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

ifgi
Institute for Geoinformatics
University of Münster

Edzer Pebesma

October 20, 2010

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 the temporal classes in 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 which methods are useful and feasible for the classes implemented. It tries to reuse existing infrastructure (classes, methods, functions) that is 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 **2**

2 Space-time layouts **3**
2.1 Full space-time grid 3
2.2 Partial space-time grid 5
2.3 Sparse space-time `data.frame` 5

3 Spatio-temporal full grid `data.frames` (`STFDF`) **5**
3.1 Class definition 5
3.2 Coercion to `data.frame` 8
3.3 Coercion to `xts` 9
3.4 Attribute retrieval and replacement: `[[` and `$` 10
3.5 Selection with `[` 11

4 Space-time partial `data.frames` (`STPDF`) **13**
4.1 Class definition 14

5 Spatio-temporal sparse `data.frames` (`STSDF`) **14**
5.1 Class definition 14
5.2 Methods 15

6 Methods: obtaining a snapshot or history **16**

7 Coercion **17**

8 Spatial footprint or support, time intervals **18**
8.1 Time periods 18
8.2 Spatial Points 18
8.3 Spatial Polygons 19

9 Worked examples **20**
9.1 Interpolating Irish wind 20
9.2 Conversion from and to trip 22
9.3 Trajectory data: ltraj in adehabitat 25

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, and GPS tracks for people or animals.

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 tries to partially fill this gap.

A possible reason why data are often analysed conditionally is that they are often either overly abundant in space, or in time, and relatively sparse in the other. Satellite imagery is typically very abundant in space, giving lots of detail in high resolution for large areas, but much less abundant in time. Also, repeated images over time may suffer from problems like difference in light conditions, errors in georeferencing resulting in spatial mismatch, and changes in obscured areas due to changed cloud coverage. On the other hand, data from fixed sensors give often very detailed signals over time, allowing for elaborate modelling, but relatively sparse 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, and less so on the temporal resolution with which the sensors register signals.

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 because it has nice tools for reorganizing time and a very flexible syntax to select time periods that adheres ISO 8601[1]. We do not use the `xts` objects to store 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.

2 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 will be implemented, along with convenience methods and coercion methods to get from one to the other.

A full space-time grid[2] 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.

A partial 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.

Sparse space-time data are those where time and space points of measured values can have arbitrary organization: for each measured value the spatial location and time point is stored. This is equivalent to a partial grid where the index for observation k is $[k, k]$, and hence can be dropped. For these objects, $n = m$ and equals the number of records. The next subsections will illustrate these three classes.

2.1 Full space-time grid

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`.

