Classes and methods for spatio-temporal data in R: the spacetime package

ifgi
Institute for Geoinformatics
University of Münster

Edzer Pebesma

July 15, 2011

Abstract

This document describes a set of classes and methods for spatio-temporal data in R. 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 obvious long or wide format.

Contents

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 package is a start to fill this gap.

Spatio-temporall 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 et al. (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 current version of the package is experimental, class definitions and methods are subject to change.

We use `xts` for time, not only because it supports various basic types to represent time or date[1], but also because it has good tools for *aggregation* over time using arbitrary aggregation functions, and a very flexible syntax to select time periods that adheres ISO 8601[2]. We do not use the `xts` objects to store the spatio-temporal attribute information, as it is restricted to `matrix` objects, and hence can only store a single type, and not combine numeric and factor. Instead, as in the classes of `sp`, 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`.

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 in time is found in the North Carolina Sudden Infant Death Syndrome (sids) data set, which is in the **time wide format**:

```
> library(foreign)
> 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, for which the first six records are

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

[1] currently supported by xts are: `Date`, `POSIXct`, `timeDate`, `yearmon`, and `yearqtr`; Ripley and Hornik, 2001; advice on which class to use is found in Grothendiek and Petzoldt, 2004

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

```
  year month day   RPT   VAL   ROS   KIL   SHA  BIR   DUB   CLA   MUL   CLO
1   61     1   1 15.04 14.96 13.17  9.29 13.96 9.87 13.67 10.25 10.83 12.58
2   61     1   2 14.71 16.88 10.83  6.50 12.62 7.67 11.50 10.04  9.79  9.67
3   61     1   3 18.50 16.88 12.33 10.13 11.17 6.17 11.25  8.04  8.50  7.67
4   61     1   4 10.58  6.63 11.75  4.58  4.54 2.88  8.63  1.79  5.83  5.88
5   61     1   5 13.33 13.25 11.42  6.17 10.71 8.21 11.92  6.54 10.92 10.34
6   61     1   6 13.21  8.12  9.96  6.67  5.37 4.50 10.67  4.42  7.17  7.50
    BEL   MAL
1 18.50 15.04
2 17.54 13.83
3 12.75 12.71
4  5.46 10.88
5 12.92 11.83
6  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, the first five records are

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

```
    state year     pcap     hwy   water    util       pc   gsp    emp unemp
1 ALABAMA 1970 15032.67 7325.80 1655.68 6051.20 35793.80 28418 1010.5   4.7
2 ALABAMA 1971 15501.94 7525.94 1721.02 6254.98 37299.91 29375 1021.9   5.2
3 ALABAMA 1972 15972.41 7765.42 1764.75 6442.23 38670.30 31303 1072.3   4.7
4 ALABAMA 1973 16406.26 7907.66 1742.41 6756.19 40084.01 33430 1135.5   3.9
5 ALABAMA 1974 16762.67 8025.52 1734.85 7002.29 42057.31 33749 1169.8   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 documents 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[3] of observations for spatial location (points, lines, polygons, grid cells) $s_i, i = 1, ..., n$ 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 need to be in time order, as `xts` objects are used to store them.

In this data class (figure 1), for each location, the same temporal sequence of data is sampled. Altenatively 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 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 can 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).

3.3 Irregular space-time `data.frame`

Space-time irregular `data.frames` (STIDF, figure 3) are those where time and space points of measured values can have no organization: for each measured value the spatial location and time point is stored, as in the long format. This is equivalent to the most 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 be unique, but will be 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

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

Class "ST" [package "spacetime"]

Slots:
```

