

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



ifgi
Institute for Geoinformatics
University of Münster

Edzer Pebesma

September 30, 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 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 grids	3
2.2	Partial space-time grids	3
2.3	Sparse space-time grids	3
3	Spatio-temporal full grid data frames (STFDF)	7
3.1	Class definition	7
3.2	Coercion to data.frame	8
3.3	Coercion to xts	9
3.4	Attribute retrieval and replacement: [[and \$	9
3.5	Selection with [.	10

4	Space-time partial data frames (STPDF)	13
4.1	Class definition	13
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
9	Worked examples	18
9.1	Interpolating Iris wind	18
9.2	Tracking data: trip and ltraj objects	20
9.3	Conversion from and to trip	22
9.4	Trajectory data: ltraj in adehabitat	23

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 in *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 ground 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.

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 grid¹ of observations for spatial location (points, lines, polygons, grid cells) $s_i, i = 1, \dots, n$ and observation time $t_j, j = 1, \dots, m$ is obtained when the full set of $n \times m$ set of observations z_k is stored, with $k = 1, \dots, nm$. We choose to cycle spatial locations first, so observation k corresponds to location $((k - 1) \sim \text{mod} \sim n) + 1$ and with time moment $((k - 1)/n) + 1$, with $/$ the integer division.

A partial grid has the same general layout, with measurements layed 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 value k is $[k, k]$, and hence can be dropped. The next subsections will illustrate these three classes.

2.1 Full space-time grids

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.

2.2 Partial space-time grids

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 grids

Space-time sparse data frames (STSDf, figure 3) simply store for each value the spatial location and time point.

¹note that neither locations nor time points need to be laid out in some regular sequence

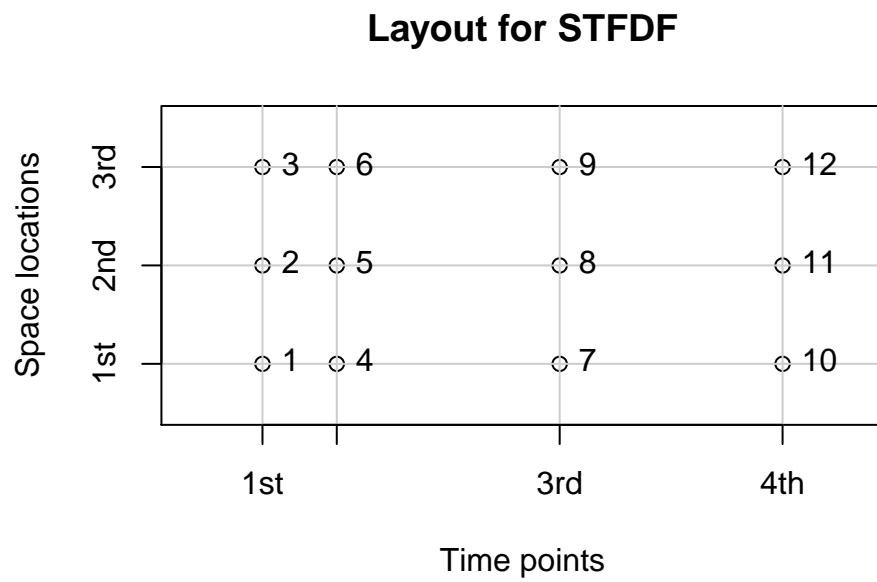


Figure 1: space-time layout of STFDF (F: Full) objects: all space-time combinations are stored; numbers refer to the ordering of rows in the data.frame with measured values: time is kept ordered, space cycles first

