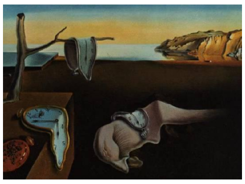


Journal of Statistical Software

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

Classes and Methods for Spatio-Temporal Data in R

ifgi
Institute for Geoinformatics
University of Münster

Edzer Pebesma
Institute for Geoinformatics
University of Münster

Abstract

This document describes classes and methods designed to deal with spatio-temporal data in R implemented in the R package **spacetime**. It builds upon the classes and methods for spatial data are 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, or 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. It is discussed when representing time intervals, i.e., storing for elementary observations their start and end time as opposed to storing only start time, is needed in practice for elementary observations. 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.

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).

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`,
- 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 ISO 8601[1].

We do not use `xts` objects to *store* spatio-temporal attribute information, as it is restricted to `matrix` objects, and hence can only store a single type, and not combine e.g., numeric and factor variables. Instead, as in the classes of **sp** (Pebesma and Bivand 2005; Bivand, Pebesma, and Gomez-Rubio 2008), we use `data.frame` to store measured values. For information that is purely temporal, the `xts` objects can be used, and will be recycled appropriately when coercing to a long format `data.frame`.

[1] http://en.wikipedia.org/wiki/ISO_8601

The organisation of this paper is as follows. We will discuss how much spatio-temporal information is organised in Section 2. Section 3 explains the three major space-time layouts. The spatio-temporal classes for each of these layouts are presented in Sections 4, 5 and 6. Section 7 provides further methods for handling them. Section 8 discusses plot methods. Section 9 discusses spatial and temporal footprint, or support, and Section 10 provides a wide range of worked examples. Section 11 concludes with a discussion.

This paper is available (in updated form) as vignette from the package **spacetime**. Other vignettes in the package deal more extensively with spatio-temporal overlay and aggregation, and with an approach to proxy data sets in a PostgreSQL table that are too large to fit in memory with the objects in package **spacetime**.

2. Space-time data in wide and long formats

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).

The Irish wind data (Haslett and Raftery 1989), for which the first six records are

```
R> data("wind", package = "gstat")
R> wind[1:6,]

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

```
4  5.88  5.46 10.88
5 10.34 12.92 11.83
6  7.50  8.12 13.17
```

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).

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 (Y. and Millo 2008), the first five records are

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

```
    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
  unemp
1   4.7
2   5.2
3   4.7
4   3.9
5   5.5
```

where the first two columns denote space and time (a default assumption in 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 whether the number `1970` refers to a year, or `ALABAMA` to a state, and where this is. Section 10 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 *spatial location* to denote a particular point, (set of) line(s), (set of) polygon(s), or pixel, for which one or more measurements are registered at particular moments in time.

Three layouts of space-time data have been implemented, along with convenience methods and coercion methods to get from one to the other. These will be introduced next.

3.1. Full space-time grid

A full space-time grid[2] of observations for spatial location (points, lines, polygons, grid cells)

[2] note that neither locations nor time points need to be laid out in a regular sequence

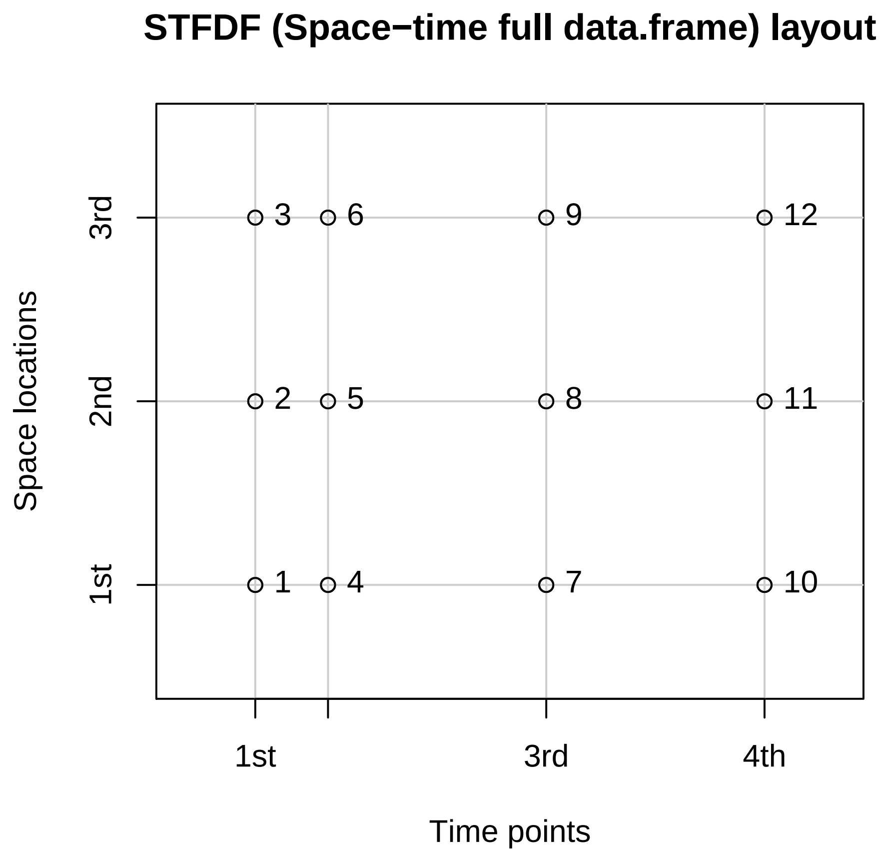


Figure 1: Space-time layout of `STFDF` (`STF`: `ST`-Full) objects: all space-time combinations are stored; numbers refer to the ordering of rows in the `data.frame` with measured values: time is kept ordered, space cycles first.

$s_i, i = 1, ..., n$ 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 locations first, so observation k corresponds to location 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 (figure 1), for each location, 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`.

3.2. Sparse space-time grid

A sparse grid has the same general layout, with measurements laid out on a space time grid (figure 2), but instead of storing the full grid, only non-missing valued observations z_k are

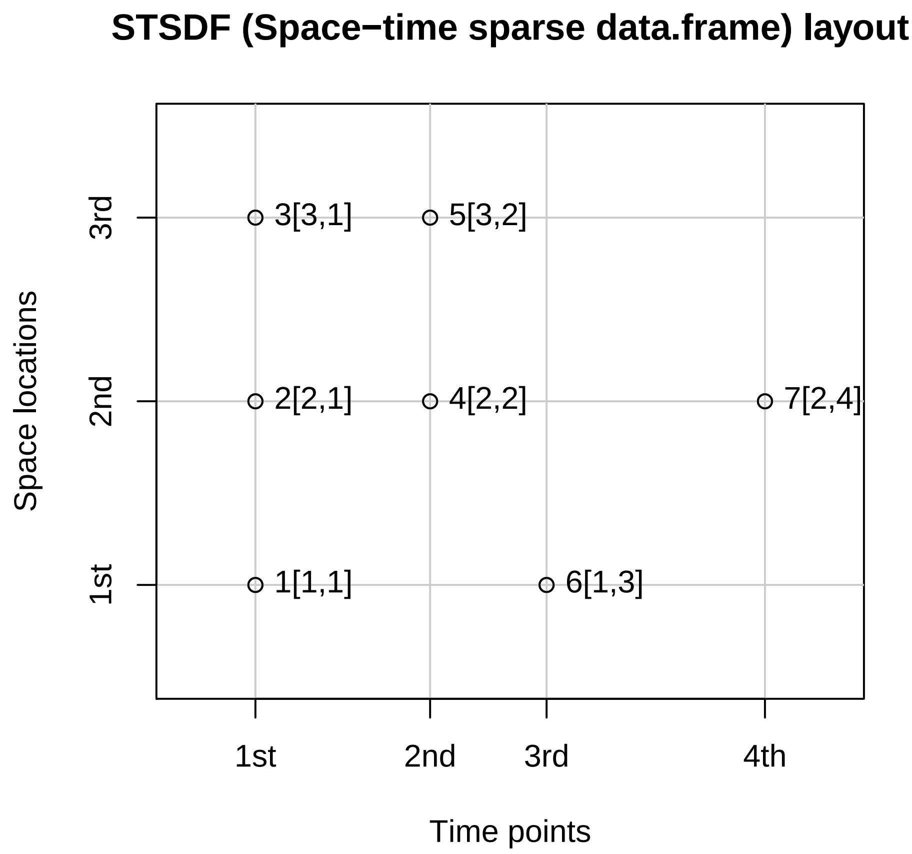


Figure 2: space-time layout of `STSDF` (`STS`: `ST`-Sparse) objects: only the non-missing part of the space-time combinations on a lattice are stored; numbers refer to the ordering of rows in the `data.frame`; an index is kept where e.g., [3,4] refers to the third item in the list of spatial locations and fourth item in the list of temporal points.

stored. For each k, an index $[i, j]$ is stored that refers which spatial location 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 locations 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 locations varies (locations of crimes registered per year, or spatially misaligned remote sensing images).

3.3. Irregular space-time `data.frame`

Space-time irregular `data.frame`s (STIDF, figure 3) are meant for the case where time and space points of measured values have no apparent organisation: for each measured value the spatial location 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. Locations and time points need not

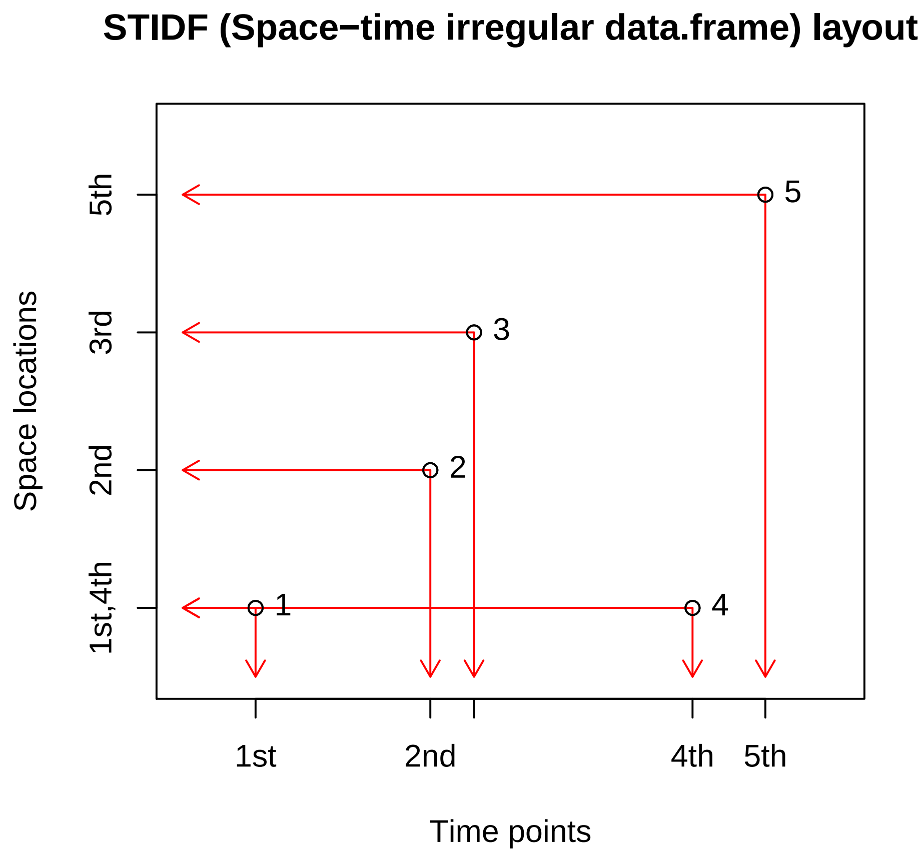


Figure 3: Space-time layout of `STIDF` (`STI`: `ST`-Irregular) objects: each observation has its spatial location and time stamp stored; in this example, spatial location 1 is stored twice – the fact that observations 1 and 4 have the same location is not registered.

be unique, and are replicated in case they are not.

4. Spatio-temporal full grid `data.frames` (`STFDF`)

For objects of class `STFDF`, time representation can be regular or irregular, as is supported by class `xts` in package **xts**. Spatial locations need to be of a class deriving from `Spatial` in package **sp**.

4.1. Class definition