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

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

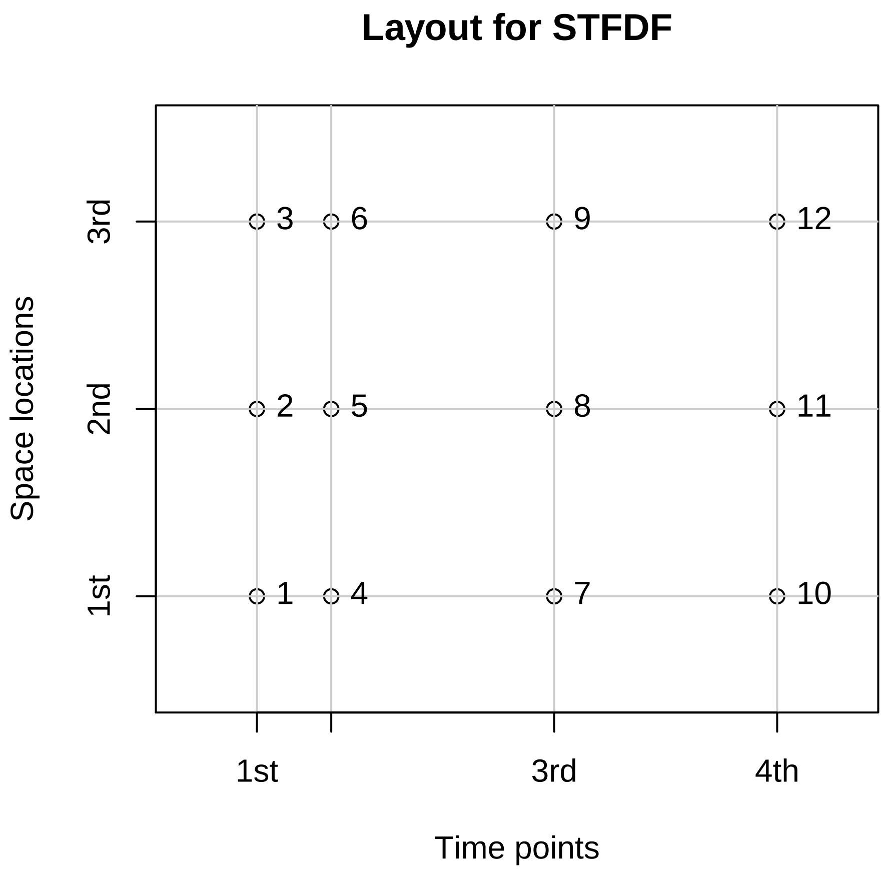


Figure 1: space-time layout of STFDF (F: Full) objects: all space-time combinations are stored; numbers refer to the ordering of rows in the `data.frame` with measured values: time is kept ordered, space cycles first

2.2 Partial space-time grid

Partial space-time grids (figure 2) have space and time points layed out on a grid, but not all grid nodes are stored and an index is kept that relates the values to the grid nodes: $[i, j]$ refers to spatial location i and time point j.

2.3 Sparse space-time `data.frame`

Space-time sparse `data.frame`s (STSDF, figure 3) simply store for each value the spatial location and time point, in time order.

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

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

3.1 Class definition

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

Class "ST" [package "spacetime"]

Slots:

Name:       sp    time
Class: Spatial     xts

Known Subclasses:
Class "STP", directly
Class "STS", directly
Class "STF", directly
Class "STPDF", by class "STP", distance 2
Class "STSDF", by class "STS", distance 2
Class "STFDF", by class "STF", distance 2
Class "STSDFtraj", by class "STSDF", 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
```

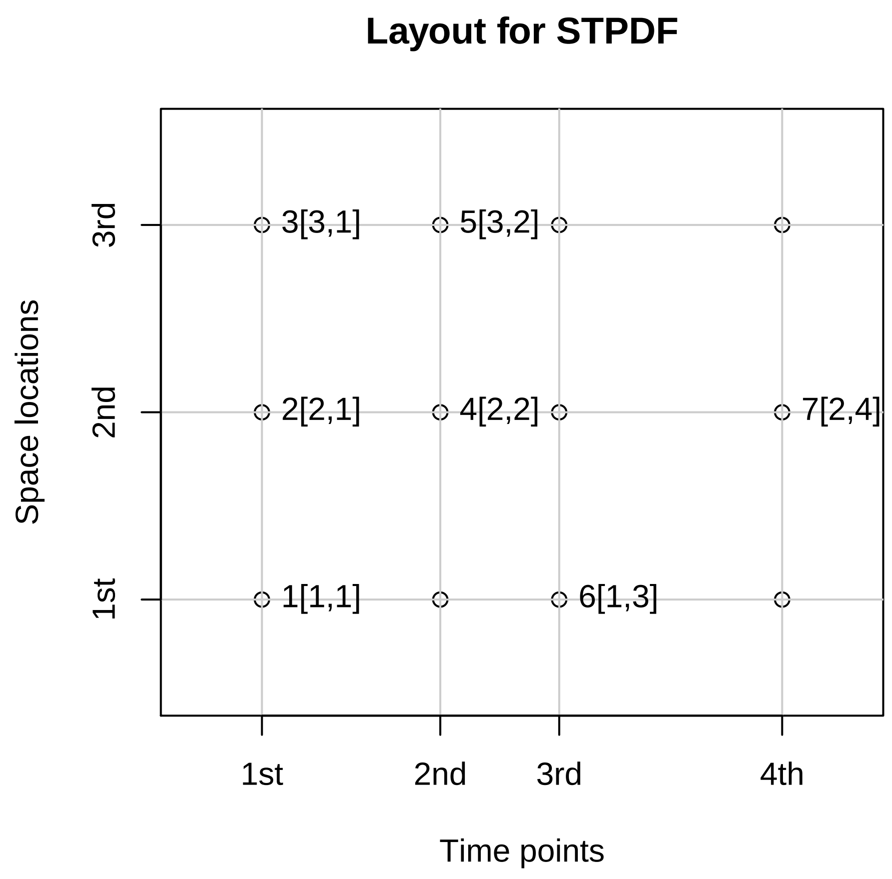


Figure 2: space-time layout of STPDF (P: partial) objects: part of the space-time combinations are stored; numbers refer to the ordering of rows in the `data.frame`; an index is kept where [3,4] refers to the third item in the list of spatial locations and fourth item in the list of temporal points.

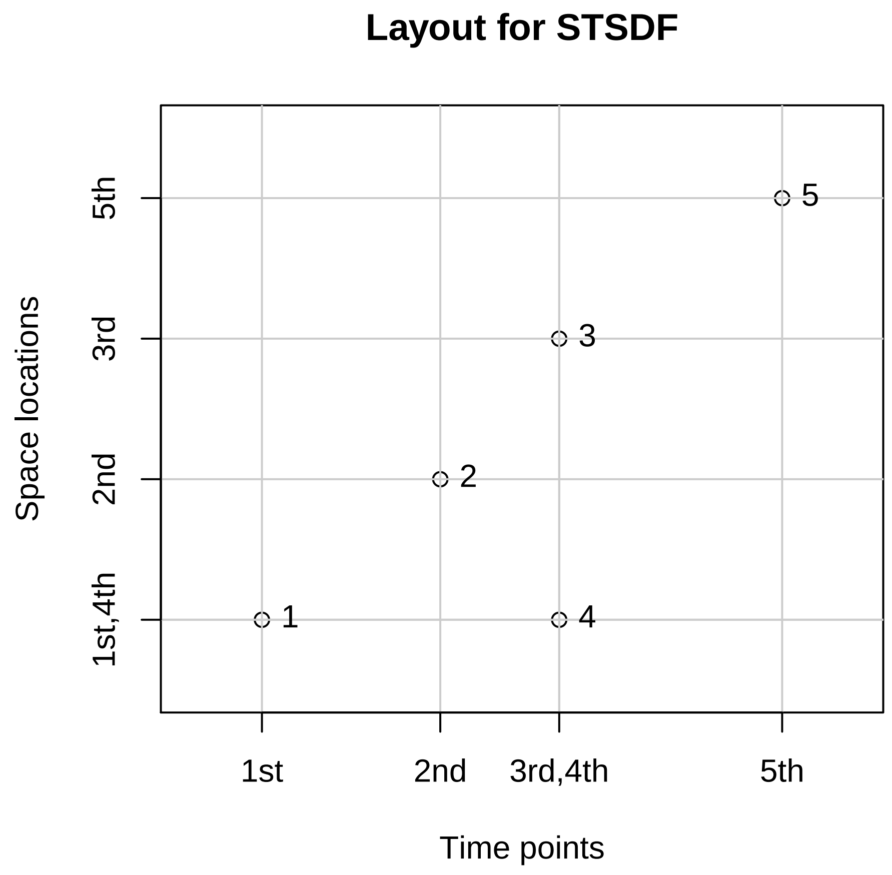


Figure 3: space-time layout of STSDF (S: Sparse) objects: each observation has its spatial location and time stamp stored; in this example, time point 3 and spatial location 1 are duplicated, so they appear twice.

