Package ‘spacetime’

September 24, 2017

Version 1.2-1
Title Classes and Methods for Spatio-Temporal Data
Depends R (>= 3.0.0)
Imports graphics, utils, stats, methods, lattice, sp (>= 1.1-0), zoo (>= 1.7-9), xts (>= 0.8-8), intervals
Suggests adehabitatLT, cshapes, diveMove, foreign, gstat (>= 1.0-16), maps, mapdata, maptools, plm, raster, RColorBrewer, rgdal, rgeos, RPostgreSQL, knitr, googleVis, ISOcodes
LazyData no
Description Classes and methods for spatio-temporal data, including space-time regular lattices, sparse lattices, irregular data, and trajectories; utility functions for plotting data as map sequences (lattice or animation) or multiple time series; methods for spatial and temporal selection and subsetting, as well as for spatial/temporal/spatio-temporal matching or aggregation, retrieving coordinates, print, summary, etc.
License GPL (>= 2)
URL http://github.com/edzer/spacetime
BugReports https://github.com/edzer/spacetime/issues
VignetteBuilder knitr
NeedsCompilation no
Author Edzer Pebesma [cre, aut], Benedikt Graeler [ctb], Tom Gottfried [ctb], Robert J. Hijmans [ctb]
Maintainer Edzer Pebesma <edzer.pebesma@uni-muenster.de>
Repository CRAN
Date/Publication 2017-09-24 21:50:53 UTC
Description

Air quality data obtained from the airBase European air quality data base. Daily averages for rural background stations in Germany, 1998-2009. In addition, NUTS1 regions (states, or Bundeslän-der) for Germany to illustrate spatial aggregation over irregular regions.

Usage

data(air)

Note

see vignette on overlay and spatio-temporal aggregation in this package; the vignette on using google charts shows where the ISO_3166_2_DE table comes from.

Author(s)

air quality data compiled for R by Benedict Graeler; NUTS1 level data obtained from http://www.gadm.org/
delta

References

http://acm.eionet.europa.eu/databases/airbase

Examples

data(air)
rural = STFDF(stations, dates, data.frame(PM10 = as.vector(air)))
# how DE was created from DE_NUTS1:
if (require(rgeos))
DE = gUnionCascaded(DE_NUTS1)

delta  

find default time interval end points when intervals are regular

Description

find default time interval end points when intervals are regular

Usage

delta(x)

Arguments

x object of class xts, or of another class that can be coerced into POSIXct;

Details

to find the interval size for the last observation (which has no next observation), x needs to be at least of length 2.

Value

sequence of POSIXct time stamps, indicating the end of the time interval, given by the next observation in x. The last interval gets the same width of the one-but-last interval.

Author(s)

Edzer Pebesma

References

http://www.jstatsoft.org/v51/i07/

Examples

x = as.POSIXct("2000-01-01") + (0:9) * 3600
delta(x)
Compute spatial or temporal empirical orthogonal function (EOF)

**Description**

Compute spatial or temporal empirical orthogonal function (EOF)

**Usage**

```r
eof(x, how = c("spatial", "temporal"), returnEOFs = TRUE, ...)
EOF(x, how = c("spatial", "temporal"), returnPredictions = TRUE, ...)
```

**Arguments**

- `x` object of class `stfdf`
- `how` character; choose "spatial" or "temporal" mode
- `returnEOFs` logical; if TRUE, the eigenvectors (EOFs) are returned in the form of a `Spatial` or `xts` object; if FALSE, the object returned by `prcomp` is returned, which can be printed, or from which a summary can be computed; see examples.
- `returnPredictions` logical; if TRUE, the functions are returned (i.e., predicted principle components, or PC scores); if FALSE, the object returned by `prcomp` is returned, which can be printed, or from which a summary can be computed; see examples (deprecated, see below).
- `...` arguments passed on to function `prcomp`; note that `scale=TRUE` needs to be specified to obtain EOFs based on correlation (default: covariance)

**Value**

In spatial mode, the appropriate `Spatial` object. In temporal mode, an object of class `xts`.

**Note**

`EOF` is deprecated: it mixes up spatial and temporal EOFs, and returns projections (PC scores) instead of EOFs (eigenvectors); to compute EOFs, use `eof`.

**Examples**

```r
if (require(gstat)) {
data(wind)
library(sp)
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"
```
fires

# match station order to names in wide table:
stations = 4:15
wind.loc = wind.loc[match(names(wind[stations]), wind.loc$Code),]
row.names(wind.loc) = wind.loc$Station
wind$time = ISOdate(wind$year+1900, wind$month, wind$day, 0)
space = list(values = names(wind)[stations])
wind.st = stConstruct(wind[stations], space, wind$time, SpatialObj = wind.loc)
# select first 500 time steps, to limit run time:
winds = wind.st[,1:500]
winds.eof.1 = eof(winds)
winds.eof.2 = eof(winds, "temporal")
winds.eof.1.PCs = eof(winds, returnEOFs = FALSE)
eof(winds, "temporal", returnEOFs = FALSE)
summary(eof(winds, returnEOFs = FALSE))
summary(eof(winds, "temporal", returnEOFs = FALSE))
plot(eof(winds, "temporal", returnEOFs = FALSE))
}

fires

Northern Los Angeles County Fires

Description

Wildfire occurrences in Northern Los Angeles County, California between 1976 and 2000. The spatial units are in scaled feet, taken from the NAD 83 state-plane coordinate system. One unit is equivalent to 100,000 feet or 18.9 miles. The times for the points were produced by the date package and represent the number of days since January 1, 1960.

Usage

data(fires)

Format

A data frame with 313 observations with day of occurrence, x and y coordinates.