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

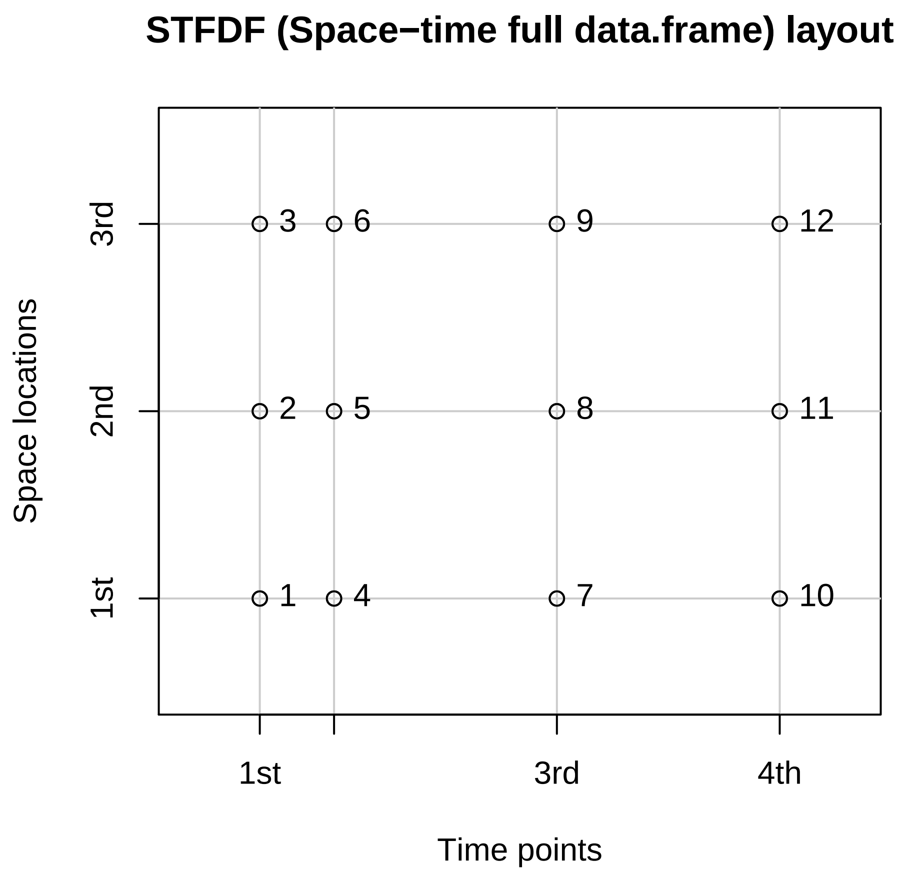


Figure 1: space-time layout of STFDF (STF: ST-Full) objects: all space-time combinations are stored; numbers refer to the ordering of rows in the `data.frame` with measured values: time is kept ordered, space cycles first

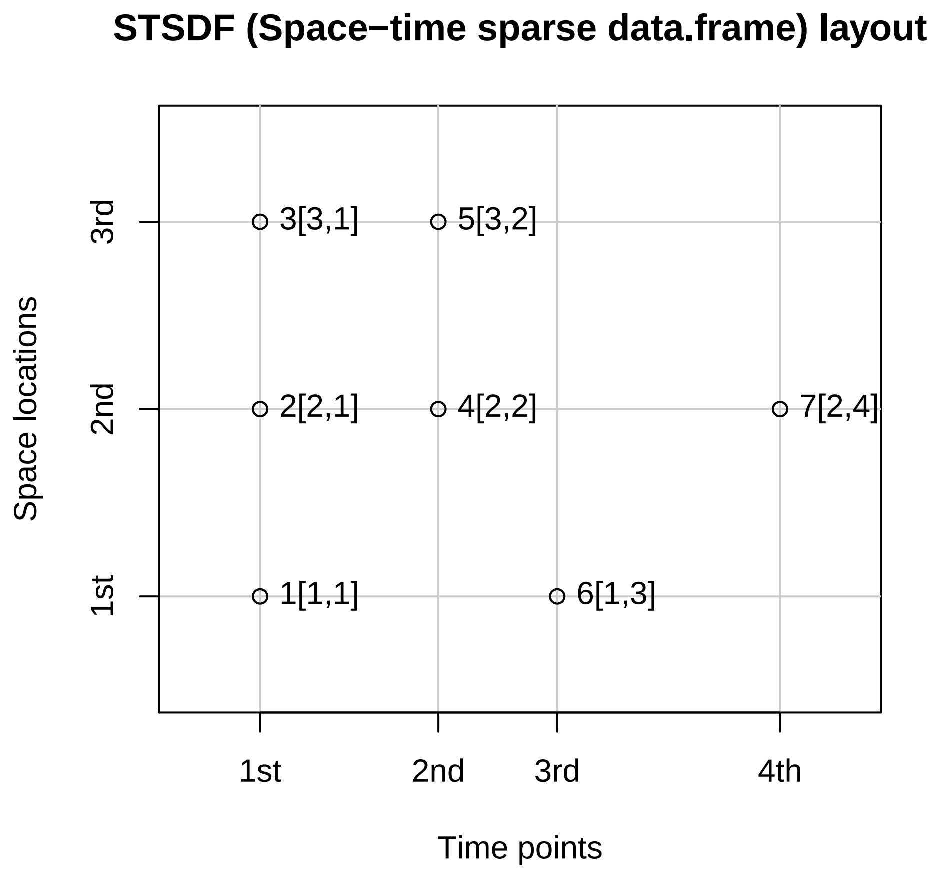


Figure 2: space-time layout of STSDF (STS: ST-Sparse) objects: only the non-missing part of the space-time combinations on a lattice are stored; numbers refer to the ordering of rows in the `data.frame`; an index is kept where e.g. [3,4] refers to the third item in the list of spatial locations and fourth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