```
> 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 = xts(1:4, as.POSIXct("2010-08-05")+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] 10.1 21.4 30.6 10.1 18.3 28.9 11.2 19.7 31.6 9.69 ...
  .. ..$ 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:
  xts Attributes:
 NULL
```

3.2 Coercion to `data.frame`

The following coercion function creates a `data.frame` using both the S3 (to set row.names) and 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)
```

```
      X1 X2  sp.ID                time values    ID
ID_1   0  0 point1 2010-08-05 10:00:00  10.10  ID_1
ID_2   0  1 point2 2010-08-05 10:00:00  21.40  ID_2
ID_3   1  1 point3 2010-08-05 10:00:00  30.60  ID_3
ID_4   0  0 point1 2010-08-05 11:00:00  10.10  ID_4
ID_5   0  1 point2 2010-08-05 11:00:00  18.30  ID_5
ID_6   1  1 point3 2010-08-05 11:00:00  28.90  ID_6
ID_7   0  0 point1 2010-08-05 12:00:00  11.20  ID_7
ID_8   0  1 point2 2010-08-05 12:00:00  19.70  ID_8
ID_9   1  1 point3 2010-08-05 12:00:00  31.60  ID_9
ID_10  0  0 point1 2010-08-05 13:00:00   9.69 ID_10
```

```
ID_11  0  1 point2 2010-08-05 13:00:00  21.10 ID_11
ID_12  1  1 point3 2010-08-05 13:00:00  30.30 ID_12
```

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

```
  X1 X2  sp.ID                time values   ID
1  0  0 point1 2010-08-05 10:00:00   10.1 ID_1
2  0  1 point2 2010-08-05 10:00:00   21.4 ID_2
3  1  1 point3 2010-08-05 10:00:00   30.6 ID_3
4  0  0 point1 2010-08-05 11:00:00   10.1 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.

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  10.10   21.4   30.6
2010-08-05 11:00:00  10.10   18.3   28.9
2010-08-05 12:00:00  11.20   19.7   31.6
2010-08-05 13:00:00   9.69   21.1   30.3
```

```
> t(unstack(stfdf))
```

```
       2010-08-05 10:00:00 2010-08-05 11:00:00 2010-08-05 12:00:00
point1                10.1                10.1                11.2
point2                21.4                18.3                19.7
point3                30.6                28.9                31.6
       2010-08-05 13:00:00
point1                9.69
point2               21.10
point3               30.30
```

```
> 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
```

3.3 Coercion to `xts`

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

```
> as(stfdf, "xts")
```

```
                    point1 point2 point3
2010-08-05 10:00:00  10.10   21.4   30.6
2010-08-05 11:00:00  10.10   18.3   28.9
2010-08-05 12:00:00  11.20   19.7   31.6
2010-08-05 13:00:00   9.69   21.1   30.3
```

3.4 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] 10.10 21.40 30.60 10.10 18.30 28.90 11.20 19.70 31.60  9.69 21.10 30.30

> stfdf[["values"]]

 [1] 10.10 21.40 30.60 10.10 18.30 28.90 11.20 19.70 31.60  9.69 21.10 30.30

> 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   10.10  OldIDs1 -0.3222119 NewIDs12
2   21.40  OldIDs2 -1.6785938 NewIDs11
3   30.60  OldIDs3 -0.8655378 NewIDs10
4   10.10  OldIDs4 -0.1094041  NewIDs9
5   18.30  OldIDs5  1.0978210  NewIDs8
6   28.90  OldIDs6 -1.2534154  NewIDs7
7   11.20  OldIDs7 -1.8644769  NewIDs6
8   19.70  OldIDs8 -0.8257209  NewIDs5
9   31.60  OldIDs9 -0.4840448  NewIDs4
10   9.69 OldIDs10  0.6564169  NewIDs3
11  21.10 OldIDs11  0.2000710  NewIDs2
12  30.30 OldIDs12  2.5221398  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
```

3.5 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)   10.1 OldIDs1 -0.3222119 NewIDs12
2      (0, 1)   21.4 OldIDs2 -1.6785938 NewIDs11
3      (1, 1)   30.6 OldIDs3 -0.8655378 NewIDs10

> stfdf[,,1]

An object of class "STFDF"
Slot "data":
   values
1   10.10
2   21.40
3   30.60
4   10.10
5   18.30
6   28.90
7   11.20
8   19.70
9   31.60
10   9.69
11  21.10
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
```