Author(s)

Roger Peng, taken from (non-CRAN) package ptproc,
http://www.biostat.jhsph.edu/~rpeng/software/index.html;
example code by Roger Bivand.
Examples

data(fires)
fires$X <- fires$X*100000
fires$Y <- fires$Y*100000
library(sp)
coordinates(fires) <- c("X", "Y")
proj4string(fires) <- CRS("+init=epsg:2229 +ellps=GRS80")
dates <- as.Date("1960-01-01")+(fires$Time-1)
Fires <- STIDF(as(fires, "SpatialPoints"), dates, data.frame(time=fires$Time))
if (require(rgdal)) {
  library(maptools)
  library(mapdata)
  m <- map("county", "california", xlim=c(-119.1, -117.5),
            ylim=c(33.7, 35.0), plot=FALSE)
  cc <- spTransform(map2SpatialLines(m,
                        proj4string=CRS("+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84"),
                        CRS("+init=epsg:2229 +ellps=GRS80"))
  plot(cc, xlim=c(6300000, 6670000), ylim=c(1740000, 2120000))
  plot(slot(Fires, "sp"), pch=3, add=TRUE)
  stplot(Fires, sp.layout=list("sp.lines", cc))
}

mfn

Generic mfn method

Description

Compute mfn from spatial, temporal, or spatio-temporal data

Usage

mfn(x, ...)
## S3 method for class 'matrix'
mfn(x, ..., Sigma.Noise, use = "complete.obs")
## S3 method for class 'mts'
mfn(x, ..., use = "complete.obs")
## S3 method for class 'zoo'
mfn(x, ..., use = "complete.obs")
## S3 method for class 'SpatialPixelsDataFrame'
mfn(x, ..., use = "complete.obs")
## S3 method for class 'SpatialGridDataFrame'
mfn(x, ..., Sigma.Noise, use = "complete.obs")
## S3 method for class 'RasterStack'
mfn(x, ..., use = "complete.obs")
## S3 method for class 'RasterBrick'
mfn(x, ..., use = "complete.obs")
## S3 method for class 'STSDF'
mfn(x, ..., use = "complete.obs", mode = "temporal")
## mnf

### S3 method for class 'STFDF'

```r
mnf(x, ..., use = "complete.obs", mode = "temporal")
```

#### Arguments

- `x`: object for which an mnf method is available
- `...`: ignored
- `Sigma.Noise`: Noise covariance matrix; when missing, estimated from the data by using the covariance of lag-one spatial or temporal differences (MAF)
- `use`: method to deal with missing values when computing covariances; see `cov`
- `mode`: for ST objects: if "temporal", compute covariances in time dimension, if "spatial", compute them in spatial dimension.

#### Details

Uses MAF (Min/max Autocorrelation Factors) to estimate the noise covariance. This implementation estimates the noise covariance by $0.5 \text{Cov}(Z(s) - Z(s + \Delta))$, so that eigenvalues can be directly interpreted as approximate estimates of the noise covariance.

#### Value

object of class (c("mnf", "prcomp")); see `prcomp`. Additional elements are `values`, containing the eigenvalues.

#### See Also

http://r-spatial.org/r/2016/03/09/MNF-PCA-EOF.html

#### Examples

```
# temporal data:
set.seed(13531) # make reproducible
s1 = arima.sim(list(ma = rep(1,20)), 500)
s2 = arima.sim(list(ma = rep(1,20)), 500)
s3 = arima.sim(list(ma = rep(1,20)), 500)
s3 = s3 + rnorm(500, sd = 10)
d = cbind(s1,s2,s3)
plot(d)
m = mnf(d)
summary(m)
plot(predict(m))
```

# spatial example:
```
## Not run:
library(sp)
grd = SpatialPoints(expand.grid(x=1:100, y=1:100))
grided(grd) = TRUE
fullgrid(grd) = TRUE
pts = spsample(grd, 50, "random")
```
na.locf = replace NA attribute values; disaggregation time series

Description

replace NA attribute values in time series, using last or next observation, or using (temporal) interpolation, and disaggregation

Usage

## S3 method for class 'STFDF'
na.locf(object, na.rm = FALSE, ...)
## S3 method for class 'STFDF'
na.approx(object, x = time(object), xout, ..., na.rm = TRUE)
## S3 method for class 'STFDF'
na.spline(object, x = time(object), xout, ..., na.rm = TRUE)
Arguments

- **object**: object of class `stfdf`, with potentially NA values
- **na.rm**: logical; need non-replaced NA values be removed?
- **x**: times at which observations are taken; should not be modified
- **xout**: if present, new times at which the time series should be approximated (disaggregated)
- ...
  passed on to underlying zoo functions; see details

Details

details are found in `na.locf`, `na.approx`, `na.spline`.

Value

object of class `stfdf`, with NA values replaced.

Author(s)

Edzer Pebesma

References

http://www.jstatsoft.org/v51/i07/

Examples

```r
# toy example:
library(sp)
pts = SpatialPoints(cbind(c(0,1),c(0,1)))
Sys.setenv(TZ="GMT")
tm = seq.POSIXct("2012-11-25", as.POSIXct("2012-11-30"), "1 day")
df = data.frame(a = c(NA,NA,2,3,NA,NA,NA,2,NA,NA,4,NA), b = c(NA,2,3,4,5,1,2,NA,NA,NA,3))
x = STFDF(pts, tm, df)
as(x, "xts")
as(na.locf(x), "xts")
as(na.locf(x, fromLast = TRUE), "xts")
as(na.locf(na.locf(x), fromLast = TRUE), "xts")
# drops first record:
as(na.approx(x[,1]), "xts")
# keep it:
cbind(as(na.approx(x[,1], na.rm=FALSE), "xts"),
as(na.approx(x[,2]), "xts"))
cbind(as(na.spline(x[,1]), "xts"),
as(na.spline(x[,2]), "xts"))
#disaggregate:
xout = seq(start(x), end(x), "6 hours")
as(na.approx(x[,1], xout = xout), "xts")
as(na.spline(x[,1], xout = xout), "xts")
as(na.spline(x[,2], xout = xout), "xts")
```
# larger/real data:
data(air)
rural = STFDF(stations, dates, data.frame(air = as.vector(air)))
# fill NA's with last non-NA
r = na.locf(rural)
# sample (NOT aggregate) to monthly:
m = seq(start(rural), end(rural), "1 month")
stplot(na.approx(rural[1:20,"2003:2005"], xout = m), mode = 'ts')

---

**nbMult**

*convert a spatial nb object to a matching STF object*

**Description**

convert a spatial nb object to a matching STF object

**Usage**

nbMult(nb, st, addT = TRUE, addST = FALSE)

**Arguments**

- `nb`: object of class nb (see package spdep), which is valid for the spatial slot of object `st`. `length(nb)` should equal `length(st@sp)`
- `st`: object of class STF
- `addT`: logical; should temporal neighbours be added?
- `addST`: logical; should spatio-temporal neighbours be added?