```
R> library("spacetime")
R> showClass("ST")

Class "ST" [package "spacetime"]
```

```
Slots:

Name:       sp    time
Class: Spatial     xts

Known Subclasses:
Class "STS", directly
Class "STI", directly
Class "STF", directly
Class "STSDF", by class "STS", distance 2
Class "STIDF", by class "STI", distance 2
Class "STFDF", by class "STF", distance 2
Class "STIDFtraj", by class "STIDF", distance 3
```

```
R> showClass("STFDF")
```

```
Class "STFDF" [package "spacetime"]

Slots:

Name:        data         sp       time
Class: data.frame    Spatial        xts

Extends:
Class "STF", directly
Class "ST", by class "STF", distance 2
```

```
R> sp = cbind(x = c(0,0,1), y = c(0,1,1))
R> row.names(sp) = paste("point", 1:nrow(sp), sep="")
R> sp = SpatialPoints(sp)
R> time = as.POSIXct("2010-08-05", tz = "GMT")+3600*(10:13)
R> m = c(10,20,30) # means for each of the 3 point locations
R> mydata = rnorm(length(sp)*length(time),mean=rep(m, 4))
R> IDs = paste("ID",1:length(mydata), sep = "_")
R> mydata = data.frame(values = signif(mydata,3), ID=IDs)
R> stfdf = STFDF(sp, time, mydata)
```

```
R> str(stfdf)
```

```
Formal class 'STFDF' [package "spacetime"] with 3 slots
  ..@ data:'data.frame':        12 obs. of  2 variables:
  .. ..$ values: num [1:12] 9.28 18.8 30.7 10.4 19.4 30.2 10.6 18.7 30.7 11.8 ...
  .. ..$ ID    : Factor w/ 12 levels "ID_1","ID_10",..: 1 5 6 7 8 9 10 11 12 2 ...
  ..@ sp  :Formal class 'SpatialPoints' [package "sp"] with 3 slots
  .. .. ..@ coords     : num [1:3, 1:2] 0 0 1 0 1 1
  .. .. .. ..- attr(*, "dimnames")=List of 2
  .. .. .. .. ..$ : chr [1:3] "point1" "point2" "point3"
```