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

```
                    values
2010-08-05 10:00:00  10.10
2010-08-05 11:00:00  10.10
2010-08-05 12:00:00  11.20
2010-08-05 13:00:00   9.69
```

```
> 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   10.10
4   10.10
7   11.20
10   9.69

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   10.1 OldIDs1 -0.3222119 NewIDs12
2   21.4 OldIDs2 -1.6785938 NewIDs11
3   30.6 OldIDs3 -0.8655378 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 6.

4 Space-time partial `data.frames` (`STPDF`)

Space-time partial `data.frames` 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.

4.1 Class definition

```
> showClass("STPDF")

Class "STPDF" [package "spacetime"]

Slots:

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

Extends:
Class "STP", directly
Class "ST", by class "STP", 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 `data[i,]` corresponds to location j and time k.

5 Spatio-temporal sparse `data.frames` (`STSDF`)

Space-time sparse `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).

5.1 Class definition

```
> showClass("STSDF")

Class "STSDF" [package "spacetime"]

Slots:

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

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

Known Subclasses: "STSDFtraj"
```

```
> sp = expand.grid(x = 1:3, y = 1:3)
> row.names(sp) = paste("point", 1:nrow(sp), sep="")
> sp = SpatialPoints(sp)
> time = xts(1:9, as.POSIXct("2010-08-05")+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)
> stsdf = STSDF(sp, time, mydata)
> stsdf
```

```
An object of class "STSDF"
Slot "data":
  values   ID
1   11.0 ID 1
2   19.9 ID 2
3   31.1 ID 3
4   41.0 ID 4
5   49.4 ID 5
6   59.8 ID 6
7   71.0 ID 7
8   82.3 ID 8
9   90.5 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
```

5.2 Methods

Selection takes place with the `[` method:

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

```
An object of class "STSDF"
Slot "data":
  values   ID
1   11.0 ID 1
2   19.9 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
```

6 Methods: obtaining a snapshot or history

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
7      (0, 0)   11.2 OldIDs7 -1.8644769 NewIDs6
8      (0, 1)   19.7 OldIDs8 -0.8257209 NewIDs5
9      (1, 1)   31.6 OldIDs9 -0.4840448 NewIDs4
```

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

```
> 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  10.10
2010-08-05 11:00:00  10.10
2010-08-05 12:00:00  11.20
2010-08-05 13:00:00   9.69
```

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

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

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

Note that for objects of class `STSDF`, `drop = TRUE` is not (yet) implemented as it is not clear to which classe a single record should be reduced; for sets of records, further processing is needed to find out whether a single point in time or a single spatial location is retrieved.

7 Coercion

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

```
> class(stfdf)

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

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

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

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

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

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

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

On our way back, the reverse coercion takes place:

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

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

> class(as(as(x, "STPDF"), "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] FALSE
```

8 Spatial footprint or support, time intervals

8.1 Time periods

Time series typically store for each record a time stamp, not a time interval. The implicit assumption of time 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, where ”high” and ”low” aggregated (minimum, maximum taken 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 it seems that this syntax lets one define, unambiguously, yearly, monthly, daily, hourly or minute intervals, but not 10- or 30-minute intervals; for ten minutes, 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"
```

8.2 Spatial Points

All examples above work with spatial points, i.e. data having a point support. The assumption of data having points support is implicit.