**Details**

if both `addT` and `addST` are false, only spatial neighbours are added for each time replicate.

details are found in


**Value**

object of class nb

**Author(s)**

Edzer Pebesma
Description

consistent spatio-temporal overlay for ST, STS and STI objects, as well as their *DF counterpart: retrieves the indexes or attributes from one geometry at the spatio-temporal points of another

Usage

```r
## S4 method for signature 'STF,STF'
over(x, y, returnList = FALSE, fn = NULL, ...)
## S4 method for signature 'xts,xts'
over(x, y, returnList = FALSE, fn = NULL, ...)
## S4 method for signature 'ST'
aggregate(x, by, FUN, ..., simplify = TRUE)
```

Arguments

- `x`: geometry (S/T locations) of the queries
- `y`: layer from which the geometries or attributes are queried
- `returnList`: logical; determines whether a list is returned, or an index vector
- `fn`: (optional) a function; see value
- `by`: geometry over which attributes in `x` are aggregated (this can be a Spatial* geometry, or a ST* geometry), or temporal aggregation, such as "month", "10 minutes", or a function such as `as.yearmon`; see `aggregate.zoo`. In case `x` is of class STFDF, argument by may be "time" or "space", in which cases aggregation over all time or all space is carried out.
- `FUN`: aggregation function
- `simplify`: boolean; if TRUE, and space or time dimensions can be dropped, the simpler (Spatial or xts) object will be returned
- `...`: arguments passed on to function `fn` or `FUN`

Value

an object of length `length(x)`, or a data.frame with number of rows equal to `length(x)`. If `returnList` is FALSE, a vector with indices of `y` for each geometry (point, grid cell centre, polygon or lines x time point) in `x`. If `returnList` is TRUE, a list of length `length(x)`, with list element `i` the vector of indices of the geometries in `y` that correspond to the `i`-th geometry in `x`.

The aggregate method for ST objects aggregates the attribute values of `x` over the geometry (space, time, or space-time) of `by`, using aggregation function `FUN`.

For the matching of time intervals, see `timeMatch`.

For setting, or retrieving whether time represents intervals, see `timeIsInterval`. 
Methods

x = "STF", y = "STF"
            finds the row index of the instance or interval of time instances of x matching
to y. Only if timeIsInterval(x) == TRUE, intervals are sought. In that case, time intervals
start at the time instance of a record, and end at the next. The last time interval length is set to
the interval length of the one-but-last (non-zero) interval. In case of a single time instance for
y, its interval is right-open.

Note

See also over; methods intersecting SpatialLines with anything else, or SpatialPolygons with Spa-
tialPolygons, need rgeos to be loaded first.

Author(s)

Edzer Pebesma, <edzer.pebesma@uni-muenster.de>

References

http://www.jstatsoft.org/v51/i07/

See Also

over; vignette('sto'), vignette('over'), timeMatch, timeIsInterval

---

read.tgrass  read or write tgrass (time-enabled grass) files

Description

read or write tgrass (time-enabled grass) files

Usage

read.tgrass(fname, localName = TRUE, useTempDir = TRUE, isGeoTiff = TRUE)
write.tgrass(obj, fname, ...)

Arguments

fname file name to read from, or write to
localName logical; if TRUE, fname is a local file, else it is a the full path name to the file
useTempDir logical: use a temporary directory for extraction?
isGeoTiff logical: are the files in the tar.gz file GeoTIFFs?
obj object to export, of class STFDF or RasterStack
... arguments passed on to writeRaster
Details
The tgrass format is a gzip’ed tar file (.tar.gz) that has geotiff files (with suffix .tif), and three files (list.txt, proj.txt and init.txt) describing the file names and time slices, coordinate reference system, and dimensions

Value
read.tgrass returns an object of class RasterStack, writegrass returns nothing

Author(s)
Edzer Pebesma; time-enabled grass by Soeren Gebbert

References
http://dx.doi.org/10.1016/j.envsoft.2013.11.001

Examples
```r
# Not run:
library(spacetime)
r = read.tgrass("precipitation_1950_2011_yearly.tar.gz", useTempDir = FALSE)
write.tgrass(r, "myfile.tar.gz")

# End(Not run)
```

---

ST-class

Class “ST”

Description
An abstract class from which useful spatio-temporal classes are derived

Usage

```
ST(sp, time, endTime)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sp</td>
<td>an object deriving from class Spatial, such as a SpatialPoints or SpatialPolygons</td>
</tr>
<tr>
<td>time</td>
<td>an object of class xts, or a time vector (currently: Date, POSIXct, timeDate, yearmon and yearqtr; are supported; see xts); in the latter case, it should be in time order</td>
</tr>
<tr>
<td>endTime</td>
<td>vector of class POSIXct holding end points of time intervals</td>
</tr>
</tbody>
</table>

Objects from the Class

Objects of this class are not meant to be useful; only derived classes can be meaningful
Slots

sp: Object deriving from class "Spatial"

time: Object of class "xts"

Methods

[[ signature(obj = "ST")]: retrieves the attribute element

$ signature(obj = "ST")]: retrieves the attribute element

$[< signature(obj = "ST")]: sets or replaces the attribute element

$< signature(obj = "ST")]: sets or replaces the attribute element

Note

argument (and object slot) sp can be pure geometry, or geometry with attributes. In the latter case, the geometries are kept with the sp slot, and only replicated (when needed) on coercion to the long format, with as.data.frame.

Slot time needs to be of class xts; if a time or date vector is passed as argument to SP, it will be converted into an xts object.

When endTime is missing, an error is thrown.

ST is meant as a super-class, and is not to be used for representing data, similar to Spatial in the sp package.

Author(s)

Edzer Pebesma, <edzer.pebesma@uni-muenster.de>

References

http://www.jstatsoft.org/v51/i07/

Examples

time = as.Date("2008-01-01")+1:2
library(sp)
sp = SpatialPoints(cbind(c(0,1),c(0,1)))
ST(sp, time, delta(time))
**stbox**

*obtain ranges of space and time coordinates*

---

**Description**

obtain ranges of space and time coordinates

**Usage**

```
stbox(obj)
bbox(obj)
```

**Arguments**

obj object of a class deriving from ST

**Value**

`stbox` returns a `data.frame`, with three columns representing x-, y- and time-coordinates, and two rows containing min and max values. `bbox` gives a matrix with coordinate min/max values, compatible to `bbox`

**Methods**

```
  stbox signature(x = "ST"): obtain st range from object
```

---

**stConstruct**

*create ST* objects from long or wide tables

---

**Description**

create ST* objects from long or wide tables

**Usage**