```
  .. .. .. .. ..$ : chr [1:2] "x" "y"
  .. .. ..@ bbox       : num [1:2, 1:2] 0 0 1 1
  .. .. .. ..- attr(*, "dimnames")=List of 2
  .. .. .. .. ..$ : chr [1:2] "x" "y"
  .. .. .. .. ..$ : chr [1:2] "min" "max"
  .. .. ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slots
  .. .. .. .. ..@ projargs: chr NA
  ..@ time:An 'xts' object from 2010-08-05 10:00:00 to 2010-08-05 13:00:00 containing:
  Data: int [1:4, 1] 1 2 3 4
  Indexed by objects of class: [POSIXct,POSIXt] TZ: GMT
  xts Attributes:
 NULL
```

4.2. Coercion to `data.frame`

The following coercion function creates a `data.frame`, using either the S3 (to set row.names) or S4 "as()" method. It gives data in the long format, meaning that time and space are replicated appropriately:

```
R> as.data.frame(stfdf, row.names = IDs)

      x y  sp.ID                time timedata values    ID
ID_1  0 0 point1 2010-08-05 10:00:00        1   9.28  ID_1
ID_2  0 1 point2 2010-08-05 10:00:00        1  18.80  ID_2
ID_3  1 1 point3 2010-08-05 10:00:00        1  30.70  ID_3
ID_4  0 0 point1 2010-08-05 11:00:00        2  10.40  ID_4
ID_5  0 1 point2 2010-08-05 11:00:00        2  19.40  ID_5
ID_6  1 1 point3 2010-08-05 11:00:00        2  30.20  ID_6
ID_7  0 0 point1 2010-08-05 12:00:00        3  10.60  ID_7
ID_8  0 1 point2 2010-08-05 12:00:00        3  18.70  ID_8
ID_9  1 1 point3 2010-08-05 12:00:00        3  30.70  ID_9
ID_10 0 0 point1 2010-08-05 13:00:00        4  11.80 ID_10
ID_11 0 1 point2 2010-08-05 13:00:00        4  21.00 ID_11
ID_12 1 1 point3 2010-08-05 13:00:00        4  30.30 ID_12

R> as(stfdf, "data.frame")[1:4,]

  x y  sp.ID                time timedata values   ID
1 0 0 point1 2010-08-05 10:00:00        1   9.28 ID_1
2 0 1 point2 2010-08-05 10:00:00        1  18.80 ID_2
3 1 1 point3 2010-08-05 10:00:00        1  30.70 ID_3
4 0 0 point1 2010-08-05 11:00:00        2  10.40 ID_4
```

Note that `sp.ID` denotes the ID of the spatial location; coordinates are shown for point, pixel or grid cell centre locations; in case locations refer to lines or polygons, the line's start coordinate and coordinate centre of weight are given, respectively, as the coordinate values in this representation.

For a single attribute, we can obtain a `data.frame` object if we properly unstack the column, giving the data in both its wide formats when in addition we apply transpose `t()`:

```
R> unstack(stfdf)

                    point1 point2 point3
2010-08-05 10:00:00   9.28   18.8   30.7
2010-08-05 11:00:00  10.40   19.4   30.2
2010-08-05 12:00:00  10.60   18.7   30.7
2010-08-05 13:00:00  11.80   21.0   30.3

R> t(unstack(stfdf))

       2010-08-05 10:00:00 2010-08-05 11:00:00 2010-08-05 12:00:00
point1                9.28                10.4                10.6
point2               18.80                19.4                18.7
point3               30.70                30.2                30.7
       2010-08-05 13:00:00
point1                11.8
point2                21.0
point3                30.3

R> unstack(stfdf, which = 2)

                    point1 point2 point3
2010-08-05 10:00:00   ID_1   ID_2   ID_3
2010-08-05 11:00:00   ID_4   ID_5   ID_6
2010-08-05 12:00:00   ID_7   ID_8   ID_9
2010-08-05 13:00:00  ID_10  ID_11  ID_12
```

4.3. Coercion to matrix or objects of class `xts`

We can coerce an object of class STFDF to an object of class `xts` if we select a single numeric attribute:

```
R> as(stfdf[,,"values"], "xts")

                    point1 point2 point3
2010-08-05 10:00:00   9.28   18.8   30.7
2010-08-05 11:00:00  10.40   19.4   30.2
2010-08-05 12:00:00  10.60   18.7   30.7
2010-08-05 13:00:00  11.80   21.0   30.3
```

An `xts` object is a matrix, with time (in some form) stored in an attribute, and time non-decreasing over rows. Method `index` retrieves the time points:

```
R> x = as(stfdf[,,"values"], "xts")
R> index(x)
```

```
[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"
```

4.4. Spatial, temporal and spatio-temporal aggregation

Aggregating values over *all* space locations or time instances can be done by coercing to `xts` (i.e., to a matrix form) and then using `apply`, either over space:

```
R> x = as(stfdf[,,"values"], "xts")
R> apply(x, 1, mean)

2010-08-05 10:00:00 2010-08-05 11:00:00 2010-08-05 12:00:00
           19.59333            20.00000            20.00000
2010-08-05 13:00:00
           21.03333
```

or over time:

```
R> apply(x, 2, mean)

point1 point2 point3
10.520 19.475 30.475
```

Aggregation 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) can be done using the method `aggregate`. More information with illustrated examples is found in the vignette on this, obtained by:

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

To obtain the 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 GI Science as the *spatial overlay*.

4.5. Attribute retrieval and replacement: `[[` and `$`

We can define the `[[` and `$` retrieval and replacement methods for all classes deriving from ST at once. Here are some examples:

```
R> stfdf[[1]]

 [1]  9.28 18.80 30.70 10.40 19.40 30.20 10.60 18.70 30.70 11.80 21.00
[12] 30.30

R> stfdf[["values"]]
```

```
 [1]  9.28 18.80 30.70 10.40 19.40 30.20 10.60 18.70 30.70 11.80 21.00
[12] 30.30
```

```
R> stfdf[["newVal"]] = rnorm(12)
R> stfdf$ID
```

```
 [1] ID_1  ID_2  ID_3  ID_4  ID_5  ID_6  ID_7  ID_8  ID_9  ID_10 ID_11
[12] ID_12
12 Levels: ID_1 ID_10 ID_11 ID_12 ID_2 ID_3 ID_4 ID_5 ID_6 ... ID_9
```

```
R> stfdf$ID = paste("OldIDs", 1:12, sep="")
R> stfdf$NewID = paste("NewIDs", 12:1, sep="")
R> stfdf
```

```
An object of class "STFDF"
Slot "data":
   values       ID      newVal    NewID
1    9.28  OldIDs1 -0.15389513 NewIDs12
2   18.80  OldIDs2 -0.35297257 NewIDs11
3   30.70  OldIDs3 -0.30063030 NewIDs10
4   10.40  OldIDs4 -1.27681523  NewIDs9
5   19.40  OldIDs5 -0.46543726  NewIDs8
6   30.20  OldIDs6 -0.54947509  NewIDs7
7   10.60  OldIDs7 -1.85959239  NewIDs6
8   18.70  OldIDs8 -0.99049105  NewIDs5
9   30.70  OldIDs9  0.04464429  NewIDs4
10  11.80 OldIDs10  1.12077888  NewIDs3
11  21.00 OldIDs11  0.69355910  NewIDs2
12  30.30 OldIDs12  0.82508981  NewIDs1