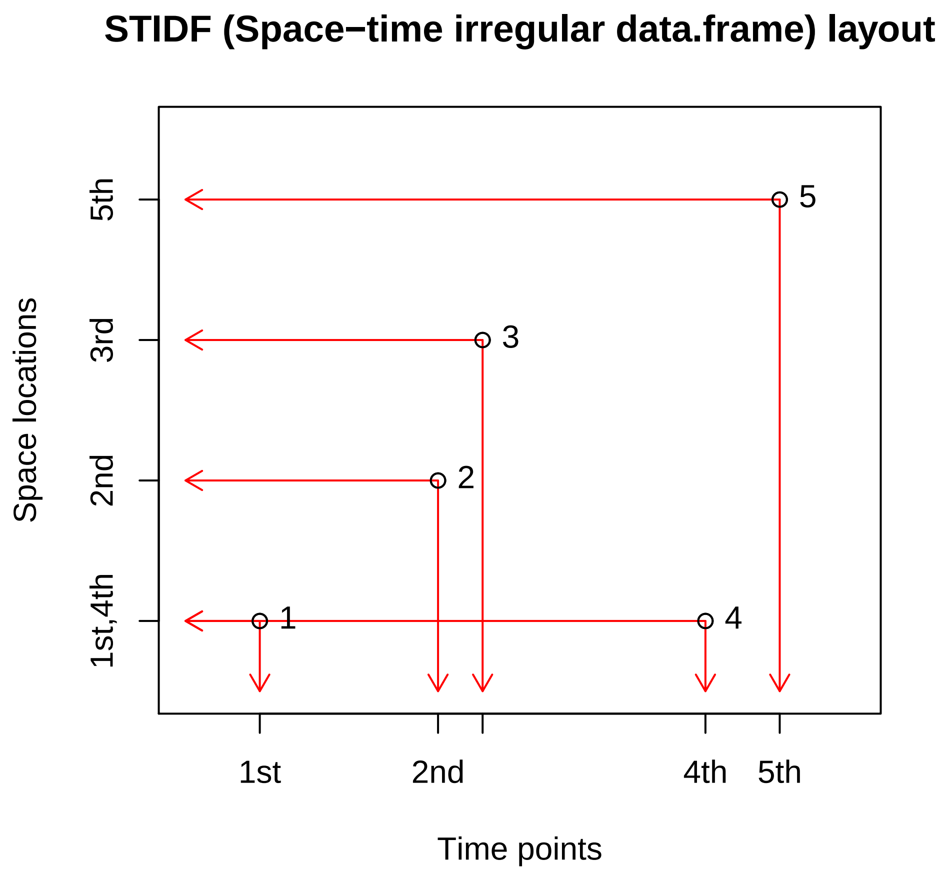


Figure 3: space-time layout of STIDF (STI: ST-Irregular) objects: each observation has its spatial location and time stamp stored; in this example, spatial location 1 is stored twice–observations 1 and 4 having the same location is not registered.

```
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
```

```
> 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
```

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

```
Formal class 'STFDF' [package "spacetime"] with 3 slots
  ..@ data:'data.frame':        12 obs. of  2 variables:
  .. ..$ values: num [1:12] 9.86 21.5 29.8 8.66 19.8 32.6 9.84 18.5 30.8 8.48 ...
  .. ..$ 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:

```
> 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   9.86  ID_1
ID_2  0 1 point2 2010-08-05 10:00:00        1  21.50  ID_2
ID_3  1 1 point3 2010-08-05 10:00:00        1  29.80  ID_3
ID_4  0 0 point1 2010-08-05 11:00:00        2   8.66  ID_4
ID_5  0 1 point2 2010-08-05 11:00:00        2  19.80  ID_5
ID_6  1 1 point3 2010-08-05 11:00:00        2  32.60  ID_6
ID_7  0 0 point1 2010-08-05 12:00:00        3   9.84  ID_7
ID_8  0 1 point2 2010-08-05 12:00:00        3  18.50  ID_8
ID_9  1 1 point3 2010-08-05 12:00:00        3  30.80  ID_9
ID_10 0 0 point1 2010-08-05 13:00:00        4   8.48 ID_10
ID_11 0 1 point2 2010-08-05 13:00:00        4  20.00 ID_11
ID_12 1 1 point3 2010-08-05 13:00:00        4  32.40 ID_12
```

```
> 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   9.86 ID_1
2 0 1 point2 2010-08-05 10:00:00        1  21.50 ID_2
3 1 1 point3 2010-08-05 10:00:00        1  29.80 ID_3
4 0 0 point1 2010-08-05 11:00:00        2   8.66 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()`:

```
> unstack(stfdf)

                    point1 point2 point3
2010-08-05 10:00:00   9.86   21.5   29.8
2010-08-05 11:00:00   8.66   19.8   32.6
2010-08-05 12:00:00   9.84   18.5   30.8
2010-08-05 13:00:00   8.48   20.0   32.4
```

```
> t(unstack(stfdf))

       2010-08-05 10:00:00 2010-08-05 11:00:00 2010-08-05 12:00:00
point1                9.86                8.66                9.84
point2               21.50               19.80               18.50
point3               29.80               32.60               30.80
       2010-08-05 13:00:00
point1                8.48
point2               20.00
point3               32.40
```

```
> 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 xts if we select a single numeric attribute:

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

                    point1 point2 point3
2010-08-05 10:00:00   9.86   21.5   29.8
2010-08-05 11:00:00   8.66   19.8   32.6
2010-08-05 12:00:00   9.84   18.5   30.8
2010-08-05 13:00:00   8.48   20.0   32.4
```

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:

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

```
> x = as(stfdf[,,"values"], "xts")
> # aggregate over space, i.e. over columns:
> apply(x, 1, mean)

2010-08-05 10:00:00 2010-08-05 11:00:00 2010-08-05 12:00:00 2010-08-05 13:00:00
           20.38667            20.35333            19.71333            20.29333
```

```
> # aggregate over time, i.e. over rows:
> apply(x, 2, mean)
```

```
point1 point2 point3
  9.21  19.95  31.40
```

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 function `aggregate`. More information is found in the vignette on this:

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

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:

```
> stfdf[[1]]
```

```
 [1]  9.86 21.50 29.80  8.66 19.80 32.60  9.84 18.50 30.80  8.48 20.00 32.40
```

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

```
 [1]  9.86 21.50 29.80  8.66 19.80 32.60  9.84 18.50 30.80  8.48 20.00 32.40
```

```
> stfdf[["newVal"]] = rnorm(12)
> 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 ID_12
Levels: ID_1 ID_10 ID_11 ID_12 ID_2 ID_3 ID_4 ID_5 ID_6 ID_7 ID_8 ID_9
```

