "STTDF")
R> stplot(sttdf, by="time*id")
```

which is shown in figure 11.

7.7. Country shapes in cshapes

The **cshapes** (Weidmann, Kuse, and Gleditsch 2010) package contains a GIS dataset of country boundaries (1946-2008), and includes functions for data extraction and the computation of

[5] taken from **adehabitatLT**, `demo(mangltraj)`

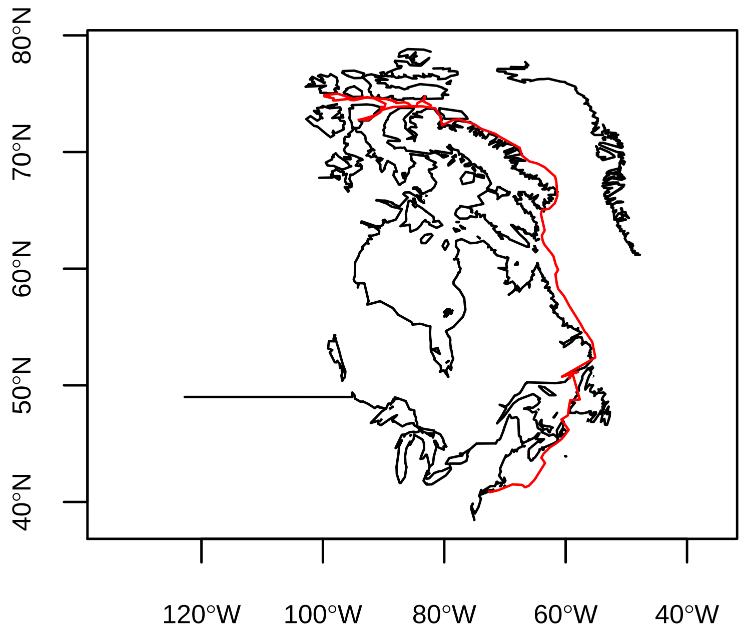

Figure 10: Single trajectory of a seal

distance matrices. The data set consist of a `SpatialPolygonsDataFrame`, with the following data variables:

```
R> library("cshapes")
R> cs = cshp()
R> names(cs)

 [1] "CNTRY_NAME" "AREA"       "CAPNAME"    "CAPLONG"    "CAPLAT"
 [6] "FEATUREID"  "COWCODE"    "COWSYEAR"   "COWSMONTH"  "COWSDAY"
[11] "COWEYEAR"   "COWEMONTH"  "COWEDAY"    "GWCODE"     "GWSYEAR"
[16] "GWSMONTH"   "GWSDAY"     "GWEYEAR"    "GWEMONTH"   "GWEDAY"
[21] "ISONAME"    "ISO1NUM"    "ISO1AL2"    "ISO1AL3"
```

where two data bases are used, "COW" (correlates of war project[6]) and "GW" Gleditsch and Ward (1999). The variables COWSMONTH and COWEMONTH denote the start month and end month, respectively, according to the COW data base.

[6] Correlates of War Project. 2008. State System Membership List, v2008.1. Online, `http://correlatesofwar.org/`

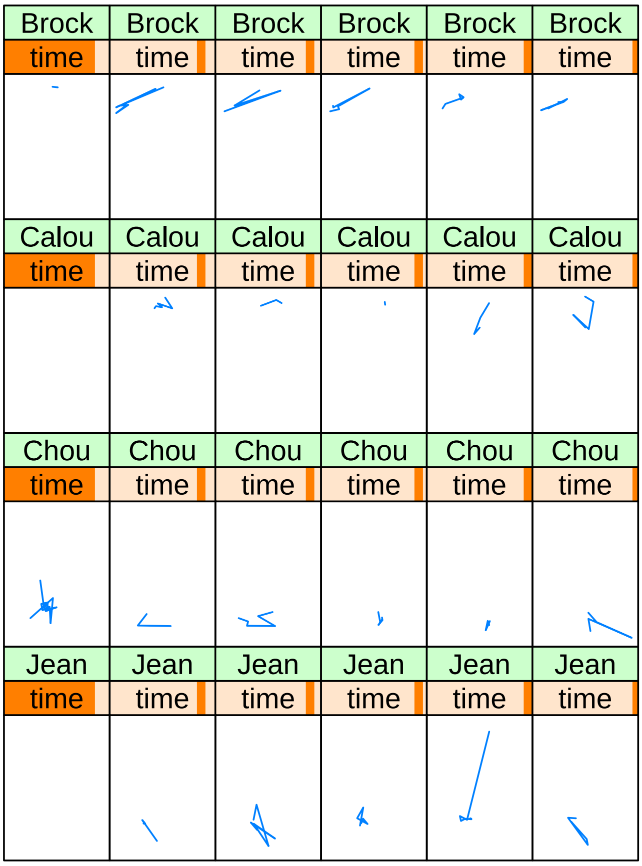

Figure 11: Trajectories, split by id (rows) and by time (columns).

To select the country boundaries corresponding to a particular date and classification system, one can use