Slot "sp":
SpatialPoints:
       x y
point1 0 0
point2 0 1
point3 1 1
Coordinate Reference System (CRS) arguments: NA

Slot "time":
                    [,1]
2010-08-05 10:00:00    1
2010-08-05 11:00:00    2
2010-08-05 12:00:00    3
2010-08-05 13:00:00    4
```

4.6. 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` – this is one of the stronger intuitive areas of R syntax. 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 entities (points, lines, polygons, pixels) and rows as the attributes – a convention that was partially broken for class `SpatialGridDataFrame`, where `a[i,j,k]` could select the k-th attribute 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).

For the spatio-temporal data classes described here, `a[i,j,k]` selects spatial entity/entities `i`, temporal entity/entities `j`, and attribute(s) `k`:

```
R> stfdf[,1] # SpatialPointsDataFrame

  coordinates values      ID     newVal    NewID
1      (0, 0)   9.28 OldIDs1 -0.1538951 NewIDs12
2      (0, 1)  18.80 OldIDs2 -0.3529726 NewIDs11
3      (1, 1)  30.70 OldIDs3 -0.3006303 NewIDs10

R> stfdf[,,1]

An object of class "STFDF"
Slot "data":
   values
1    9.28
2   18.80
3   30.70
4   10.40
5   19.40
6   30.20
7   10.60
8   18.70
9   30.70
10  11.80
11  21.00
12  30.30

Slot "sp":
SpatialPoints:
       x y
point1 0 0
point2 0 1
point3 1 1
Coordinate Reference System (CRS) arguments: NA

Slot "time":
                    [,1]
2010-08-05 10:00:00    1
2010-08-05 11:00:00    2
```

```
2010-08-05 12:00:00    3
2010-08-05 13:00:00    4

R> stfdf[1,,1] # xts

                    values
2010-08-05 10:00:00   9.28
2010-08-05 11:00:00  10.40
2010-08-05 12:00:00  10.60
2010-08-05 13:00:00  11.80

R> stfdf[,,"ID"]

An object of class "STFDF"
Slot "data":
         ID
1   OldIDs1
2   OldIDs2
3   OldIDs3
4   OldIDs4
5   OldIDs5
6   OldIDs6
7   OldIDs7
8   OldIDs8
9   OldIDs9
10 OldIDs10
11 OldIDs11
12 OldIDs12

Slot "sp":
SpatialPoints:
       x y
point1 0 0
point2 0 1
point3 1 1
Coordinate Reference System (CRS) arguments: NA

Slot "time":
                    [,1]
2010-08-05 10:00:00    1
2010-08-05 11:00:00    2
2010-08-05 12:00:00    3
2010-08-05 13:00:00    4

R> stfdf[1,,"values", drop = FALSE] # stays STFDF:
```

```
An object of class "STFDF"
Slot "data":
  values
1   9.28
2  10.40
3  10.60
4  11.80

Slot "sp":
SpatialPoints:
       x y
point1 0 0
Coordinate Reference System (CRS) arguments: NA

Slot "time":
                    [,1]
2010-08-05 10:00:00    1
2010-08-05 11:00:00    2
2010-08-05 12:00:00    3
2010-08-05 13:00:00    4
```

```
R> stfdf[,1, drop=FALSE] #stays STFDF
```

```
An object of class "STFDF"
Slot "data":
  values      ID     newVal    NewID
1   9.28 OldIDs1 -0.1538951 NewIDs12
2  18.80 OldIDs2 -0.3529726 NewIDs11
3  30.70 OldIDs3 -0.3006303 NewIDs10

Slot "sp":
SpatialPoints:
       x y
point1 0 0
point2 0 1
point3 1 1
Coordinate Reference System (CRS) arguments: NA

Slot "time":
                    ..1
2010-08-05 10:00:00   1
```

Clearly, unless `drop=FALSE`, selecting a single time or single location object results in an object that is no longer spatio-temporal; see also Section 7.

5. Space-time sparse `data.frames` (`STSDF`)

Space-time sparse `data.frame`s have a layout over a grid, meaning that particular times and locations are typically present more than once, but only the data for the time/location combinations are stored. An index keeps the link between the measured values (rows) in the data slot, and the locations and times.

5.1. Class definition

```
R> showClass("STSDF")

Class "STSDF" [package "spacetime"]

Slots:

Name:        data      index         sp       time
Class: data.frame     matrix    Spatial        xts

Extends:
Class "STS", directly
Class "ST", by class "STS", distance 2
```

In this class, index is an $n \times 2$ matrix. If in this index row i has entry $[j, k]$, it means that the i-th row in the data slot corresponds to location j and time k.

6. Spatio-temporal irregular `data.frames` (`STIDF`)

Space-time irregular `data.frame`s store for each data record the location and time. No index is kept. Location and time need not be organized. Data are stored such that time is ordered (as it is an `xts` object).

6.1. Class definition

```
R> showClass("STIDF")

Class "STIDF" [package "spacetime"]

Slots:

Name:        data         sp       time
Class: data.frame    Spatial        xts

Extends:
Class "STI", directly
Class "ST", by class "STI", distance 2

Known Subclasses: "STIDFtraj"
```

```
R> sp = expand.grid(x = 1:3, y = 1:3)
R> row.names(sp) = paste("point", 1:nrow(sp), sep="")
R> sp = SpatialPoints(sp)
R> time = as.POSIXct("2010-08-05", tz = "GMT")+3600*(11:19)
R> m = 1:9 * 10 # means for each of the 9 point locations
R> mydata = rnorm(length(sp), mean=m)
R> IDs = paste("ID",1:length(mydata))
R> mydata = data.frame(values = signif(mydata,3),ID=IDs)
R> stidf = STIDF(sp, time, mydata)
R> stidf

An object of class "STIDF"
Slot "data":
  values   ID
1   11.1 ID 1
2   18.5 ID 2
3   30.2 ID 3
4   40.4 ID 4
5   47.7 ID 5
6   58.2 ID 6
7   69.3 ID 7
8   78.3 ID 8
9   89.0 ID 9

Slot "sp":
SpatialPoints:
      x y
 [1,] 1 1
 [2,] 2 1
 [3,] 3 1
 [4,] 1 2
 [5,] 2 2
 [6,] 3 2
 [7,] 1 3
 [8,] 2 3
 [9,] 3 3
Coordinate Reference System (CRS) arguments: NA

Slot "time":
                    [,1]
2010-08-05 11:00:00    1
2010-08-05 12:00:00    2
2010-08-05 13:00:00    3
2010-08-05 14:00:00    4
2010-08-05 15:00:00    5
2010-08-05 16:00:00    6
2010-08-05 17:00:00    7
```

```
2010-08-05 18:00:00    8
2010-08-05 19:00:00    9
```

6.2. Methods

Selection takes place with the `[` method:

```
R> stidf[1:2,]

An object of class "STIDF"
Slot "data":
  values   ID
1   11.1 ID 1
2   18.5 ID 2

Slot "sp":
SpatialPoints:
     x y
[1,] 1 1
[2,] 2 1
Coordinate Reference System (CRS) arguments: NA

Slot "time":
                    [,1]
2010-08-05 11:00:00    1
2010-08-05 12:00:00    2
```

7. Further methods: snapshot, history, coercion

7.1. *Snap* and *Hist*

A time snapshot (Galton 2004) to a particular moment in time can be obtained through selecting a particular time moment:

```
R> stfdf[,time[3]]

  coordinates values       ID    newVal   NewID
1      (0, 0)   11.8 OldIDs10 1.1207789 NewIDs3
2      (0, 1)   21.0 OldIDs11 0.6935591 NewIDs2
3      (1, 1)   30.3 OldIDs12 0.8250898 NewIDs1
```

by default, a simplified object of the underlying `Spatial` class for this particular time is obtained (`drop=TRUE`); if we specify `drop = FALSE`, the class will not be changed:

```
R> class(stfdf[,time[3]])
```

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

```
R> class(stfdf[,time[3],drop=FALSE])
```

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

A time series (or *history*, according to Galton, 2004) for a single particular location is obtained by selecting this location, e.g.,

```
R> stfdf[1, , "values"]
```

```
                    values
2010-08-05 10:00:00   9.28
2010-08-05 11:00:00  10.40
2010-08-05 12:00:00  10.60
2010-08-05 13:00:00  11.80
```

Again, the class is not reduced to the simpler when `drop = FALSE` is specified:

```
R> class(stfdf[1,])
```

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

```
R> class(stfdf[1,drop=FALSE])
```

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

For objects of class `STIDF`, `drop = TRUE` results in a `Spatial` object when a single time value is selected.

7.2. Coercion between ST*xxx* classes

Coercion from full to sparse and/or irregular space-time `data.frame`s, we can use as:

```
R> class(stfdf)
```

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

```
R> class(as(stfdf, "STSDF"))
```

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

R> class(as(as(stfdf, "STSDF"), "STIDF"))

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

R> class(as(stfdf, "STIDF"))

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

On our way back, the reverse coercion takes place:

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

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

R> class(as(as(x, "STSDF"), "STFDF"))

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

R> class(as(x, "STFDF"))

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

R> xx = as(x, "STFDF")
R> identical(stfdf, xx)

[1] TRUE
```

7.3. Coercion to class `SpatialXxDataFrame`

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) attribute is retained:

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

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

R> xs1

       coordinates X2010.08.05.10.00.00 X2010.08.05.11.00.00
point1      (0, 0)                 9.28                 10.4
point2      (0, 1)                18.80                 19.4
point3      (1, 1)                30.70                 30.2
       X2010.08.05.12.00.00 X2010.08.05.13.00.00
point1                 10.6                 11.8
point2                 18.7                 21.0
point3                 30.7                 30.3
```

as time stamps do not work well as column names, this object gets the proper times 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> xs2 = as(x, "Spatial")
R> class(xs2)

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

R> xs2[1:4,]

  coordinates values      ID     newVal    NewID                time
1      (0, 0)   9.28 OldIDs1 -0.1538951 NewIDs12 2010-08-05 10:00:00
2      (0, 1)  18.80 OldIDs2 -0.3529726 NewIDs11 2010-08-05 10:00:00
3      (1, 1)  30.70 OldIDs3 -0.3006303 NewIDs10 2010-08-05 10:00:00
4      (0, 0)  10.40 OldIDs4 -1.2768152  NewIDs9 2010-08-05 11:00:00
```

8. Graphs of spatio-temporal data: `stplot`

8.1. 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 separte 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 6. The `stplot` is a wrapper around `spplot` in package `sp`, and inherits most of its options.
- y and x denote value and time; one panel for each spatial location, colors may different attributes (`type="tp"`)
- y and x denote value and time; one panel for each attribute, colors may denote different stations (`type="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. An example on a so-called Hovmöller plot of the sea surface temperature data in Cressie and Wikle (2011) is obtained by

```
R> demo(CressieWikle)
```

Hovmöller plots only make sense 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 8.

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).

8.2. 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 7 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)
```

9. Spatial footprint or support, time intervals

9.1. 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 define and to compute with time intervals (e.g., Allen (1983)). It does not provide structures to attach these intervals to time series data.

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:

```
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"
```

9.2. 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 polygons, the assumption will be that values reflect aggregates 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.

10. 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.

10.1. North Carolina SIDS

As an example, the North Carolina Sudden Infant Death Syndrome (sids) data 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. Figure 4 shows the plot generated.

```
R> library("maptools")
R> fname = system.file("shapes/sids.shp", package="maptools")[1]
R> nc = readShapePoly(fname, proj4string=CRS("+proj=longlat +datum=NAD27"))
R> data = data.frame(
+          BIR = c(nc$BIR74, nc$BIR79),
+          NWBIR = c(nc$NWBIR74, nc$NWBIR79),
+          SID = c(nc$SID74, nc$SID79))
R> time = as.POSIXct(strptime(c("1974-01-01", "1979-01-01"), "%Y-%m-%d"),
+          tz = "GMT")
R> nct = STFDF(
+          sp = as(nc, "SpatialPolygons"),
+          time = time,
+          data = data)
R> stplot(nct[,,"SID"], c("1974-1978", "1979-1984"))
```

10.2. Panel data

The panel data discussed in Section 2 are imported as a full ST data.frame (STFDF), and linked to the proper state polygons of maps. Both `Produc` and the states in package **maps**

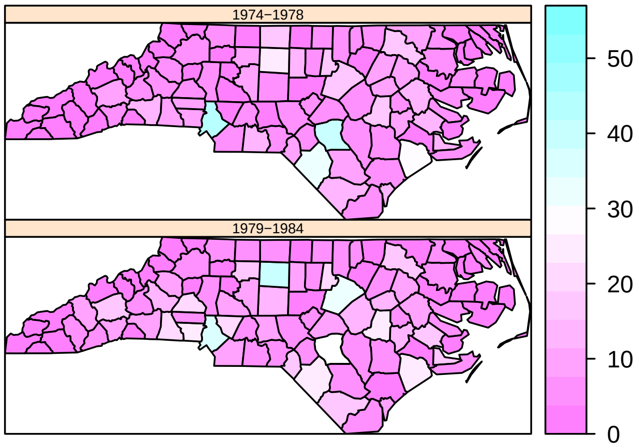


Figure 4: North Carolina sudden infant death syndrome (sids) data.

(Brownrigg and Minka 2011) order states alphabetically; the only thing to watch out for is that the former does not include District of Columbia, but the latter does (record 8):

```
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)
R> library("plm")
R> data("Produc")
R> yrs = 1970:1986
R> time = as.POSIXct(paste(yrs, "-01-01", sep=""), tz = "GMT")
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> stplot(Produc.st[,,"unemp"], yrs)
```

(The plot itself was omitted for reasons of file size.) Time and state were not removed from the data table on construction; printing these data as a `data.frame` confirms that time and state were matched correctly. The routines in package **plm** can be used on the data, back transformed to a data.frame, when `index` is specified (the first two columns from the back-transformed data no longer contain state and year):

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

Oneway (individual) effect Within Model

Call:
```

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

Balanced Panel: n=48, T=17, N=816

Residuals :
    Min.  1st Qu.   Median  3rd Qu.     Max.
-0.12000 -0.02370 -0.00204  0.01810  0.17500

Coefficients :
             Estimate  Std. Error t-value  Pr(>|t|)
log(pcap) -0.02614965  0.02900158 -0.9017    0.3675
log(pc)    0.29200693  0.02511967 11.6246 < 2.2e-16 ***
log(emp)   0.76815947  0.03009174 25.5273 < 2.2e-16 ***
unemp     -0.00529774  0.00098873 -5.3582 1.114e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Total Sum of Squares:    18.941
Residual Sum of Squares: 1.1112
R-Squared      :  0.94134
      Adj. R-Squared :  0.88135
F-statistic: 3064.81 on 4 and 764 DF, p-value: < 2.22e-16
```

10.3. 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 5.