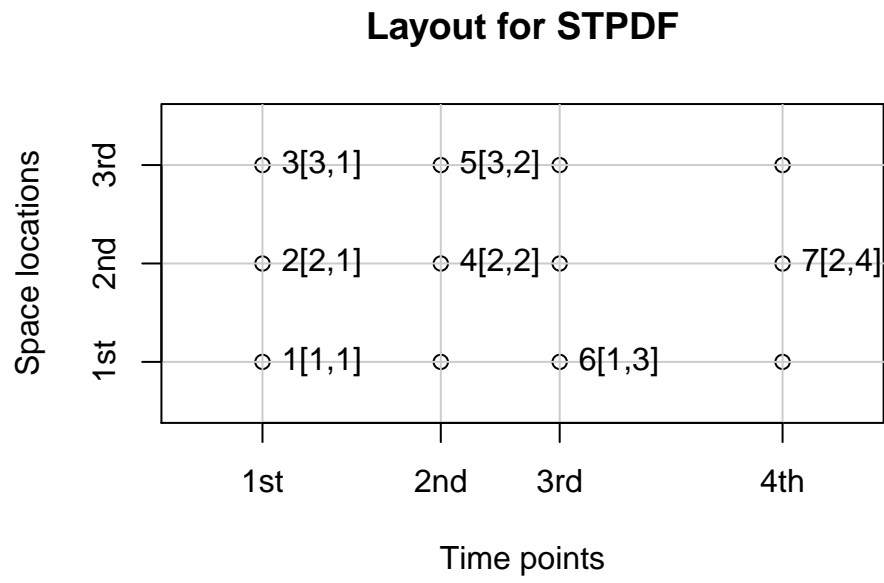


Figure 2: space-time layout of STPDF (P: partial) objects: part of the space-time combinations are stored; numbers refer to the ordering of rows in the data.frame; an index is kept where [3,4] refers to the third item in the list of spatial locations and fourth item in the list of temporal points.

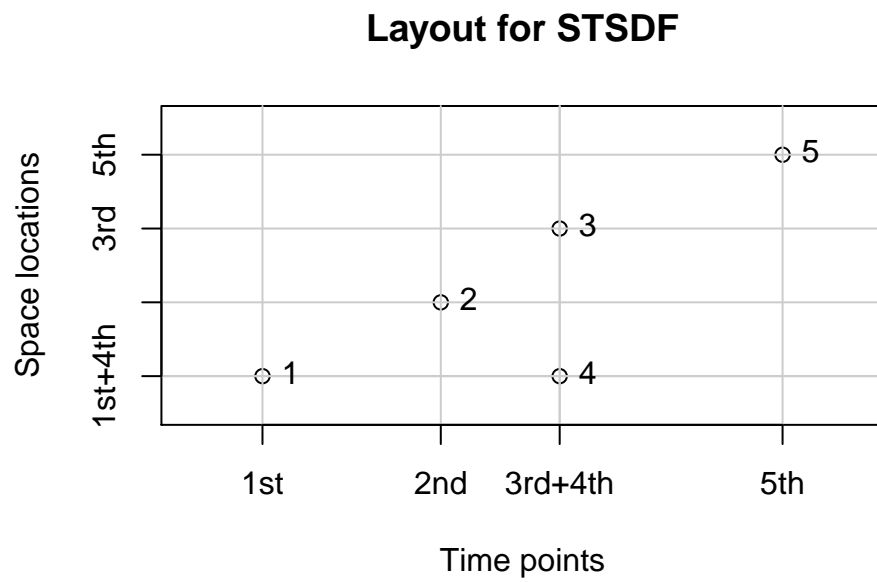


Figure 3: space-time layout of STSDF (S: Sparse) objects: each observation has its spatial location and time stamp stored; In the example, time point 3 and spatial location 1 are duplicated.

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

> 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 = 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))
> 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] 9.38 19.8 27.9 9.76 20.5 27.1 11.3 19.5 29.8 10 ...
.. ..$ 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
.. ..@ - attr(*, "values")=List of 3
.. ..@ - attr(*, "point1")=chr [1:3] "point1" "point2" "point3"
.. ..@ - attr(*, "point2")=chr [1:2] "x" "y"
.. ..@ - attr(*, "point3")=chr [1:2] "x" "y"
.. ..@ - attr(*, "bbox")=List of 2
.. ..@ - attr(*, "dimnames")=List of 2
.. ..@ - attr(*, "values")=List of 3
.. ..@ - attr(*, "point1")=chr [1:2] "x" "y"
.. ..@ - attr(*, "point2")=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: [POSIXt,POSIXct] 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()`) way. It gives data in the long format, meaning that time and space are replicated appropriately:

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

	values	ID	X1	X2	sp.ID	time
1	9.38	ID 1	0	0	point1	2010-08-05 10:00:00
2	19.80	ID 2	0	1	point2	2010-08-05 10:00:00
3	27.90	ID 3	1	1	point3	2010-08-05 10:00:00
4	9.76	ID 4	0	0	point1	2010-08-05 11:00:00
5	20.50	ID 5	0	1	point2	2010-08-05 11:00:00
6	27.10	ID 6	1	1	point3	2010-08-05 11:00:00
7	11.30	ID 7	0	0	point1	2010-08-05 12:00:00
8	19.50	ID 8	0	1	point2	2010-08-05 12:00:00
9	29.80	ID 9	1	1	point3	2010-08-05 12:00:00
10	10.00	ID 10	0	0	point1	2010-08-05 13:00:00
11	21.20	ID 11	0	1	point2	2010-08-05 13:00:00
12	29.50	ID 12	1	1	point3	2010-08-05 13:00:00

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

	values	ID	X1	X2	sp.ID	time
1	9.38	ID 1	0	0	point1	2010-08-05 10:00:00
2	19.80	ID 2	0	1	point2	2010-08-05 10:00:00
3	27.90	ID 3	1	1	point3	2010-08-05 10:00:00
4	9.76	ID 4	0	0	point1	2010-08-05 11:00:00

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 lines start coordinate or coordinate centre of weight are given in 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	9.38	19.8	27.9
2010-08-05 11:00:00	9.76	20.5	27.1
2010-08-05 12:00:00	11.30	19.5	29.8
2010-08-05 13:00:00	10.00	21.2	29.5

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

	2010-08-05 10:00:00	2010-08-05 11:00:00	2010-08-05 12:00:00
point1	9.38	9.76	11.3
point2	19.80	20.50	19.5
point3	27.90	27.10	29.8

```
> 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	9.38	19.8	27.9
2010-08-05 11:00:00	9.76	20.5	27.1
2010-08-05 12:00:00	11.30	19.5	29.8
2010-08-05 13:00:00	10.00	21.2	29.5

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]  9.38 19.80 27.90  9.76 20.50 27.10 11.30 19.50 29.80 10.00 21.20 29.50

> stfdf[["values"]]

[1]  9.38 19.80 27.90  9.76 20.50 27.10 11.30 19.50 29.80 10.00 21.20 29.50

> 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	9.38	OldIDs1	0.07166773	NewIDs12
2	19.80	OldIDs2	-0.36971586	NewIDs11
3	27.90	OldIDs3	1.48324567	NewIDs10
4	9.76	OldIDs4	0.10223975	NewIDs9
5	20.50	OldIDs5	0.07950131	NewIDs8
6	27.10	OldIDs6	-0.95841300	NewIDs7
7	11.30	OldIDs7	0.53338322	NewIDs6
8	19.50	OldIDs8	-0.66032487	NewIDs5
9	29.80	OldIDs9	1.06295021	NewIDs4
10	10.00	OldIDs10	-0.48384279	NewIDs3
11	21.20	OldIDs11	-0.40582657	NewIDs2
12	29.50	OldIDs12	-0.54698281	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)	9.38	OldIDs1	0.07166773	NewIDs12
2	(0, 1)	19.80	OldIDs2	-0.36971586	NewIDs11
3	(1, 1)	27.90	OldIDs3	1.48324567	NewIDs10

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

An object of class "STFDF"

Slot "data":

	values
1	9.38
2	19.80
3	27.90
4	9.76
5	20.50
6	27.10
7	11.30
8	19.50
9	29.80
10	10.00
11	21.20
12	29.50

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  9.38
2010-08-05 11:00:00  9.76
2010-08-05 12:00:00 11.30
2010-08-05 13:00:00 10.00

> 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    9.38
4    9.76
7   11.30
10  10.00

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  9.38 OldIDs1  0.07166773 NewIDs12
2 19.80 OldIDs2 -0.36971586 NewIDs11
3 27.90 OldIDs3  1.48324567 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)

Sparse data.frame 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