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

```
An object of class "STFDF"
Slot "data":
   values       ID      newVal    NewID
1    9.86  OldIDs1 -0.87758379 NewIDs12
2   21.50  OldIDs2 -0.36521740 NewIDs11
3   29.80  OldIDs3  0.44348575 NewIDs10
4    8.66  OldIDs4  0.02711075  NewIDs9
5   19.80  OldIDs5 -0.94786573  NewIDs8
6   32.60  OldIDs6 -0.07160487  NewIDs7
7    9.84  OldIDs7 -1.10336032  NewIDs6
8   18.50  OldIDs8  1.24989603  NewIDs5
9   30.80  OldIDs9  0.70424619  NewIDs4
10   8.48 OldIDs10  1.41550546  NewIDs3
11  20.00 OldIDs11  2.10203441  NewIDs2
12  32.40 OldIDs12  0.19394861  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. A construct like `a[i,j]` selects row(s) i and column(s) j. In sp, rows were taken as the spatial entities (points, lines, polygons, pixels) and rows as the attributes. This convention was broken for objects of class SpatialGridDataFrame, where `a[i,j,k]` would select the k-th attribute of the spatial grid selection with spatial grid row(s) `i` and column(s) `j`.

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

example:

```
> stfdf[,1] # SpatialPointsDataFrame

  coordinates values      ID     newVal    NewID
1      (0, 0)   9.86 OldIDs1 -0.8775838 NewIDs12
2      (0, 1)  21.50 OldIDs2 -0.3652174 NewIDs11
3      (1, 1)  29.80 OldIDs3  0.4434857 NewIDs10

> stfdf[,,1]

An object of class "STFDF"
Slot "data":
   values
1    9.86
2   21.50
3   29.80
4    8.66
5   19.80
6   32.60
7    9.84
8   18.50
9   30.80
10   8.48
11  20.00
```

```
12  32.40

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
```

```
> stfdf[1,,1] # xts
```

```
                    values
2010-08-05 10:00:00   9.86
2010-08-05 11:00:00   8.66
2010-08-05 12:00:00   9.84
2010-08-05 13:00:00   8.48
```

```
> 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

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

An object of class "STFDF"
Slot "data":
   values
1    9.86
4    8.66
7    9.84
10   8.48

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

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

An object of class "STFDF"
Slot "data":
  values      ID     newVal    NewID
1   9.86 OldIDs1 -0.8775838 NewIDs12
2  21.50 OldIDs2 -0.3652174 NewIDs11
3  29.80 OldIDs3  0.4434857 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 in the data entries (rows), and the locations and times.

5.1 Class definition

```
> 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

```
> 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"
```

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

An object of class "STIDF"
Slot "data":
  values   ID
1   10.1 ID 1
2   19.7 ID 2
3   30.4 ID 3
4   42.0 ID 4
5   50.4 ID 5
6   59.0 ID 6
7   70.6 ID 7
8   78.5 ID 8
9   91.1 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:

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

```
An object of class "STIDF"
Slot "data":
  values   ID
1   10.1 ID 1
2   19.7 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:

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

```
   coordinates values       ID    newVal   NewID
10      (0, 0)   8.48 OldIDs10 1.4155055 NewIDs3
11      (0, 1)  20.00 OldIDs11 2.1020344 NewIDs2
12      (1, 1)  32.40 OldIDs12 0.1939486 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:

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

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

```
> 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.

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

                    values
2010-08-05 10:00:00   9.86
2010-08-05 11:00:00   8.66
2010-08-05 12:00:00   9.84
2010-08-05 13:00:00   8.48
```

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

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

[1] "xts" "zoo"

> 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:

```
> class(stfdf)

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

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

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

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

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

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

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

On our way back, the reverse coercion takes place:

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

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

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

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

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

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

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

```
[1] TRUE
```

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 the time step. The panels share x- and y-axis, no space is lost by separating white space, and a common legend is used. 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.

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 the sea surface temperature data in Cressie and Wikle (2011) is obtained by

```
> demo(CressieWikle)
```

Obviously, such 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 the x- or y-axis needs to be plotted instead of the spatial index (1...n); details are found in the stplot documentation.

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:

```
> library(lattice)
> library(RColorBrewer)
> b = brewer.pal(12, "Set3")
> par.settings = list(superpose.symbol = list(col = b, fill = b),
+          superpose.line = list(col = b),
+          fontsize = list(text=9))
> 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

Time series structures available 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.

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:

```
> .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 some particular ten minute interval, the full specification is needed:

```
> .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).

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` 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.

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