```
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"
```

The first thing to do with the wind speed values is to reshape these data. Unlike the North Carolina SIDS data of Section 10.1, for this data space is sparse and time is rich, and so the data in `data.frame wind` come in space-wide form with stations time series in columns:

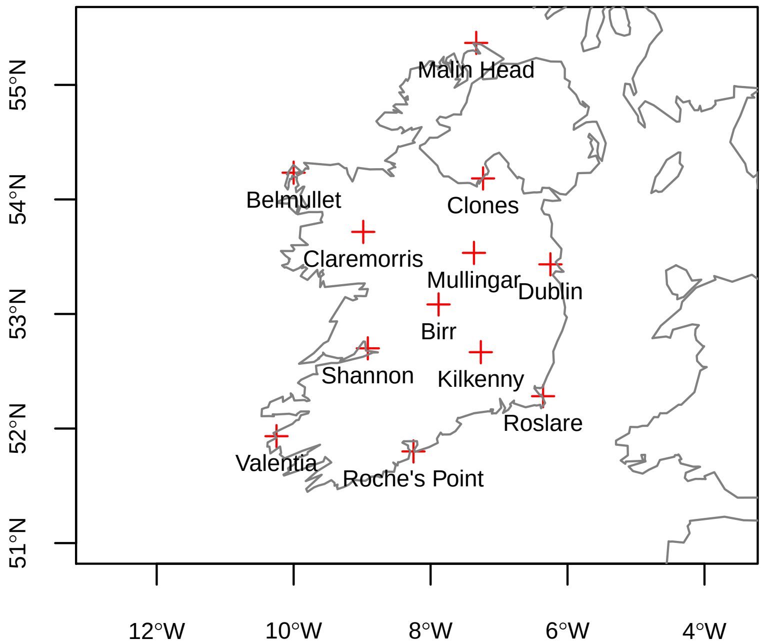


Figure 5: Station locations for Irish wind data.

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

```
R> wind$time = ISOdate(wind$year+1900, wind$month, wind$day)
R> wind$jday = as.numeric(format(wind$time, '%j'))
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, and project the longitude/latitude coordinates and country boundary to the appropriate UTM zone, using `spTransform` in package **rgdal** (Keitt, Bivand, Pebesma, and Rowlingson 2011) for coordinate transformation:

```
R> # order locations to order of columns in wind;
R> # connect station names to location coordinates
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$Station
R> pts = SpatialPoints(pts)
R> # convert to utm zone 29, to be able to do interpolation in
R> # proper Euclidian (projected) space:
R> proj4string(pts) = "+proj=longlat +datum=WGS84"
R> library("rgdal")
R> utm29 = CRS("+proj=utm +zone=29 +datum=WGS84")
R> pts = spTransform(pts, utm29)
R> # construct from space-wide table:
R> w = stConstruct(velocities, space = list(values = 1:ncol(velocities)),
+          time = wind$time, SpatialObj = pts)
R> library("maptools")
R> m = map2SpatialLines(
+          map("worldHires", xlim = c(-11,-5.4), ylim = c(51,55.5), plot=F))
```

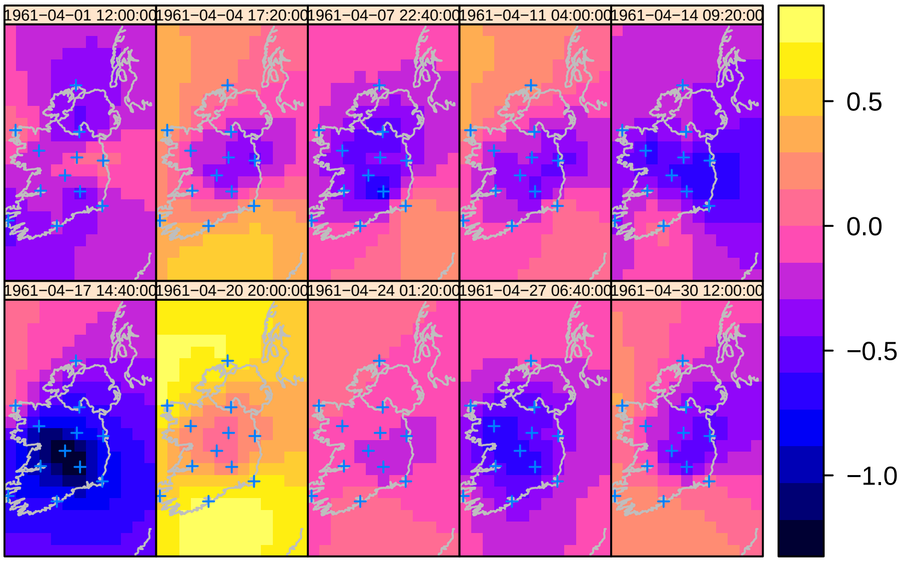


Figure 6: Space-time interpolations of wind (square root transformed, detrended) over Ireland using a separable product covariance model, for 10 time points regularly distributed over the month for which daily data was considered (April, 1961).

```
R> proj4string(m) = "+proj=longlat +datum=WGS84"
R> m = spTransform(m, utm29)
R> # setup grid
R> grd = SpatialPixels(SpatialPoints(makegrid(m, n = 300)),
+           proj4string = proj4string(m))
R> # select april 1961:
R> w = w[, "1961-04"]
R> # 10 prediction time points, evenly spread over this month:
R> n = 10
R> tgrd = xts(1:n, seq(min(index(w)), max(index(w)), length=n))
R> # separable covariance model, exponential 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, STF(grd, tgrd), v)
R> wind.ST = STFDF(grd, tgrd, data.frame(sqrt_speed = pred))
```

the results of which are shown in figure 6, created with `stplot`.

10.4. 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:

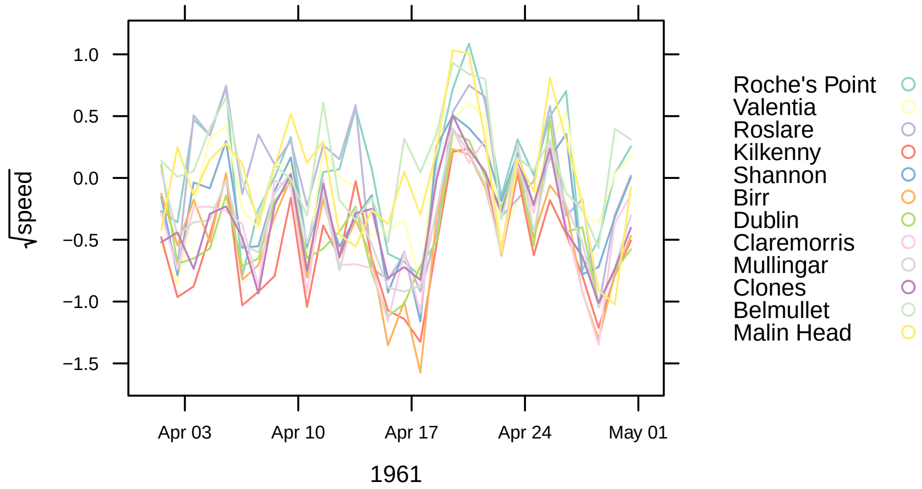


Figure 7: Time series plot of daily wind speed at 12 stations, used for interpolation in figure 6.