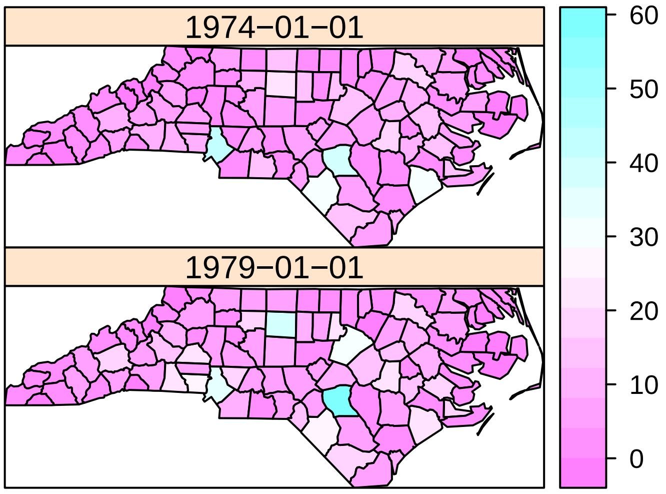


Figure 4: North Carolina suddan infant death syndrome data

8.3 Spatial Polygons

As an example, the North Carolina SIDS data in package `maptools` will be used; they are sparse in time (2 instances), but have polygons in space. See figure 4 for the plot generated by:

```
> library(maptools)

        Note: polygon geometry computations in maptools
         depend on the package gpclib, which has a
         restricted licence. It is disabled by default;
         to enable gpclib, type gpclibPermit()

Checking rgeos availability as gpclib substitute:
FALSE

> 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 = xts(1:2, as.POSIXct(strptime(c("1974-01-01", "1979-01-01"),
+     "%Y-%m-%d")))
> nct = STFDF(sp = as(nc, "SpatialPolygons"), time = time, data = data)
> stplot(nct[, , "SID"], as.table = TRUE)
```

9 Worked examples

9.1 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 largely reproduces 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 8.3, for this data space is sparse and time is rich, and so the data in `data.frame wind` come in 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:

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

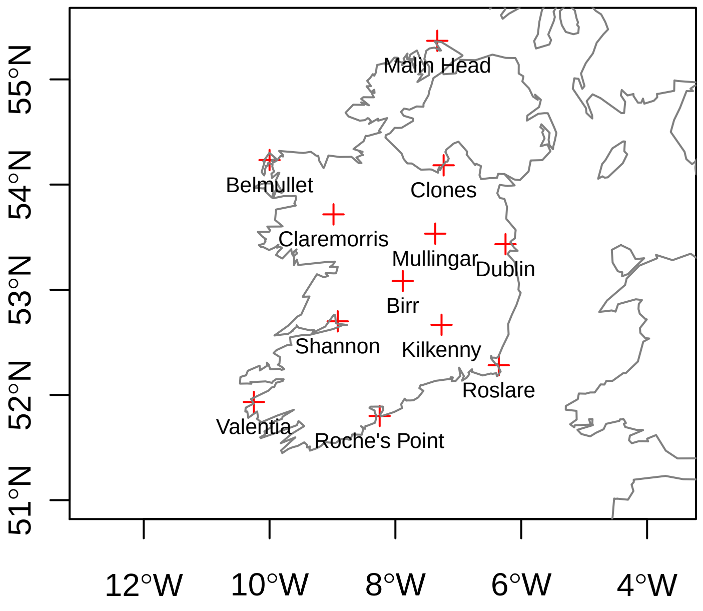


Figure 5: Station locations for Irish wind data

Next, we will match the wind data to its location, and convert the long/lat coordinates and country boundary to the appropriate UTM zone:

```
> 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$Station
> pts = SpatialPoints(pts)
> # convert to utm zone 29, to be able to do interpolation in
> # proper Euclidian (projected) space:
> proj4string(pts) = "+proj=longlat +datum=WGS84"
> library(rgdal)
> utm29 = CRS("+proj=utm +zone=29 +datum=WGS84")
> t = xts(1:nrow(wind), wind$time)
> pts = spTransform(pts, utm29)
> # note the t() in:
> w = STFDF(pts, t, data.frame(values = as.vector(t(velocities))))
> 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`.

9.2 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 `STSDF` 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(system.file(file.path("data", "sealLocs.csv"),
+     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))
```