```
> library(maps)
> states.m = map('state', plot=FALSE, fill=TRUE)
> IDs <- sapply(strsplit(states.m$names, ":"), function(x) x[1])
> library(maptools)
> states = map2SpatialPolygons(states.m, IDs=IDs)
> library(plm)
> data(Produc)
> yrs = 1970:1986
> time = as.POSIXct(paste(yrs, "-01-01", sep=""), tz = "GMT")
> # deselect District of Columbia, polygon 8, which is not present in Produc:
> Produc.st = STFDF(states[-8], time, Produc[order(Produc[2], Produc[1]),])
> stplot(Produc.st[,,"unemp"], yrs)
```

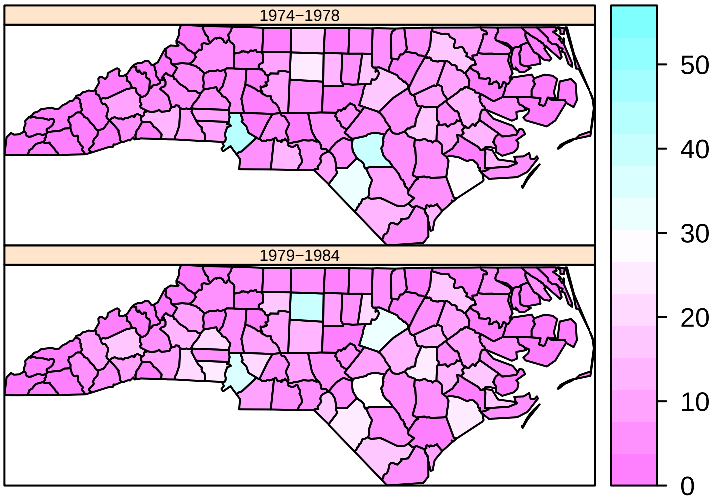


Figure 4: North Carolina sudden infant death syndrome (sids) data

(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 plm routines 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):

```
> zz <- plm(log(gsp) ~ log(pcap) + log(pc) + log(emp) + unemp,
+     data = as.data.frame(Produc.st), index = c("state", "year"))
> 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`. 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 6.

```
> library(gstat)
> data(wind)
> wind.loc$y = as.numeric(char2dms(as.character(wind.loc[["Latitude"]])))
> wind.loc$x = as.numeric(char2dms(as.character(wind.loc[["Longitude"]])))
> coordinates(wind.loc) = ~x + y
> 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:

```
> wind[1:3, ]

  year month day   RPT   VAL   ROS   KIL   SHA  BIR   DUB   CLA   MUL   CLO
1   61     1   1 15.04 14.96 13.17  9.29 13.96 9.87 13.67 10.25 10.83 12.58
2   61     1   2 14.71 16.88 10.83  6.50 12.62 7.67 11.50 10.04  9.79  9.67
3   61     1   3 18.50 16.88 12.33 10.13 11.17 6.17 11.25  8.04  8.50  7.67
    BEL   MAL
1 18.50 15.04
2 17.54 13.83
3 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):

```
> wind$time = ISOdate(wind$year + 1900, wind$month, wind$day)
> wind$jday = as.numeric(format(wind$time, "%j"))
> stations = 4:15
```

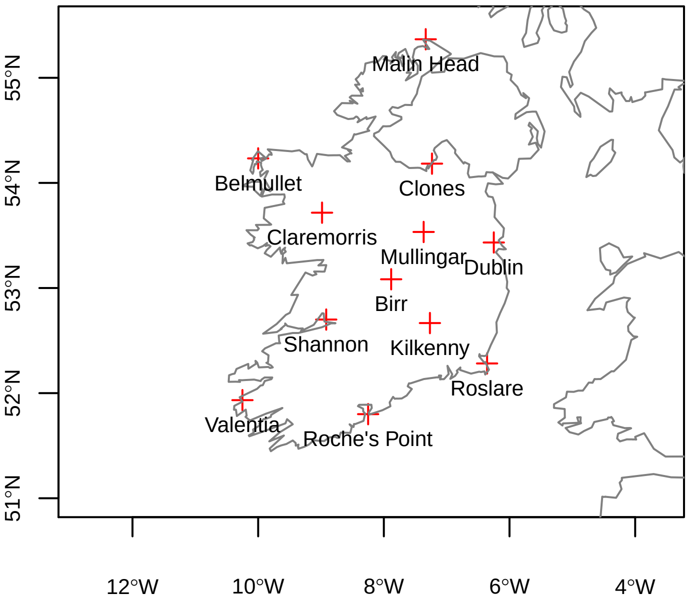


Figure 5: Station locations for Irish wind data

```
> windsqrt = sqrt(0.5148 * wind[stations])
> Jday = 1:366
> daymeans = apply(sapply(split(windsqrt - mean(windsqrt), wind$jday),
+     mean), 2, mean)
> meanwind = lowess(daymeans ~ Jday, f = 0.1)$y[wind$jday]
> 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:

```
> # order locations to order of columns in wind;
> # connect station names to location coordinates
> wind.loc = wind.loc[match(names(wind[4:15]), wind.loc$Code),]
> pts = coordinates(wind.loc[match(names(wind[4:15]), wind.loc$Code),])
> rownames(pts) = wind.loc$Station
> pts = SpatialPoints(pts)
> # convert to utm zone 29, to be able to do interpolation in
> # proper Euclidian (projected) space:
> proj4string(pts) = "+proj=longlat +datum=WGS84"
> library(rgdal)
> utm29 = CRS("+proj=utm +zone=29 +datum=WGS84")
> pts = spTransform(pts, utm29)
> # construct from space-wide table:
> w = stConstruct(velocities, space = list(values = 1:ncol(velocities)),
+         time = wind$time, SpatialObj = pts)
> library(maptools)
> m = map2SpatialLines(
+         map("worldHires", xlim = c(-11,-5.4), ylim = c(51,55.5), plot=F))
> proj4string(m) = "+proj=longlat +datum=WGS84"
> m = spTransform(m, utm29)
> # setup grid
> grd = SpatialPixels(SpatialPoints(makegrid(m, n = 300)),
+         proj4string = proj4string(m))
> # select april 1961:
> w = w[, "1961-04"]
> # 10 prediction time points, evenly spread over this month:
> n = 10
> tgrd = xts(1:n, seq(min(index(w)), max(index(w)), length=n))
> # separable covariance model, exponential with ranges 750 km and 1.5 day:
> v = list(space = vgm(0.6, "Exp", 750000), time = vgm(1, "Exp", 1.5 * 3600 * 24))
> pred = krigeST(sqrt(values)~1, w, STF(grd, tgrd), v)
> 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):