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

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 6.

10.5. 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 attribute 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:

```
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"))
R> # convert to SPSDFtraj, and plot:
R> setAs("trip", "STIDFtraj",
+          function(from) {
```

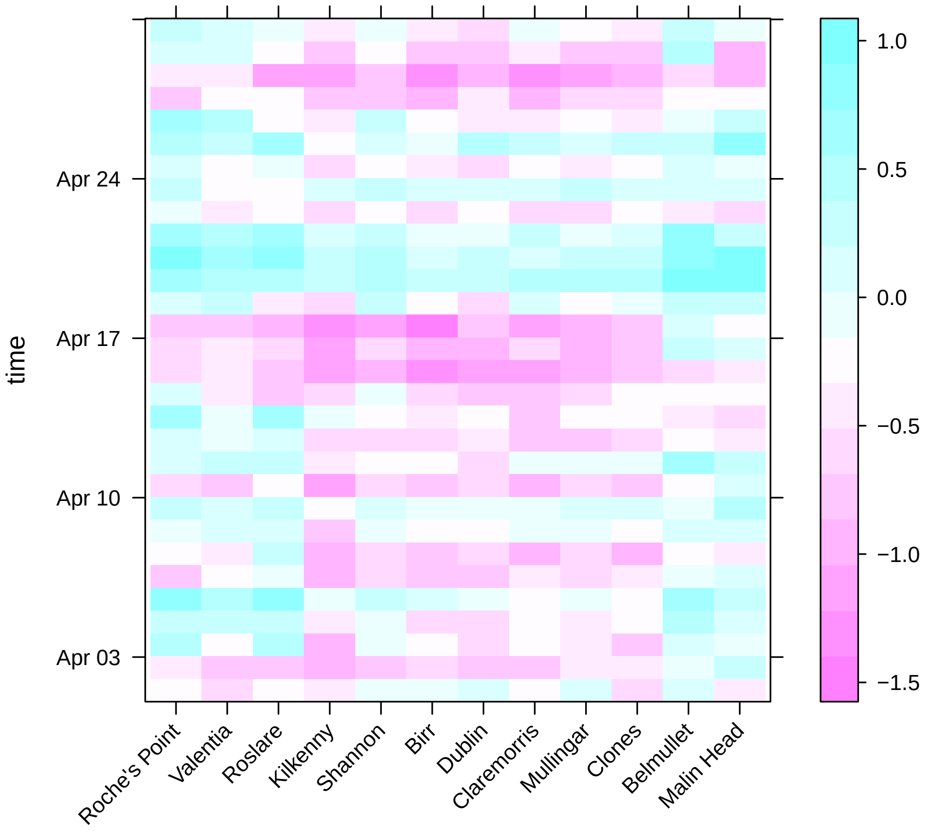


Figure 8: Space-time (Hovmöller) plot of wind station data.

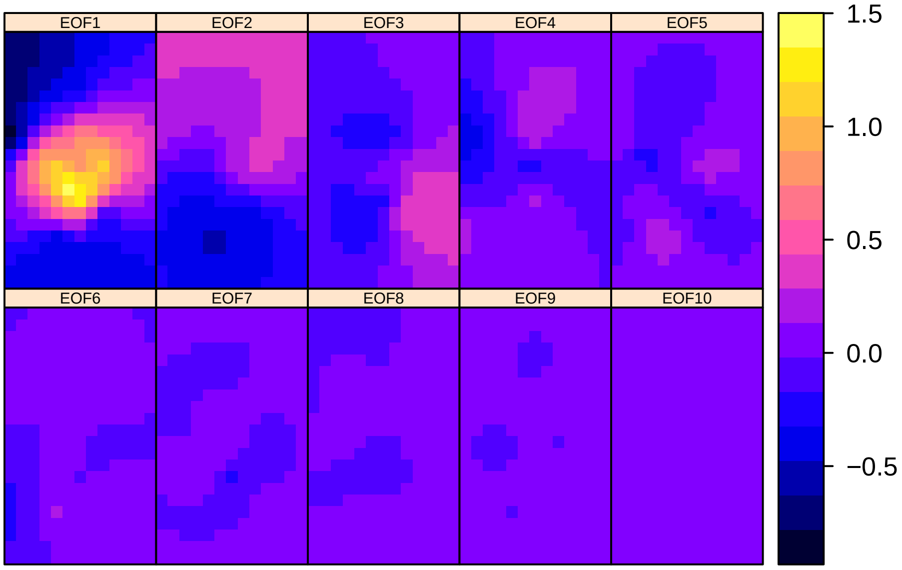
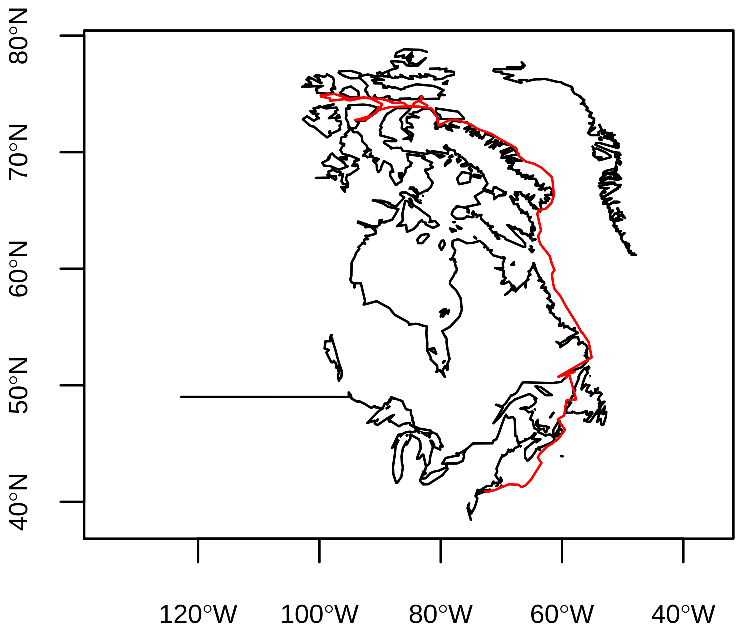


Figure 9: EOFs of space-time interpolations of wind over Ireland (for spatial reference, see figure 6), for the 10 time points at which daily data was chosen above (April, 1961).

```
+                    from$burst = from[[from@TOR.columns[2]]]
+                    time = from[[from@TOR.columns[1]]]
+                    new("STIDFtraj", STIDF(as(from, "SpatialPoints"), time, from@data))
+          }
+  )
R> x = as(tr, "STIDFtraj")
R> m = map2SpatialLines(map("world",
+          xlim = c(-100,-50), ylim = c(40,77), plot=F))
R> proj4string(m) = "+proj=longlat +datum=WGS84"
R> plot(m, axes=TRUE, cex.axis =.7)
R> plot(x, add=TRUE, col = "red")
R> # convert back, compare:
R> setAs("STIDFtraj", "trip", function(from) {
+                    from$time = index(from@time)
+                    trip(SpatialPointsDataFrame(from@sp, from@data), c("time", "burst"))
+          }
+  )
R> y = as(x, "trip")
R> y$burst = NULL
R> all.equal(y, tr, check.attributes = FALSE)
```

```
[1] TRUE
```