```
stConstruct(x, space, time, SpatialObj = NULL, TimeObj = NULL, crs = CRS(as.character(NA)), interval, endTime)
```
Arguments

- **x**
  - object of class `matrix` or `data.frame`, holding the long, space-wide or time-wide table; see details.

- **space**
  - in case `x` is a long table, character or integer holding the column index in `x` where the spatial coordinates are (if `length(space)==2`) or where the ID of the spatial location is (if `length(space)==1`). If `x` is a space-wide table, a list with each (named) list element a set of columns that together form a variable

- **time**
  - in case `x` is a long table, character or integer indicating the column in `x` with times;

- **SpatialObj**
  - object of class `Spatial-class`, containing the locations of a time-wide table, or the locations of a long table

- **TimeObj**
  - in case of space-wide table, object of class `xts`, containing the times for each of the columns in a list element of `space`

- **crs**
  - object of class `CRS-class`; only used when coordinates are in `x` and no CRS can be taken from `SpatialObj`

- **interval**
  - logical; specifies whether time should reflect time instance (FALSE) or time intervals (TRUE). If omitted, defaults values depend on the class

- **endTime**
  - vector of POSIXct, specifying (if present) the end points of observation time intervals

Details

For examples, see below.

A long table is a data.frame with each row holding a single observation in space-time, and particular columns in this table indicate the space (location or location ID) and time.

A space-wide table is a table in which different columns refer to different locations, and each row reflects a particular observation time.

A time-wide table is a table where different times of a particular characteristic are represented as different columns; rows in the table represent particular locations or location IDs.

Value

Depending on the arguments, an object of class STIDF or STFDF.

References

http://www.jstatsoft.org/v51/i07/

Examples

```r
# example 0: construction of STFDF from long table:
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
t = as.POSIXct(paste(yrs, "-01-01", sep=""), tz = "GMT")  
# deselect District of Columbia, polygon 8, which is not present in Produc:
Produc$st = STFDY(states[-8], t, Produc[(order(Produc[,2], Produc[,1]))],)

# example 1: st from long table, with states as Spatial object:
# use Date format for time:
Produc$time = as.Date(paste(yrs, "01", "01", sep=""))
# take centroids of states:
xy = coordinates(states[-8])
Produc$x = xy[,1]
Produc$y = xy[,2]
# using stConstruct, use polygon centroids for location:
x = stConstruct(Produc, c("x", "y"), "time", interval = TRUE)
class(x)
stplot(x[,,"unemp"])

# alternatively, pass states as SpatialObj:
Produc$state = gsub("TENNESSE", "TENNESSEE", Produc$state)
Produc$state = gsub("", " ", tolower(Produc$state))
x = stConstruct(Produc, "State", "time", states[-8])
class(x)
all.equal(x, Produc.st, check.attributes = FALSE)

# stConstruct multivariable, time-wide
library(maptools)
fname = system.file("shapes/sids.shp", package="maptools")[1]
nc = readShapePoly(fname, proj4string = CRS("+proj=longlat +datum=NAD27 +ellps=clrk66"))
timesList = list()
BIR=c("BIR74", "BIR79"),  # sets of variables that belong together
NWBIR=c("NWBIR74", "NWBIR79"), # only separated by space
SID=c("SID74", "SID79")
)
t = as.Date(c("1974-01-01","1979-01-01"))
nc.st = stConstruct(as(nc, data.frame), geometry(nc), timesList, TimeObj = t, interval = TRUE)

# stConstruct multivariable, space-wide
if (require(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"

# match station order to names in wide table:
stations = 4:15
wind.loc = wind.loc[match(names(wind[stations]), wind.loc$Code),]
row.names(wind.loc) = wind.loc$Station
# convert to utm zone 29, to be able to do interpolation in
# proper Euclidian (projected) space:

# create time variable
wind$time = ISOdate(wind$year+1900, wind$month, wind$day, 0)

w = STFDF(wind.loc, wind$time,
data.frame(values = as.vector(t(wind[stations]))))
space = list(values = names(wind)[stations])
wind.st = stConstruct(wind[stations], space, wind$time, SpatialObj = wind.loc, interval = TRUE)
all.equal(w, wind.st)
class(wind.st)
}

---

**STFDF-class**

**Class "STFDF"**

### Description

A class for spatio-temporal data with full space-time grid; for n spatial locations and m times, n x m observations are available

### Usage

```r
STF(sp, time, endTime = delta(time))
STFDF(sp, time, data, endTime = delta(time))
```

### Arguments

- **sp**  
  object of class `Spatial`, having n elements

- **time**  
  object holding time information, of length m; see `ST` for details

- **endTime**  
  vector of class POSIXct, holding end points of time intervals; by default, time intervals equal the time step width, see `delta`

- **data**  
  data frame with n*m rows corresponding to the observations (spatial index moving fastest)

- **x**  
  an object of class STFDF

- **i**  
  selection of spatial entities

- **j**  
  selection of temporal entities (see syntax in package xts)
STFDF-class

... selection of attribute(s)
drop if TRUE and a single spatial entity is selected, an object of class xts is returned; if TRUE and a single temporal entity is selected, and object of the appropriate Spatial class is returned; if FALSE, no coercion to reduced classes takes place
from object of class STFDF
to target class
strict ignored

Value

the as.data.frame coercion returns the full long table, with purely spatial attributes and purely time attributes replicated appropriately.

Objects from the Class

Objects of this class represent full space/time data with a full grid (or lattice) layout

Slots

sp: spatial object; see ST-class
time: temporal object; see ST-class
data: Object of class data.frame, which holds the measured values; space index cycling first, time order preserved

Methods

[ signature(x = "STFDF"): selects spatial entities, temporal entities, and attributes
coerce STFDF,xts
coerce STFDF,Spatial(from) coerces to (wide form) SpatialXxDataFrame, where SpatialXx is the spatial class of from@sp
plot signature(x = "STF", y = "missing"): plots space-time layout
plot signature(x = "STFDF", y = "missing"): plots space-time layout, indicating full missing valued records

Author(s)

Edzer Pebesma, <edzer.pebesma@uni-muenster.de>

References

http://www.jstatsoft.org/v51/i07/
Examples

```r
sp = cbind(x = c(0,0,1), y = c(0,1,1))
row.names(sp) = paste("point", 1:nrow(sp), sep="")
library(sp)
sp = SpatialPoints(sp)
time = 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 = STDFD(sp, time, mydata)
stfdf
```

```
stfdf[1:2,]
stfdf[,1:2]
stfdf[,2]
stfdf[,"values"]
stfdf[1,]
stfdf[,2]
as(stfdf[,1], "xts")
as(stfdf[,2], "xts")
# examples for [[, [<=, $ and $<-
stfdf[1]]]
stfdf["values"]
stfdf["newVal"] <- rnorm(12)
stfdf$ID
stfdf$ID = paste("OldIDs", 1:12, sep="")
stfdf$NewID = paste("NewIDs", 12:1, sep="")
stfdf
x = stfdf[stfdf[1:2,]]
all.equal(x, stfdf[1:2,])
all.equal(stfdf, stfdf[stfdf,]) # converts character to factor...
```

---

**STIDF-class**

Class "STIDF"

**Description**

A class for unstructured spatio-temporal data; for n spatial locations and times, n observations are available.