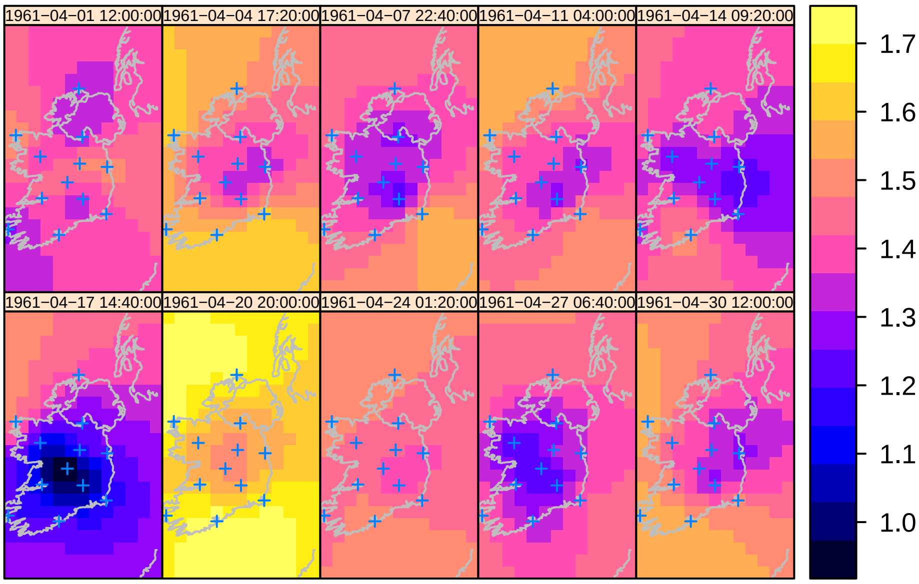


Figure 6: Space-time interpolations of wind (square root transformed, detrended) over Ireland using a separable product covariance model, for 10 time points regularly distributed over the month for which daily data was considered (April, 1961)

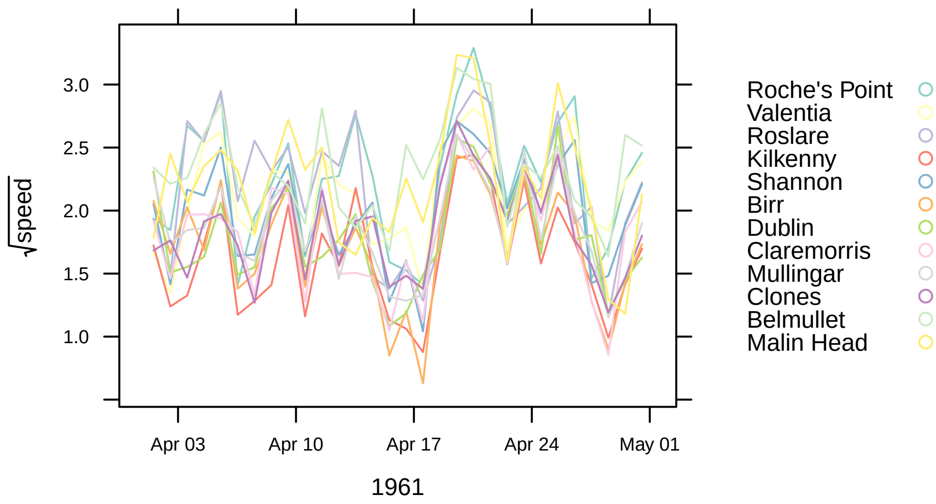


Figure 7: Time series plot of daily wind speed at 12 stations, used for interpolation in figure 6

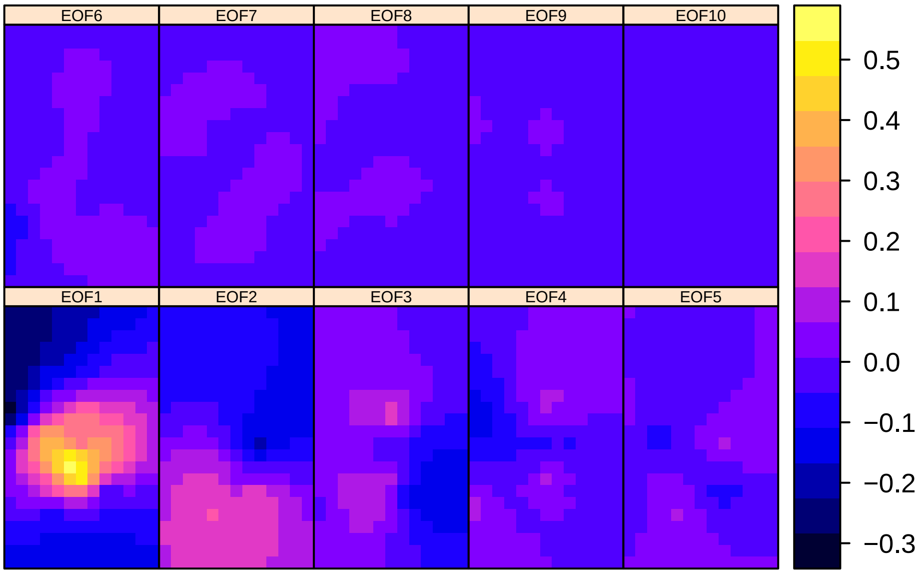


Figure 8: EOFs of space-time interpolations of wind over Ireland (for spatial reference, see figure 6), for the 10 time points at which daily data was chosen above (April, 1961)