```

```

> 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   9.31 ID 1
2  21.60 ID 2
3  29.20 ID 3
4  38.70 ID 4
5  48.60 ID 5

```

```

6  60.50 ID 6
7  69.30 ID 7
8  79.60 ID 8
9  89.90 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:

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

An object of class "STSD"

```
Slot "data":
```

```

      values  ID
1    9.31 ID 1
2   21.60 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)  11.3 01dIDs7  0.5333832 NewIDs6
8      (0, 1)  19.5 01dIDs8 -0.6603249 NewIDs5
9      (1, 1)  29.8 01dIDs9  1.0629502 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  9.38
2010-08-05 11:00:00  9.76
2010-08-05 12:00:00 11.30
2010-08-05 13:00:00 10.00

```

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

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, "Open" and "Close" refer to the values for moments at opening and closing of the stock exchange, where "high" and "low" aggregated (minimum, maximum over the time interval between opening and closing times).

9 Worked examples

9.1 Interpolating Iris wind

This worked example is a shortened version of the analysis present in `demo(wind)`, in 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 spatiotemporal classes.

In the next fragment, we will load the wind data from package `gstat`, and convert character representation (such as 51d56'N) to proper numerical coordinates.

```
> 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 is to reshape these data. As space is sparse and time is rich, the data in data.frame `wind` come stations time series in columns. The station locations come in a separate data.frame, called `wind.loc`.

```
> 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
```

```
> wind.loc[1:3, ]

      coordinates      Station Code Latitude Longitude MeanWind
1  (-10.25, 51.9333)    Valentia  VAL  51d56'N  10d15'W      5.48
2    (-10, 54.2333)   Belmullet  BEL  54d14'N  10d00'W      6.75
3 (-8.98333, 53.7167) Claremorris  CLA  53d43'N   8d59'W      4.32
```

```

> library(mapdata)
> plot(wind.loc, xlim = c(-11, -5.4), ylim = c(51, 55.5), axes = T,
+      col = "red")
> map("worldHires", add = T, col = grey(0.5))
> text(coordinates(wind.loc), pos = 1, label = wind.loc$Station,
+      cex = 0.7)

```

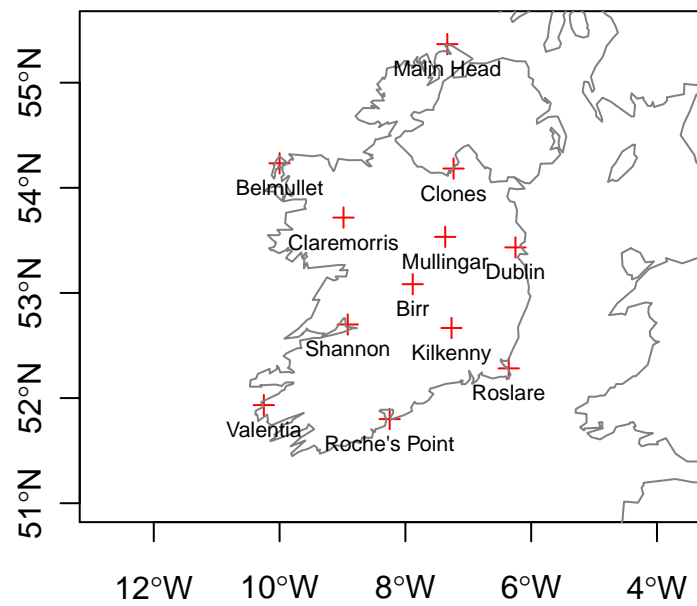


Figure 4: Station locations for Irish wind data

First, we will recode the time columns to an appropriate time data structure, and subtract a smooth trend of daily means:

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

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

```
> pts = coordinates(wind.loc[match(names(wind[4:15]), wind.loc$Code),
+ ])
> pts = SpatialPoints(pts)
> proj4string(pts) = "+proj=longlat +datum=WGS84"
> library(rgdal)
> pts = spTransform(pts, CRS("+proj=utm +zone=29 +datum=WGS84"))
> t = xts(1:nrow(wind), wind$time)
> stations = 4:15
> w = STFDF(pts, t, data.frame(values = as.vector(t(wind[stations]))))
> 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