**Usage**

```r
STI(sp, time, endTime)
STIDF(sp, time, data, endTime)
## S4 method for signature 'STIDF'
x[i, j, ..., drop = FALSE]
## S4 method for signature 'STIDF,STSDF'
coerce(from, to, strict=TRUE)
```
**STIDF-class**

**Arguments**

- **sp**
  - object of class Spatial

- **time**
  - object holding time information; when STIDF is called, a non-ordered vector with times, e.g. POSIXct will also work, and rearrange the sp and data slots according to the ordering of time; for this to work no ties should exist.

- **endTime**
  - vector of class POSIXct, indicating the end points of time intervals for the observations. By default, for STI objects time is taken, indicating that time intervals have zero width (time instances)

- **data**
  - data frame with appropriate number of rows

- **x**
  - an object of class STFDF

- **i**
  - selection of record index (spatial/temporal/spatio-temporal entities)

- **j**
  - or character string with temporal selection

- **...**
  - first element is taken as column (variable) selector

- **drop**
  - if TRUE and a single spatial entity is selected, an object of class xts is returned (NOT yet implemented); if TRUE and a single temporal entity is selected, and object of the appropriate Spatial class is returned; if FALSE, no coercion to reduced classes takes place

- **from**
  - object of class STFDF

- **to**
  - target class

- **strict**
  - ignored

**Objects from the Class**

Objects of this class carry full space/time grid data

**Slots**

- **sp**
  - Object of class "Spatial"

- **time**
  - Object holding time information, see ST-class

- **data**
  - Object of class data.frame, which holds the measured values

**Methods**

- `signature(x = "STIDF")`: selects spatial-temporal entities, and attributes

**Note**

arguments sp, time and data need to have the same number of records, and regardless of the class of time (xts or POSIXct) have to be in corresponding order: the triple sp[i], time[i] and data[i,] refer to the same observation

**Author(s)**

Edzer Pebesma, <edzer.pebesma@uni-muenster.de>
References

http://www.jstatsoft.org/v51/i07/

Examples

```r
sp = cbind(x = c(0, 0, 1), y = c(0, 1, 1))
row.names(sp) = paste("point", 1:nrow(sp), sep="\"")
library(sp)
sp = SpatialPoints(sp)
time = 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)
stidf = as(STFDF(sp, time, mydata), "STIDF")
stidf[1:2,]
all.equal(stidf, stidf[stidf,])
```

---

**stInteraction**

subtract marginal (spatial and temporal) means from observations

**Description**

subtract marginal (spatial and temporal) means from observations

**Usage**

```r
stInteraction(x, ...)
```

**Arguments**

- `x` object of class STFDF
- `...` arguments passed to rowMeans, colMeans and mean, such as na.rm=TRUE

**Value**

object of class STFDF with each attribute replaced by its residual, computed by $y_{ij} = x_{ij} - m_{.j} - m_{i.} / m$ with $m$ the grand mean, $m_{.j}$ the temporal mean, $m_{i.}$ the spatial mean and $m$ the grand mean.

**Examples**

```r
if (require(gstat)) {
library(sp)
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"
# match station order to names in wide table:
stations = 4:15
wind.loc = wind.loc[match(names(wind[stations]), wind.loc$Code),]
row.names(wind.loc) = wind.loc$Station
wind$time = ISOdate(wind$year+1900, wind$month, wind$day, 0)
space = list(values = names(wind)[stations])
wind.st = stConstruct(wind[stations], space, wind$time, SpatialObj = wind.loc)

wind.sti = stInteraction(wind.st)
# temporal means for any station should be zero:
c(mean(wind.sti[,3]),
# spatial mean for each time step should be zero:
mean(wind.sti[,5][[,1]]))

stplot  produce trellis plot for STxDF object

Description

create trellis plot for ST objects

Usage

stplot(obj, 

stplot.STDF(obj, names.attr = trimDates(obj), 

as.table = TRUE, at, cuts = 15, scales = list(draw = FALSE),
animate = 0, mode = "xy", scaleX = 0, auto.key = list(space = key.space),
main, key.space = "right", type = "l", do.repeat = TRUE, range.expand = 0.001)
stplot.STIDF(obj, 

names.attr = NULL, as.table = TRUE,
scales = list(draw = FALSE), xlab = NULL, ylab = NULL,
type = "p", number = 6, tcuts, sp.layout = NULL, xlim =
bbox(obj)[1, ], ylim = bbox(obj)[2, ])

Arguments

obj object of a class deriving from ST
names.attr names that will be used in the strip; trimDates(obj) trims "-01" ending(s) from printed Dates
as.table logical: if TRUE, time will increas from top to bottom; if FALSE, time will increase from bottom to top
at values at which colours will change; see levelplot
cuts numeric; number of levels the range of the attribute would be divided into
animate numeric; if larger than 0, the number of seconds between subsequent animated time steps (loop; press ctrl-C or Esc to stop)
mode plotting mode; if "xy", maps for time steps are plotted; if "xt", a space-time plot is constructed (see argument `scaleX`, but read details below); if "ts", multiple-locations time series are plotted in a single plot, or in a separate panel for each attribute; if "tp" single- or multi-attribute time series are plotted in multiple panels, one panel per location.
scaleX integer: 0, 1 or 2; when `mode` is "xt", used to determine whether the index of the spatial location is shown (0), the x coordinate (1) or the y coordinate (2).
auto.key see the auto.key argument in `xyplot`
main character; plot title, use NULL to omit title
key.space character; see `xyplot`
scales scales drawing; see scales argument of `xyplot`
xlab x-axis label
ylab y-axis label
type character; use 'l' for lines, 'p' for symbols, 'b' for both lines and symbols
do.repeat logical; repeat the animation in an infinite loop?
rangexpand numeric; if `at` is not specified, expand the data range with this factor to cover all values
number number of time intervals, equally spaced
tcuts time cuts in units of `index(obj)`; this overrides `number`
sp.layout list or NULL; see `spplot`
... arguments passed on to `spplot` in case of plotting objects of class `STFDF` or `STIDF`, or to `xyplot` in case of `stplot.STIDF`
xlim numeric, x range
ylim numeric, y range

Value

In non-animation and "xy" mode, `stplot` is a wrapper around `spplot`, that automatically plots each time stamp in a panel. The returned value is a lattice plot.

In "xt" mode, a space-time plot with space on the x-axis and time on the y-axis is plotted. By default, the space ID is plotted on the x-axis, as space can be anything (points, polygons, grid cells etc). When `scaleX` is set to 1 or 2, the x- resp. y-coordinates of the spatial locations, obtained by coordinates, is used instead. Beware: when the x-coordinate is plotted, and for each (x,t) element multiple y-coordinates are sent to the plot, it is not clear which (x,y,t) value becomes the plotted value, so slicing single y values is advised – no checking is done. The returned value is a lattice plot.

In animation mode (animate > 0), single maps are animated in an endless loop, with animate seconds between each. No proper value is returned: the loop needs to be interrupted by the user.
**Methods**

- `stplot` signature\(x = \text{"STDF"}\): plots object of class \text{STDF}
- `stplot` signature\(x = \text{"STSDF"}\): plots object of class \text{STSDF}
- `stplot` signature\(x = \text{"STI"}\): plots object of class \text{STI}
- `stplot` signature\(x = \text{"STIDF"}\): plots object of class \text{STIDF}
- `stplot` signature\(x = \text{"STT"}\): plots object of class \text{STT}
- `stplot` signature\(x = \text{"STTDF"}\): plots object of class \text{STTDF}

**Note**

vignette("spacetime") contains several examples

**References**

http://www.jstatsoft.org/v51/i07/

---

### Class "STSDF"

**Description**

A class for spatio-temporal data with partial space-time grids; for \(n\) spatial locations and \(m\) times, an index table is kept for which nodes observations are available

**Usage**