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

or in temporal form by:

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

the resulting object is of the appropriate `Spatial` subclass (`SpatialGrid`, `SpatialPolygons` etc.) in the spatial form, or of class `xts` in the temporal form. Figure 8 shows the 10 spatial EOFs obtained from the interpolated wind data of figure 6.

10.5 Conversion from and to trip

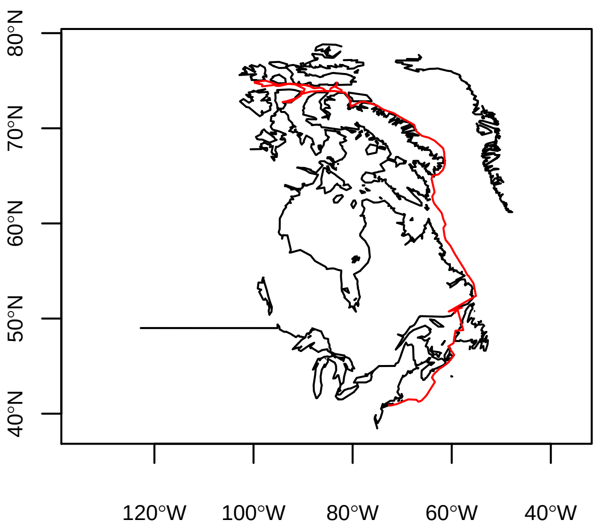
Objects of class `trip` (Sumner, 2010) extend objects of class `SpatialPointsDataFrame` by indicating in which attribute columns time and trip ID are, in slot `TOR.columns`. To not lose this information (in particular, which column contains the IDs), we will extend class `STIDF` to retain this info.

Currently it does assume that time in a trip object is in order, as xts will order it anyhow:

```
> library(diveMove)
> library(trip)
> 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 = ";")
> ringy = subset(locs, id == "ringy" & !is.na(lon) & !is.na(lat))
> coordinates(ringy) = ringy[c("lon", "lat")]
> tr = trip(ringy, c("time", "id"))
> setAs("trip", "STIDFtraj", function(from) {
+     from$burst = from[[from@TOR.columns[2]]]
+     time = from[[from@TOR.columns[1]]]
+     new("STIDFtraj", STIDF(as(from, "SpatialPoints"), time, from@data))
+ })
> x = as(tr, "STIDFtraj")
> m = map2SpatialLines(map("world", xlim = c(-100, -50), ylim = c(40,
+     77), plot = F))
> proj4string(m) = "+proj=longlat +datum=WGS84"
> plot(m, axes = TRUE, cex.axis = 0.7)
> plot(x, add = TRUE, col = "red")
> setAs("STIDFtraj", "trip", function(from) {
+     from$time = index(from@time)
+     trip(SpatialPointsDataFrame(from@sp, from@data), c("time",
+         "burst"))
+ })
> y = as(x, "trip")
> y$burst = NULL
> all.equal(y, tr, check.attributes = FALSE)

[1] TRUE
```



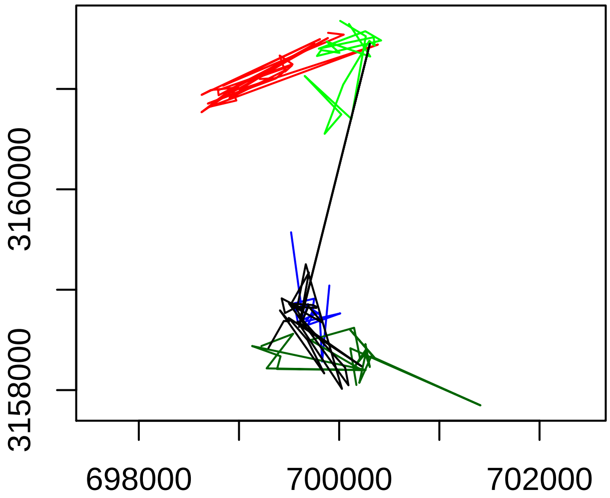
10.6 Trajectory data: ltraj in adehabitatLT

Trajectory objects of class `ltraj` 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:

```
> library(adehabitatLT)
> # from: adehabitat/demo/managltraj.r
> # demo(managltraj)
> data(puechabonsp)
> # locations:
> locs = puechabonsp$relocs
> xy = coordinates(locs)
> ### Conversion of the date to the format POSIX
> da = as.character(locs$Date)
> da = as.POSIXct(strptime(as.character(locs$Date),"%y%m%d"), tz = "GMT")
> ## object of class "ltraj"
> ltr = as.ltraj(xy, da, id = locs$Name)
> foo = function(dt) dt > 100*3600*24
> ## The function foo returns TRUE if dt is longer than 100 days
> ## We use it to cut ltr:
> l2 = cutltraj(ltr, "foo(dt)", nextr = TRUE)
> stidfTrj = as(l2, "STIDFtraj")
> ltr0 = as(stidfTrj, "ltraj")
> all.equal(l2, ltr0, check.attributes = FALSE)
```

```
[1] TRUE
```

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



A more complicated plot is shown in figure 9, obtained by the command

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

the output of which is shown in figure 9.