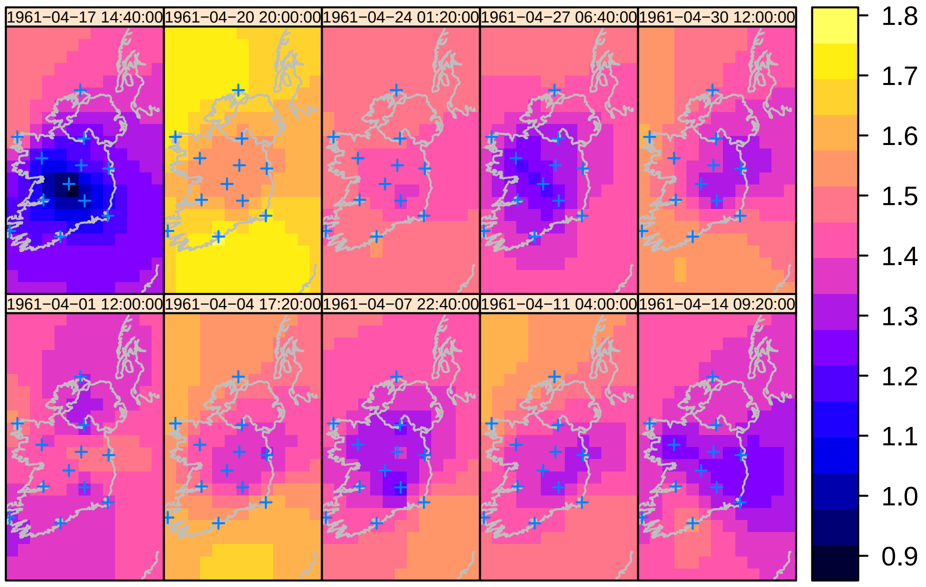


Figure 6: Space-time interpolations of wind (square root transformed, de-trended) over Ireland using a separable product covariance model, for 10 time points regularly distributed over the month for which daily data was considered (April, 1961)

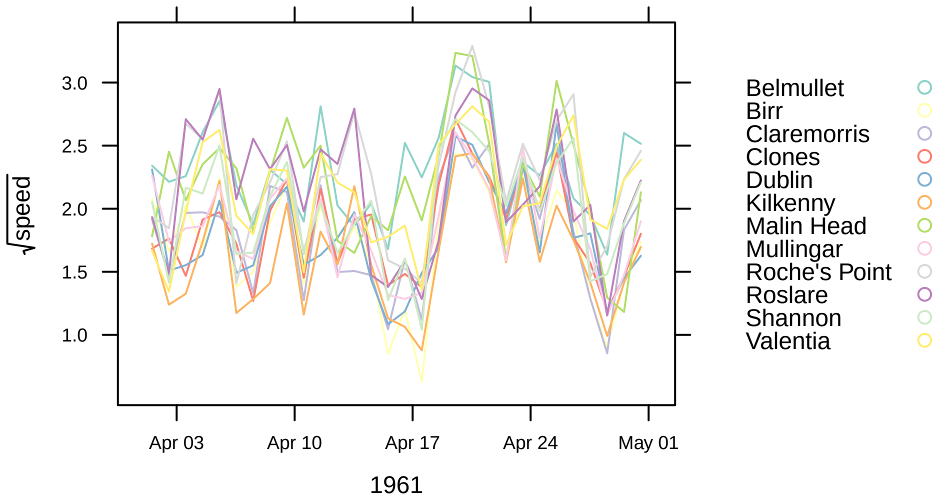
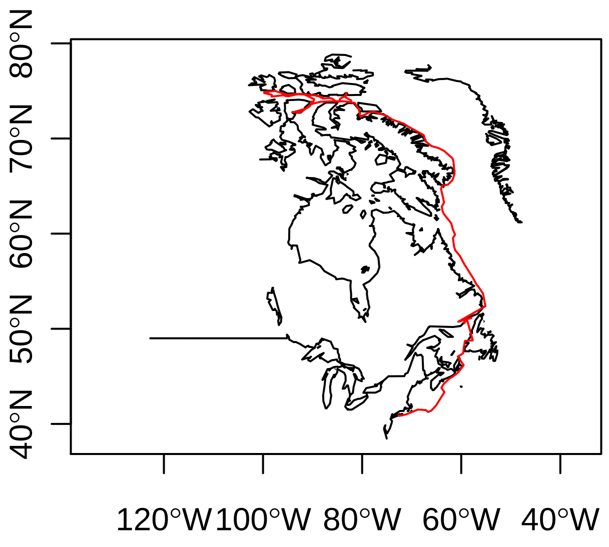