```
R> cshp.2002 <- cshp(date=as.Date("2002-6-30"), useGW=TRUE)
```

In the following fragment, we create the time index:

```
R> t = as.POSIXct(strptime(paste(cs$COWSYEAR,
+          cs$COWSMONTH,cs$COWSDAY, sep="-"), "%Y-%m-%d"), tz = "GMT")
```

and the spatio-temporal object:

```
R> st = STIDF(geometry(cs), t, as.data.frame(cs))
```

A possible query would be which countries are found at 7°East and 52°North,

```
R> pt = SpatialPoints(cbind(7, 52), CRS(proj4string(cs)))
R> as.data.frame(st[pt,,1:5])
```

```
        V1       V2 sp.ID       time timedata
1  9.41437 50.57623   188 1955-05-05      188
2 10.38084 51.09070   187 1990-10-03      187
                CNTRY_NAME     AREA CAPNAME CAPLONG   CAPLAT
1 Germany Federal Republic 247366.4    Bonn     7.1 50.73333
2                  Germany 356448.2  Berlin    13.4 52.51667
```

which turns out to be Western Germany and Germany, before and after the merge. No data before 1955-5-5 is available.

8. Further material

8.1. Statistics for spatio-temporal data

The data sets that are provided by Cressie and Wikle (2011), as well as a number of plots presented there, are obtained by

```
R> library(spacetime)
R> demo(CressieWikle)
```

This demo script downloads the data from the book web site, and reproduces a number of graphs from them, shown in the book. It should be noted that the the book examples only deal with `STFDF` objects.

Section 7.3 contains an example of a spatial interpolation with a spatio-temporal separable or or product-sum covariance model. The functions for this are found in package **gstat**, and more information is found through

```
R> library(gstat)
R> vignette("st")
```

8.2. Data base connections

An example where (potentially large) data sets are proxied through R objects is given in a vignette in the **spacetime** package, obtained by

```
R> library(spacetime)
R> vignette("stpg")
```

A proxy object is an object that contains no data, but only references to tables in a data base. Selections on this object are translated into SQL statements that return the actually selected data. This way, the complete data set does not have to be loaded in memory (R), but can be analyzed part by part. Selection in the data base uses indexes on the spatial and temporal references.

8.3. Overlay and aggregation

Examples of overlay and aggregation methods for spatio-temporal data are further detailed in a separate vignette, obtained by