10.7 Country shapes in cshapes

The `cshapes` package contains a GIS dataset of country 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:

```
> library(cshapes)
> cs = cshp()
> 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, 2008), 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.

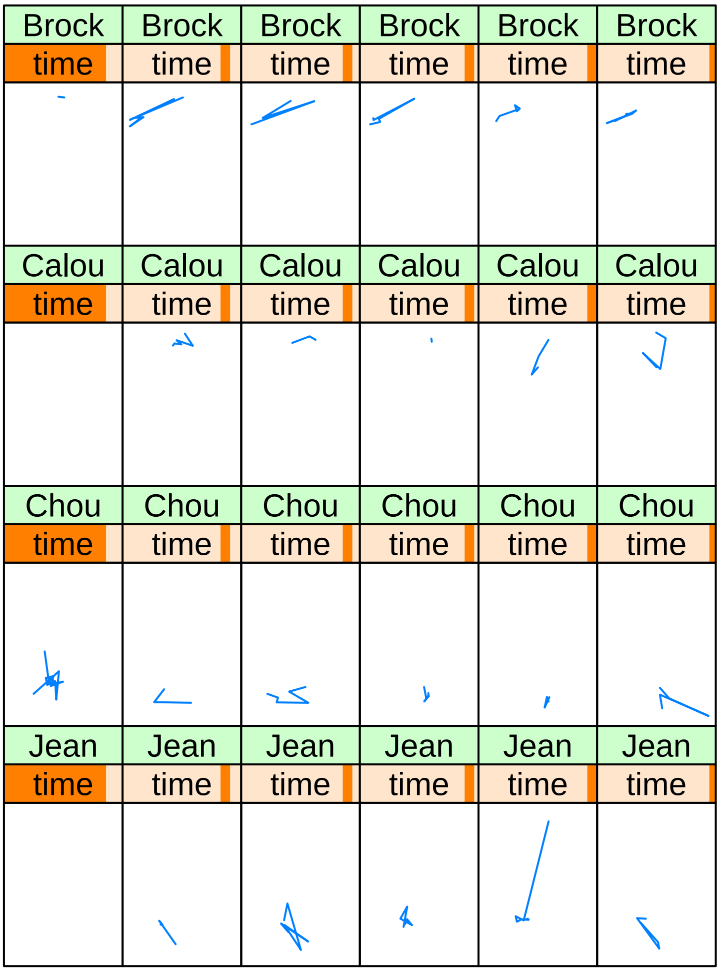


Figure 9: trajectories, by id (rows) and time (columns)

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

```
> 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`):

```
> t = as.POSIXct(strptime(paste(cs$COWSYEAR, cs$COWSMONTH, cs$COWSDAY,
+     sep = "-"), "%Y-%m-%d"), tz = "GMT")
> st = STIDF(geometry(cs), t, as.data.frame(cs))
> pt = SpatialPoints(cbind(7, 52), CRS(proj4string(cs)))
> as.data.frame(st[pt, , 1:5])
```

```
        V1       V2 sp.ID       time timedata               CNTRY_NAME     AREA
1  9.41437 50.57623   188 1955-05-05      188 Germany Federal Republic 247366.4
2 10.38084 51.09070   187 1990-10-03      187                  Germany 356451.5
  CAPNAME CAPLONG   CAPLAT
1    Bonn     7.1 50.73333
2  Berlin    13.4 52.51667
```

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 contributed in many useful discussions.

References

Baltagi B (2001). Econometric Analysis of Panel Data. John Wiley and Sons, 3rd edition. (see http://www.wiley.com/legacy/wileychi/baltagi/)

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

Calenge, C., S. Dray, M. Royer-Carenzi (2008). The concept of animals' trajectories from a data analysis perspective. Ecological informatics 4, 34-41.

Cressie, N., C. Wikle, 2011. Statistics for spatio-temporal data. Wiley, NY.

Croissant Y., G. Millo, 2008. Panel Data Econometrics in R: The plm Package. Journal of Statistical Software, 27(2). http://www.jstatsoft.org/v27/i02/.

Galton, A. (2004). Fields and Objects in Space, Time and Space-time. Spatial cognition and computation 4(1).

Güting, R.H., M. Schneider, 2005. Moving Objects Databases. Morgan Kaufmann Publishers.

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

Schabenberger, O., and Gotway, C.A., 2004. Statistical methods for spatial data analysis. Boca Raton: Chapman and Hall.

Sumner, M., 2010. The tag location problem. Unpublished PhD thesis, Institute of Marine and Antarctic Studies University of Tasmania, September 2010.

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

Gleditsch, Kristian S., Michael D. Ward. 1999. Interstate System Membership: A Revised List of the Independent States since 1816. International Interactions 25: 393-413. Online, `http://privatewww.essex.ac.uk/~ksg/statelist.html`

Grothendieck, G. and T. Petzoldt. R Help Desk: Date and time classes in R. R News, 4(1):29-32, June 2004

Ripley, B.D., and K. Hornik. Date-time classes. R News, 1(2):8-11, June 2001.