```r
STS(sp, time, index, endTime = delta(time))
STSDF(sp, time, data, index, endTime = delta(time))
## S4 method for signature 'STSDF'

x[i, j, ..., drop = is(x, "STSDF")]
## S4 method for signature 'STSDF,STFDF'

coerce(from, to, strict=TRUE)
## S4 method for signature 'STSDF,STIDF'

coerce(from, to, strict=TRUE)
```

**Arguments**

- `sp` object of class \text{Spatial}
- `time` object holding time information; see \text{ST-class}
- `data` data frame with rows corresponding to the observations (spatial index moving faster than temporal)
- `index` two-column matrix: rows corresponding to the nodes for which observations are available, first column giving spatial index, second column giving temporal index
endTime vector of class POSIXct with end points of time intervals for the observations
x an object of class STFDF
i selection of spatial entities
j selection of temporal entities (see syntax in package xts)
... selection of attribute(s)
drop if TRUE and a single spatial entity is selected, an object of class xts is returned; if TRUE and a single temporal entity is selected, and object of the appropriate Spatial class is returned; if FALSE, no coercion to reduced classes takes place
from object of class STFDF
to target class
strict ignored

Objects from the Class

Objects of this class carry sparse space/time grid data

Slots

sp: Object of class "Spatial"
time: Object holding time information; see ST-class for permitted types
index: matrix of dimension n x 2, where n matches the number of rows in slot data
data: Object of class data.frame, which holds the measured values

Methods

[ signature(x = "STSDF"): selects spatial entities, temporal entities, and attributes
plot signature(x = "ST", y = "missing"): plots space-time layout
plot signature(x = "STSDF", y = "missing"): plots space-time layout, indicating records partially NA

Author(s)

Edzer Pebesma, <edzer.pebesma@uni-muenster.de>

References

http://www.jstatsoft.org/v51/i07/

See Also

delta
**Examples**

```r
sp = cbind(x = c(0,0,1), y = c(0,1,1))
row.names(sp) = paste("point", 1:nrow(sp), sep="")
library(sp)
sp = SpatialPoints(sp)
library(xts)
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 = STTDF(sp, time, mydata)
stfdf
stddf = as(stfdf, "STSDF")
stddf[1:2,]
stddf[,1:2]
stddf[,2]
stddf[,"values"]
stddf[1,]
stddf[2,]
# examples for [[, [[<-, $ and $<- stddf[[1]]
stddf["values"]]
stddf["newVal"] <- rnorm(12)
stddf$ID
stddf$ID = paste("OldIDs", 1:12, sep="")
stddf$NewID = paste("NewIDs", 12:1, sep="")
stddf
x = stddf[stddf,]
x = stddf[stddf[1:2,],]
all.equal(x, stddf[1:2,])
```

---

### STTDF-class

**Class "STTDF"**

**Description**

A class for spatio-temporal trajectory data

**Usage**

```r
## S4 method for signature 'STTDF,ltraj'
coerce(from, to, strict=TRUE)
## S4 method for signature 'ltraj,STTDF'
coerce(from, to, strict=TRUE)
```
Arguments

from          from object
to            target class
strict        ignored

Objects from the Class

Objects of this class carry sparse (irregular) space/time data

Slots

sp: Object of class "Spatial", containing the bounding box of all trajectories
time: Object of class "xts", containing the temporal bounding box of all trajectories
traj: Object of class list, each element holding an STI object reflecting a single trajectory;
data: Object of class data.frame, which holds the data values for each feature in each trajectory

Methods

[ signature(x = "STTDF")]: select trajectories, based on index, or spatial and/or temporal predicates

Note

The data.frame needs to have a column called burst which is a factor (or character) and contains the grouping of observations that come from a continuous sequence of observations. In addition, a column id is used to identify individual items.

Author(s)

Edzer Pebesma, <edzer.pebesma@uni-muenster.de>

References

http://www.jstatsoft.org/v51/i07/

Examples

library(sp)
m = 3# nr of trajectories
n = 100 # length of each
l = vector("list", m)
t0 = as.POSIXct("2013-05-05", tz="GMT")
set.seed(1331) # fix randomness
for (i in 1:m) {
  x = cumsum(rnorm(n))
y = cumsum(rnorm(n))
sp = SpatialPoints(cbind(x,y))
#t = t0 + (0:(n-1) + (i-1)*n) * 60
 t = t0 + (0:(n-1) + (i-1)*n/2) * 60
}
timeIsInterval

```r
l[[1]] = STI(sp, t)
}
st = STT(l)
stddf = STTDF(stt, data.frame(attr = rnorm(n*m), id = paste("ID", rep(1:m, each=n))))
x = as(stt, "STI")
stplot(stddf, col=1:m, scales=list(draw=TRUE))
stplot(stddf, by = "id")
stplot(stddf[1])
stplot(stddf[1])