```
R> library(spacetime)
R> vignette("sto")
```

It illustrates the methods with daily air quality data taken from the European air quality data base, for 1998-2009. Aggregations are temporal, and spatial, and both.

9. Discussion

Handling and analyzing spatio-temporal data is often complicated by the size and complexity of these data. Also, data may come in many different forms, may be time-rich, space-rich, as sets of space-time points or as trajectories.

Building on existing infrastructure for spatial and temporal data, we have successfully implemented a coherent set of classes for spatio-temporal data that covers regular space-time layouts, partially regular (sparse) space-time layouts, irregular space-time layouts and trajectory data. The set is flexible in the sense that several representations of space (points, lines, polygons, grid) and time (POSIXt, Date, timeDate, yearmon, yearqtr) can be combined.

We have given examples for constructing objects of these classes from various data sources, coercing them from one to another, exporting them to spatial or temporal representations, as well as visualising them in various forms. We have also shown how one can go from one form into another by ways of prediction based on a statistical model, using an example on spatio-temporal geostatistical interpolation. In addition to spatio-temporally varying information, objects of the classes can contain data values that are purely spatial or purely temporal. Selection can be done based on spatial features, time (intervals), or data variables, and follows a logic similar to that for selection on data tables (`data.frames`).

Using existing infrastructure had the consequence that data that refer to time *intervals* are stored with a (start) time instance only. This may seem incomplete, but seems to reflect current practice in time series analysis. It limits the covered cases to those with non-overlapping time intervals.

Challenges that remain include

- the representation of spatio-temporal polygons in a consistent way, i.e. such that each point in space-time refers to one and only one space-time feature
- dealing with complex developments, such as merging, splitting, and death and birth of objects (further examples are found in Galton (2004))
- explicitly registring the support, or footprint of spatio-temporal data
- annotating objects such that incorrect operations (such as the interpolation of a point process, or the weighted density estimates on a geostatitical process) can lead to warning or error messages
- making handling of massive data sets easier, and implementing efficient spatio-temporal indexes for them
- integrating package **spacetime** with other packages dealing with specific spatio-temporal classes such as **raster** and **surveillance**.

The classes and methods presented in this paper are a first attempt to cover a number of useful cases for spatio-temporal data. In a set of case studies it is demonstrated how they can be used, and can be useful. As software development is often opportunistic, we admittedly picked a lot of low hanging fruits, and a number of large challenges remain. We hope that these first steps will help discovering and identifying these more complex use cases.

Acknowledgements

Michael Sumner provided helpful comments on the trip example. Members from the spatio-temporal modelling lab of the Institute for Geoinformatics of the University of Münster (Ben Gräler, Katharina Henneböhl, Daniel Nüst), and Sören Gebbert contributed in several useful discussions. Participants to the workshop *Handling and analyzing spatio-temporal data in R*, held in Münster on Mar 21-22, 2011, are gratefully acknowledged.

References

Allen JF (1983). "Maintaining Knowledge about Temporal Intervals." *Commun. ACM*, **26**, 832–843. ISSN 0001-0782. doi:http://doi.acm.org/10.1145/182.358434. URL http://doi.acm.org/10.1145/182.358434.

Baltagi B (2001). *Econometric Analaysis of Panel Data, 3rd edition*. John Wiley & Sons, New York. URL http://www.wiley.com/legacy/wileychi/baltagi/.

Bivand RS, Pebesma EJ, Gomez-Rubio V (2008). *Applied Spatial Data Analysis with R*. Springer-Verlag, New York. URL http://www.asdar-book.org/.