10.6. Trajectory data: ltraj in adehabitatLT

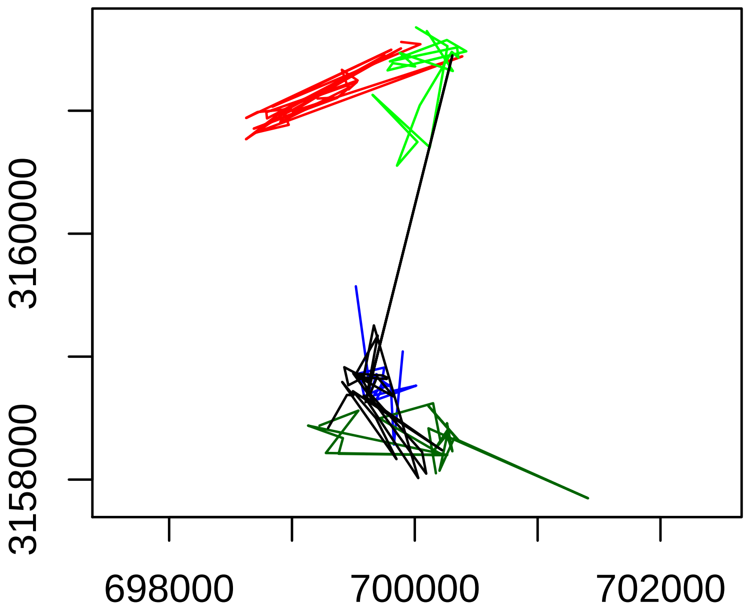
Trajectory objects of class `ltraj` in package **adehabitatLT** (Calenge, Dray, and Royer-Carenzi 2008) are lists of bursts, sets of sequentially, connected space-time points at which an object is registered. When converting a list to a single STIDF object, the ordering is according to time, and the subsequent objects become unconnected. In the coercion back to `ltraj`, based on ID and burst the appropriate bursts are restored. A simple plot is obtained by:

```
R> library("adehabitatLT")
R> # from: adehabitat/demo/managltraj.r
R> # demo(managltraj)
R> data("puechabonsp")
R> # locations:
R> locs = puechabonsp$relocs
R> xy = coordinates(locs)
R> ### Conversion of the date to the format POSIX
R> da = as.character(locs$Date)
R> da = as.POSIXct(strptime(as.character(locs$Date),"%y%m%d"), tz = "GMT")
R> ## object of class "ltraj"
R> ltr = as.ltraj(xy, da, id = locs$Name)
```

```
R> foo = function(dt) dt > 100*3600*24
R> ## The function foo returns TRUE if dt is longer than 100 days
R> ## We use it to cut ltr:
R> l2 = cutltraj(ltr, "foo(dt)", nextr = TRUE)
R> stidfTrj = as(l2, "STIDFtraj")
R> ltr0 = as(stidfTrj, "ltraj")
R> all.equal(l2, ltr0, check.attributes = FALSE)

[1] TRUE

R> plot(stidfTrj, col = c("red", "green", "blue", "darkgreen", "black"),
+          axes=TRUE)
```



A more complicated plot is shown in figure 10, obtained by the command

```
R> stplot(stidfTrj,by="time*id")
```

the output of which is shown in figure 10.

10.7. Country shapes in cshapes

The **cshapes** (Weidmann, Kuse, and Gleditsch 2011) package contains a GIS dataset of country

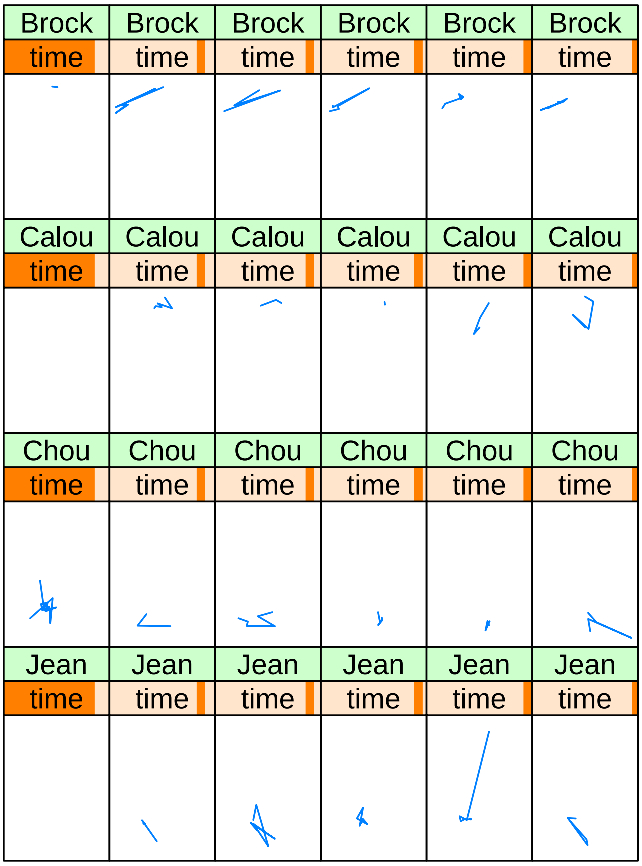


Figure 10: Trajectories, by id (rows) and time (columns).

boundaries (1946-2008), and includes functions for data extraction and the computation of weights matrices. The data set consist of a `SpatialPolygonsDataFrame`, with the following attributes:

```
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[3]) and "GW" Gleditsch and Ward (1999). The attributes COWSMONTH and COWEMONTH denote the start month and end month, respectively, according to the COW data base.

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

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

In the following fragment, an unordered list of times `t` is passed on to `STIDF`, and this will cause the geometries and attributes to be reordered (in the order of `t`):

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

11. Discussion

Building on existing infrastructure for spatial and temporal data, we have successfully implemented a coherent set of classes for spatio-temporal data, that provides regular space-time layouts, partially regular (sparse) space-time layouts and irregular space-time layouts. 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 used.

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

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 attributes that are purely spatial or purely temporal. Selection can be done based on spatial characteristics, time (intervals), or attributes, 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 reflects current practice. As the time series community, at least as far as reflected in the CRAN Task View on Time Series Analysis[4], does not care about storing time intervals, there must be a ground for this. One reason may be that time instances automatically refer to an interval, e.g. a date represents a full day, a POSIXt value a full second. Another may be that the time instance representation has an analogy to storing spatial polygons as topology, whereas time intervals may have this to representing spatial polygons as sets of rings. The interval/rings representation may be easier for some cases, but may also result in increased complexity as they may be inconsistent: intervals and rings may overlap. Representing temporally changing spatial polygons in a spatio-temporally topologically correct way is still a challenge.

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 Muenster (Ben Gräler, Katharina Henneböhl, Daniel Nüst) 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.

[4]http://cran.r-project.org/web/views/TimeSeries.html

Brownrigg R, Minka TP (2011). *maps: Draw Geographical Maps.* R package version 2.1-6, URL http://CRAN.R-project.org/package=maps.

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.

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.

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.

Weidmann NB, Kuse D, Gleditsch KS (2011). *cshapes: CShapes Dataset and Utilities*. R package version 0.3-1, URL http://CRAN.R-project.org/package=cshapes.

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

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