# select a trajectory that intersect with a polygon
p = Polygon(cbind(x=c(-20,-15,-15,-20,-20), y=c(10,10,15,15,10)))
pol=SpatialPolygons(list(Polygons(list(p), "ID")))
if (require(rgeos)) {
  stplot(stddf[pol])
  names(stddf[pol]@traj)
  stplot(stddf[pol][2],col=1:2)
  stplot(stddf[,t0])
  stplot(stddf[,"2013"])
  stplot(stddf[pol,"2013"])
  is.null(stddf[pol,t0])
}
```

timeIsInterval | retrieve, or set, information whether time reflects instance (FALSE) or intervals (TRUE)

**Description**
retrieve, or set, information whether time reflects instance (FALSE) or intervals (TRUE)

**Usage**

```r
timeIsInterval(x, ...)
timeIsInterval(x) <- value
```

**Arguments**

- `x` object, of any class
- `...` ignored
- `value` logical; sets the timeIsInterval value

**Value**
logical; this function sets or retrieves the attribute timeIsInterval of x, UNLESS x is of class ST, in which case it sets or retrieves this attribute for the time slot of the object, i.e. timeIsInterval(x@time) <- value
Note

From spacetime 0.8-0 on, timeIsInterval is dropped in favour of a more generic time intervals by specifying endTime of each observation.

See Also

over, timeIsInterval

timematch match two (time) sequences

Description

match two (time) sequences, where each can be intervals or instances.

Usage

timematch(x, y, returnList = FALSE, ...)

Arguments

x: ordered sequence, e.g. of time stamps
y: ordered sequence, e.g. of time stamps
returnList: boolean; should a list be returned with all matches (TRUE), or a vector with single matches (FALSE)?
...: end.x and end.y can be specified for xts and POSIXct methods

Details

When x and y are of class xts or POSIXct, end.x and end.y need to specify endpoint of intervals. In case x and y are both not intervals, matching is done on equality of values, using match.

If x represents intervals, then the first interval is from x[1] to x[2], with x[1] included but x[2] not (left-closed, right-open). In case of zero-width intervals (e.g. x[1]==x[2]), nothing will match and a warning is raised. Package intervals is used to check overlap of intervals, using, interval_overlap.

Value

if returnList = FALSE: integer vector of length length(x) with indexes of y matching to each of the elements of x, or NA if there is no match. See section details for definition of match.

if returnList = TRUE: list of length length(x), with each list element an integer vector with all the indexes of y matching to that element of x.
timeMatch

Author(s)

Edzer Pebesma

References

http://www.jstatsoft.org/v51/i07/

See Also

over, timeIsInterval, interval_overlap

Examples

t0 = as.POSIXct("1999-10-10")
x = t0 + c(0.5+c(2,2.1,4),5)*3600
y = t0 + 1.5 * 3600
x
y
#timeIsInterval(x) = FALSE
#timeIsInterval(y) = FALSE
timeMatch(x,y, returnList = FALSE)
timeMatch(x,y, returnList = TRUE)
#timeIsInterval(x) = TRUE
#timeIsInterval(y) = FALSE
#timeIsInterval(x, y, returnList = TRUE, end.y = delta(y))
timeMatch(x,y, returnList = TRUE, end.y = delta(y))
#timeIsInterval(x) = TRUE
#timeIsInterval(y) = FALSE
#timeIsInterval(x, y, returnList = TRUE, end.x = delta(x), end.y = delta(y))
timeMatch(x,y, returnList = TRUE, end.x = delta(x), end.y = delta(y))
#timeIsInterval(x) = TRUE
#timeIsInterval(y) = FALSE
#timeIsInterval(x, y, returnList = TRUE, end.x = delta(x))
timeMatch(x,y, returnList = TRUE, end.x = delta(x))

x = as.POSIXct("2000-01-01") + (0:9) * 3600
y = x + 1
x
y
TI = function(x, ti) {
  timeIsInterval(x) = ti
  x
}

#timeMatch(TI(y,FALSE),TI(y,FALSE))
#timeMatch(TI(y,TRUE), TI(y,TRUE))
#
#timeMatch(TI(x,FALSE), TI(y,FALSE))
#timeMatch(TI(x,TRUE), TI(y,FALSE))
#timeMatch(TI(x,TRUE), TI(y,TRUE))
#
#timeMatch(TI(x,FALSE),TI(y,FALSE), returnList = TRUE)
#timeMatch(TI(x,FALSE),TI(y,TRUE), returnList = TRUE)
#timeMatch(TI(x,TRUE), TI(y,FALSE), returnList = TRUE)
unstack

write STFDF to table forms

Description

create table forms of STFDF objects

Usage

```r
## S3 method for class 'STFDF'
unstack(x, form, which = 1, ...)
## S3 method for class 'STFDF'
as.data.frame(x, row.names, ...)
```

Arguments

- `x`: object of class STFDF
- `form`: formula; can be omitted
- `which`: column name or number to have unstacked
- `row.names`: row.names for the data.frame returned
- `...`: arguments passed on to the functions `unstack` or `as.data.frame`

Value

unstack returns the data in wide format, with each row representing a spatial entity and each column a time; see `unstack` for details and default behaviour.
as.data.frame returns the data.frame in long format, where the coordinates of the spatial locations (or line starting coordinates, or polygon center points) and time stamps are recycled accordingly.

Examples