```
> 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, CRS("+proj=utm +zone=29 +datum=WGS84"))
> grd = SpatialPixels(SpatialPoints(makegrid(m, n = 300)), proj4string = proj4string(m))
> w = w[, "1961-04"]
> covfn = function(x, y) {
+   du = spDists(coordinates(x), coordinates(y))
+   t1 = as.numeric(index(x))
+   t2 = as.numeric(index(y))
+   dt = abs(outer(t1, t2, "-"))
+   0.6 * exp(-du/750000) * exp(-dt/(1.5 * 3600 * 24))
+ }
> n = 10
> tgrd = xts(1:n, seq(min(index(w)), max(index(w)), length = n))
> pred = krige0(sqrt(values) ~ 1, w, STF(grd, tgrd), covfn)
> wind.pr = STFDF(grd, tgrd, data.frame(pred = pred))
```

9.2 Tracking data: trip and ltraj objects

```
> library(argosfilter)
> data(seal)
> library(sp)
```

```

> spl = list(list("sp.points", pts, first = F, cex = 0.5), list("sp.lines",
+   m, col = "grey"))
> print(stplot(wind.pr, col.regions = bpy.colors(), par.strip.text = list(cex = 0.5),
+   sp.layout = spl))

```

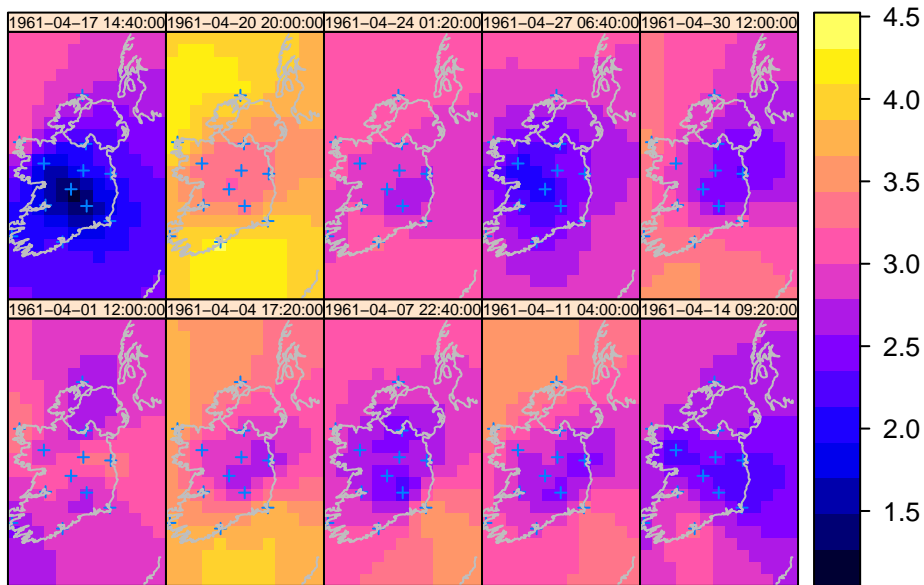


Figure 5: Space-time interpolations of wind (square root transformed, de-trended) over Ireland using a separable product covariance model, for 10 time points regularly distributed over the month for which daily data was considered (April, 1961)