Botts M, Percivall G, Reed C, Davidson J (2007). "OGC Sensor Web Enablement: Overview And High Level Architecture." *Technical report*, Open Geospatial Consortium. URL http://portal.opengeospatial.org/files/?artifact_id=25562.

Calenge C, Dray S, Royer-Carenzi M (2008). "The Concept of Animals' Trajectories from a Data Analysis Perspective." *Ecological informatics*, **4**, 34–41.

Cressie N, Wikle C (2011). *Statistics for Spatio-temporal Data*. John Wiley & Sons, New York.

Croissant Y, Millo G (2008). "Panel Data Econometrics in R: The plm Package." *Journal of Statistical Software*, **27**. URL http://www.jstatsoft.org/v27/i02/.

Galton A (2004). "Fields and Objects in Space, Time and Space-time." *Spatial cognition and computation*, **4**.

Gleditsch KS, Ward MD (1999). "Interstate System Membership: A Revised List of the Independent States since 1816." *International Interactions*, **25**, 393–413. URL http://privatewww.essex.ac.uk/~ksg/statelist.html.

Grolemund G, Wickham H (2011). "Dates and Times Made Easy with lubridate." *Journal of Statistical Software*, **40**(3), 1–25. URL http://www.jstatsoft.org/v40/i03/.

Grothendieck G, Petzoldt T (2004). "R Help Desk: Date and Time Classes in R." *R News*, **4**, 29–32. URL http://cran.r-project.org/doc/Rnews/Rnews_2004-1.pdf.

Haslett J, Raftery AE (1989). "Space-time Modelling with Long-memory Dependence: Assessing Ireland's Wind Power Resource (with Discussion)." *Applied Statistics*, **38**, 1–50.

Herring J (2011). "OpenGIS Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture." *OGC Document 06-103r4*. Accessed Mar 3, 2012, URL http://www.opengeospatial.org/standards/sfa.

Hovmöller E (1949). "The Trough-and-Ridge Diagram." *Tellus*, **1 (2)**, 62–66.

Keitt TH, Bivand R, Pebesma E, Rowlingson B (2011). *rgdal: Bindings for the Geospatial Data Abstraction Library*. R package version 0.7-1, URL http://CRAN.R-project.org/package=rgdal.

Lewin-Koh NJ, Bivand R, contributions~by Edzer J~Pebesma, Archer E, Baddeley A, Bibiko HJ, Dray S, Forrest D, Friendly M, Giraudoux P, Golicher D, Rubio VG, Hausmann P, Hufthammer KO, Jagger T, Luque SP, MacQueen D, Niccolai A, Short T, Stabler B, Turner R (2011). *maptools: Tools for Reading and Handling Spatial Objects*. R package version 0.8-10, URL http://CRAN.R-project.org/package=maptools.

Luque SP (2007). "Diving Behaviour Analysis in R." *R News*, **7**(3), 8–14. ISSN -2022. Contributions from: John P.Y. Arnould, Laurent Dubroca, and Andy Liaw, URL http://cran.r-project.org/doc/Rnews/.

Neuwirth E (2011). *RColorBrewer: ColorBrewer palettes*. R package version 1.0-5, URL http://CRAN.R-project.org/package=RColorBrewer.

Pebesma EJ (2004). "Multivariable Geostatistics in S: the gstat Package." *Computers & Geosciences*, **30**(7), 683–691.

Pebesma EJ, Bivand RS (2005). "Classes and Methods for Spatial Data in R." *R News*, **5**(2), 9–13. URL http://cran.r-project.org/doc/Rnews/.

R Development Core Team (2011). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

Ripley B, Hornik K (2001). "Date-time Classes." *R News*, **1**, 8–11.

Ryan JA, Ulrich JM (2011). *xts: eXtensible Time Series*. R package version 0.8-2, URL http://CRAN.R-project.org/package=xts.

Sarkar D (2008). *Lattice: Multivariate Data Visualization with R*. Springer-Verlag, New York. ISBN 978-0-387-75968-5, URL http://lmdvr.r-forge.r-project.org.

Schabenberger O, Gotway C (2004). *Statistical Methods for Spatial Data Analysis*. Chapman and Hall, Boca Raton.

Sumner M (2010). "The Tag Location Problem." *Technical report*, Institute of Marine and Antarctic Studies University of Tasmania. Unpublished PhD thesis.

Symons MJ, Grimson RC, Yuan YC (1983). "Clustering of rare events." *Biometrics*, **39 (1)**, 193–205.

Weidmann NB, Kuse D, Gleditsch KS (2010). "The Geography of the International System: The CShapes Dataset." *International Interactions*, **36 (1)**.

Zeileis A, Grothendieck G (2005). "zoo: S3 Infrastructure for Regular and Irregular Time Series." *Journal of Statistical Software*, **14**(6), 1–27. URL http://www.jstatsoft.org/v14/i06/.

Affiliation:

Edzer Pebesma
Institute for Geoinformatics, University of Münster
Weseler Strasse 253, Münster, Germany
E-mail: edzer.pebesma@uni-muenster.de
URL: http://ifgi.uni-muenster.de/

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume˜VV, Issue˜II *Submitted:* yyyy-mm-dd
MMMMMM YYYY *Accepted:* yyyy-mm-dd