Figure 7: Time series plot for some randomly selected grid cells of figure 6

```
> coordinates(ringy) = ringy[c("lon", "lat")]
> tr = trip(ringy, c("time", "id"))
> setAs("trip", "STSDFtraj", function(from) {
+     from$burst = from[[from@TOR.columns[2]]]
+     time = from[[from@TOR.columns[1]]]
+     new("STSDFtraj", STSDF(as(from, "SpatialPoints"), time, from@data))
+ })
> x = as(tr, "STSDFtraj")
> m = map2SpatialLines(map("world", xlim = c(-100, -50), ylim = c(40,
+     77), plot = F))
> proj4string(m) = "+proj=longlat +datum=WGS84"
> plot(m, axes = TRUE)
> plot(x, add = TRUE, line.col = "red")
> setAs("STSDFtraj", "trip", function(from) {
+     from$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
```



9.3 Trajectory data: ltraj in adehabitat

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 STSDF 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.

```
> library(adehabitat)
```

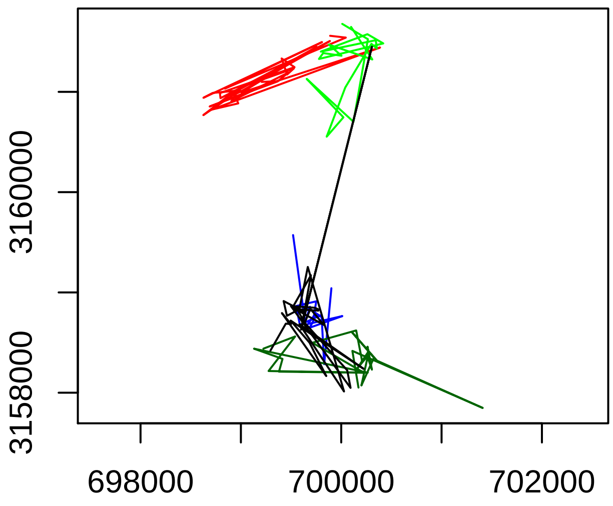
```
This package requires ade4 to be installed

Type:
demo(rastermaps) for demonstration of raster map analysis
demo(homerange) for demonstration of home-range estimation
demo(managltraj) for demonstration of animals trajectory management
demo(analysisltraj) for demonstration of animals trajectory analysis
demo(nichehs) for demonstration of niche/habitat selection analysis
```

```
> # from: adehabitat/demo/managltraj.r
> # demo(managltraj)
> data(puechabon)
> # locations:
> locs = puechabon$locs
> xy = locs[,c("X","Y")]
> ### Conversion of the date to the format POSIX
> da = as.character(locs$Date)
> da = as.POSIXct(strptime(as.character(locs$Date),"%y%m%d"))
> ## object of class "ltraj"
> ltr = as.ltraj(xy, da, id = locs$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)
> ltr.stsdf = as(l2, "STSDFtraj")
> # ltr.stsdf[1:10,]
> ltr0 = as(ltr.stsdf, "ltraj")
> all.equal(l2, ltr0, check.attributes = FALSE)
```

```
[1] TRUE
```

```
> plot(ltr.stsdf, line.col = c("red", "green", "blue", "darkgreen", "black"),
+         axes=TRUE)
```



Acknowledgements

Michael Sumner provided helpful comments on the trip example.

References

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.

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).

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.