```r
sp = cbind(x = c(0,0,1), y = c(0,1,1))
row.names(sp) = paste("point", 1:nrow(sp), sep="")
library(sp)
sp = SpatialPoints(sp)
library(xts)
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)
as.data.frame(stfdf, row.names = IDs)
```
unstack

unstack(stfdf)
t(unstack(stfdf))
unstack(stfdf, which = 2)
Index

*Topic classes
  ST-class, 13
  STFDF-class, 18
  STIDF-class, 20
  STSDF-class, 25
  STSTDF-class, 27

*Topic datasets
  air, 2
  fires, 5

*Topic dplot
  stbox, 15
  stplot, 23
  timeisinterval, 29

*Topic manip
  delta, 3
  EOF, 4
  na.locf, 8
  nbMult, 10
  read.tgrass, 12
  stConstruct, 15
  stInteraction, 22
  timeMatch, 30
  unstack, 32

*Topic methods
  over-methods, 11
  [,STF-method (STFDF-class), 18
  [,STFDF-method (STFDF-class), 18
  [,STIDF-method (STIDF-class), 20
  [,STIDF-method (STIDF-class), 20
  [,STS-method (STSDF-class), 25
  [,STSDF-method (STSDF-class), 25
  [,STSTDF-method (STSTDF-class), 27
  [,STSTDF-method (STSTDF-class), 27
  [[,ST,ANY,missing-method (ST-class), 13
  [[,<-,ST,ANY,missing-method (ST-class), 13
  $,ST-method (ST-class), 13
  $<-,ST-method (ST-class), 13
  aggregate (over-methods), 11
  aggregate.ST-method (over-methods), 11
  aggregate.zoo, 11
  air, 2
  as.data.frame, 32
  as.data.frame.STF (STFDF-class), 18
  as.data.frame.STFDF (unstack), 32
  as.data.frame.STIF (STIDF-class), 20
  as.data.frame.STIDF (STIDF-class), 20
  as.data.frame.STSD (STSDF-class), 25
  as.data.frame.STSDF (STSDF-class), 25
  as.yearmon, 11
  as.zoo (STFDF-class), 18
  bbox, 15
  bbox (stbox), 15
  bbox, ST-method (stbox), 15
  cbind.ST (ST-class), 13
  coerce,ltraj,STTDF-method (STTDF-class), 27
  coerce,STDF,Spatial-method (STDF-class), 18
  coerce,STDF,xts-method (STFDF-class), 18
  coerce,STIDF,STSDF-method (STIDF-class), 20
  coerce,STSDF,STTDF-method (STTDF-class), 27
  colMeans, 22
  coordinates, 24
  cov, 7
  CRS-class, 16
  dates (air), 2
  DE (air), 2
  DE_NUTSI (air), 2
INDEX

delta, 3, 18, 26
dim.ST (ST-class), 13
EOF, 4
eof (EOF), 4
fires, 5
geometry, ST-method (ST-class), 13
gameometry, STFDF-method (STFDF-class), 18
gameometry, STI-method (STIDF-class), 20
gameometry, STIDF-method (STIDF-class), 20
gameometry, STSDF-method (STSDF-class), 25
gameometry, STTDF-method (STTDF-class), 27
index (timeMatch), 30
interval_overlap, 30, 31
is.projected, ST-method (ST-class), 13
levelplot, 23
ltraj-class (STTDF-class), 27
MATCH (timeMatch), 30
match, 30
mean, 22
mnf, 6
na.approx, 9
na.approx (na.locf), 8
na.locf, 8, 9
na.omit.STFDF (STFDF-class), 18
na.spline, 9
na.spline (na.locf), 8
nbMult, 10
over, 12, 30, 31
over (over-methods), 11
over, ST, STS-method (over-methods), 11
over, ST, STFDF-method (over-methods), 11
over, ST, STI-method (over-methods), 11
over, ST, STIDF-method (over-methods), 11
over, ST, STSDF-method (over-methods), 11
over, ST, STTDF-method (over-methods), 11
over, STI, ST-method (over-methods), 11
over, STI, STFDF-method (over-methods), 11
over, STI, STI-method (over-methods), 11
over, STI, STIDF-method (over-methods), 11
over, STI, STS-method (over-methods), 11
over, STI, STSDF-method (over-methods), 11
over, STI, STTDF-method (over-methods), 11
over, STS, ST-method (over-methods), 11
over, STS, STFDF-method (over-methods), 11
over, STS, STI-method (over-methods), 11
over, STS, STIDF-method (over-methods), 11
over, STS, STSDF-method (over-methods), 11
over, STS, STTDF-method (over-methods), 11
over, xts, xts-method (over-methods), 11
over-methods, 11
plot, ST, missing-method (STTDF-class), 18
plot, STF, missing-method (STFDF-class), 18
plot, STI, missing-method (STIDF-class), 20
plot, STS, missing-method (STSDF-class), 25
plot, STSDF, missing-method (STSDF-class), 25
plot, STT, missing-method (STTDF-class), 27
POSIXct, 21
prcomp, 4, 7
proj4string, ST-method (ST-class), 13
proj4string<-, ST, character-method (ST-class), 13
proj4string<-, ST, CRS-method (ST-class), 13
rbind.STFDF (STFDF-class), 18
rbind.STIDF (STIDF-class), 20
rbind.STSDF (STSDF-class), 25
read.tgrass, 12
rowMeans, 22
rural (air), 2
segPanel (stplot), 23
Spatial, 4, 13, 18, 21, 25
Spatial-class, 16
SpatialPoints, 13
SpatialPolygons, 13
spplot, 24
ST, 18
ST (ST-class), 13
ST-class, 13, 19, 21, 25, 26
stack.STFDF (stplot), 23
stack.STIDF (stplot), 23
stack.STSDF (stplot), 23
stations (air), 2
INDEX

stbox, 15
stbox, ST-method (stbox), 15
stconstruct, 15
STF (STDF-class), 18
STF-class (STDF-class), 18
STDF, 11, 22, 25
STDF (STDF-class), 18
STDF-class, 18
STI, 25, 28
STI (STIDF-class), 20
STI-class (STIDF-class), 20
STIDF, 25
STIDF (STIDF-class), 20
STIDF-class, 20
stInteraction, 22
stplot, 23
stplot, RasterStackBrick-method (stplot), 23
stplot, STDF-method (stplot), 23
stplot, STI-method (stplot), 23
stplot, STIDF-method (stplot), 23
stplot, STSDF-method (stplot), 23
stplot, STT-method (stplot), 23
stplot, STTDF-method (stplot), 23
stplot, STDF (stplot), 23
stplot, STIDF (stplot), 23
STS (STSDF-class), 25
STS-class (STSDF-class), 25
STSDF, 25
STSDF (STSDF-class), 25
STSDF-class, 25
STT, 25
STT (STTDF-class), 27
STT-class (STTDF-class), 27
STTDF, 25
STTDF (STTDF-class), 27
STTDF-class, 27

timeIsInterval <-, STT, logical-method (timeIsInterval), 29
timeMatch, 11, 12, 30
timeMatch, Date, Date-method (timeMatch), 30
timeMatch, POSIXct, POSIXct-method (timeMatch), 30
timeMatch, ST, ST-method (timeMatch), 30
timeMatch, xts, xts-method (timeMatch), 30
tracksPanel (stplot), 23
unstack, 32, 32
unstack, STDF (unstack), 32
write.tgrass (read.tgrass), 12
writeRaster, 12
xts, 4, 13, 16, 19, 21, 26
xts-class (ST-class), 13
xyplot, 24
zoo-class (ST-class), 13