```

> library(trip)
> library(maptools)
> trackAngle <- function(xy) {
+   angles <- abs(c(trackAzimuth(xy), 0) - c(0, rev(trackAzimuth(xy[nrow(xy):1,
+   ]))))
+   angles <- ifelse(angles > 180, 360 - angles, angles)
+   angles[is.na(angles)] <- 180
+   angles
+ }
> vmax <- 2
> ang <- c(15, 25)
> distlim <- c(2500, 5000)
> coordinates(seal) <- ~lon + lat
> proj4string(seal) <- CRS("+proj=longlat +ellps=WGS84")
> seal$id <- "seal"
> seal <- seal[!duplicated(seal$dttime), ]
> seal.tr <- trip(seal, c("dttime", "id"))
> seal.tr$speed.ok <- speedfilter(seal.tr, max.speed = vmax * 3.6)
> dsts <- trackDistance(coordinates(seal.tr)) * 1000
> ang <- trackAngle(coordinates(seal.tr))
> dprev <- c(0, dsts)
> dnext <- c(dsts, 0)
> ok <- (seal.tr$speed.ok | dprev <= 5000) & (seal.tr$lc > -9)
> plot(seal.tr[ok, ])
> seal.tr$filt.row <- 1:nrow(seal.tr)
> seal.tr$ok <- rep(FALSE, nrow(seal.tr))
> df <- seal.tr
> df <- df[ok, ]
> for (i in 1:length(distlim)) {
+   dsts <- trackDistance(coordinates(df)) * 1000
+   ang <- trackAngle(coordinates(df))
+   dprev <- c(0, dsts)
+   dnext <- c(dsts, 0)
+   ok <- (dprev <= distlim[i] | dnext <= distlim[i]) | ang >
+     ang[i]
+   ok[c(1:2, (length(ok) - 1):length(ok))] <- TRUE
+   df <- df[ok, ]
+   ok <- rep(TRUE, nrow(df))
+ }
> seal.tr$ok[match(df$filt.row, seal.tr$filt.row)] <- ok

```

9.3 Conversion from and to trip

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

```

> setClass("STSDFtrip", representation("STSDF", TOR.columns = "character"))
[1] "STSDFtrip"

```

```

> setAs("trip", "STSDFTrip", function(from) {
+   new("STSDFTrip", STSDF(as(from, "SpatialPoints"), xts(1:nrow(from),
+     from[[from@TOR.columns[1]]]), from@data), TOR.columns = from@TOR.columns)
+ })
> setAs("STSDFTrip", "trip", function(from) trip(SpatialPointsDataFrame(from@sp,
+   from@data), from@TOR.columns))
> d <- data.frame(x = 1:10, y = rnorm(10), tms = Sys.time() + 1:10,
+   id = gl(2, 5))
> coordinates(d) <- d[c("x", "y")]
> tr <- trip(d, c("tms", "id"))
> x = as(tr, "STSDFTrip")
> y = as(x, "trip")
> all.equal(y, tr)

[1] TRUE

```

9.4 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

```

```

> data(puechabon)
> locs <- puechabon$locs
> xy <- locs[, c("X", "Y")]
> da <- as.character(locs$Date)
> da <- as.POSIXct(strptime(as.character(locs$Date), "%y%m%d"))
> ltr <- as.ltraj(xy, da, id = locs$Name)
> foo <- function(dt) {
+   return(dt > (100 * 3600 * 24))
+ }
> l2 <- cutltraj(ltr, "foo(dt)", nextr = TRUE)
> setClass("ltraj", representation("list"))

[1] "ltraj"

> setClass("STSDFltraj", representation("STSDF"))

[1] "STSDFltraj"

```

```

> setAs("ltraj", "STSDFltraj", function(from) {
+   d = do.call(rbind, from)
+   n = unlist(lapply(from, nrow))
+   d$id = rep(unlist(t(sapply(from, attributes))[, 4]), times = n)
+   d$burst = rep(unlist(t(sapply(from, attributes))[, 5]), times = n)
+   t = xts(1:nrow(d), d$date)
+   o = t[, 1]
+   new("STSDFltraj", STSDF(SpatialPoints(d[c("x", "y")])[o],
+     t, d[o, ]))
+ })
> setAs("STSDFltraj", "ltraj", function(from) {
+   xy = coordinates(from@sp)
+   da = index(from@time)
+   as.ltraj(xy, da, id = from@data[, "id"], burst = from@data[,
+     "burst"])
+ })
> ltr.stsdf = as(l2, "STSDFltraj")
> ltr0 = as(ltr.stsdf, "ltraj")
> all.equal(l2, ltr0, check.attributes = FALSE)

[1] TRUE

```

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.