Package 'spacetime'

January 21, 2020

Version 1.2-3

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

Collate Class-xts.R Class-ST.R Class-STFDF.R Class-STSDF.R Class-STIDF.R Class-STTDF.R interval.R endtime.R ST-methods.R STFDF-methods.R STSDF-methods.R STIDF-methods.R STTDF-methods.R apply.R coerce.R stconstruct.R plot.R stplot.R timematch.R over.R aggregate.R eof.R mnf.R bind.R na.R raster.R tgrass.R stinteraction.R

NeedsCompilation no

Author Edzer Pebesma [aut, cre] (<https://orcid.org/0000-0001-8049-7069>), Benedikt Graeler [ctb], Tom Gottfried [ctb], Robert J. Hijmans [ctb]

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

Repository CRAN

Date/Publication 2020-01-21 10:50:02 UTC

R topics documented:

`air` *Air quality data, rural background PM10 in Germany, daily averages 1998-2009*

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 Bundeslaender) 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/ .

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)
```

EOF *Compute spatial or temporal empirical orthogonal function (EOF)*

Description

Compute spatial or temporal empirical orthogonal function (EOF)

Usage

```
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

```
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"
```

```
# 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 firt 500 time steps, to limit run time:
wind.st = wind.st[,1:500]
wind.eof.1 = eof(wind.st)
wind.eof.2 = eof(wind.st, "temporal")
wind.eof.1.PCs = eof(wind.st, returnEOFs = FALSE)
eof(wind.st, "temporal", returnEOFs = FALSE)
summary(eof(wind.st, returnEOFs = FALSE))
summary(eof(wind.st, "temporal", returnEOFs = FALSE))
plot(eof(wind.st, "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))
}
```

---

mnf *Generic mnf method*

Description

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

Usage

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

```
## S3 method for class 'STFDF'
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 noice 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)
m
summary(m)
plot(predict(m))

# spatial example:
## Not run:
library(sp)
grd = SpatialPoints(expand.grid(x=1:100, y=1:100))
gridded(grd) = TRUE
fullgrid(grd) = TRUE
pts = spsample(grd, 50, "random")
```

```
pts$z = rnorm(50)
library(gstat)
v = vgm(1, "Sph", 90)
out = krige(z~1, pts, grd, v, nmax = 20, nsim = 4)
out[[3]] = 0.5 * out[[3]] + 0.5 * rnorm(1e4)
out[[4]] = rnorm(1e4)
spplot(out, as.table = TRUE)
m = mnf(out)
m
summary(m)

## End(Not run)
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"

 # 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, interval = TRUE)
 m = mnf(wind.st)
 m
 plot(m)
 stplot(predict(m), mode = "tp")
}
```

`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

```
# toy example:
library(sp)
pts = SpatialPoints(cbind(c(0,1),c(0,1)))
Sys.setenv(TZ="GMT")
tm = seq(as.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,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(PM10 = 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

Giovana M. de Espindola, Edzer Pebesma, Gilberto C\^amara, 2011. Spatio-temporal regression models for deforestation in the Brazilian Amazon. STDM 2011, The International Symposium on Spatial-Temporal Analysis and Data Mining, University College London - 18th-20th July 2011.

Value

object of class nb

Author(s)

Edzer Pebesma

over-methods *consistent spatio-temporal overlay for objects inheriting from ST*

Description

consistent spatio-temporal overlay for STF, 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

```
## 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"

x = "xts", y = "xts" 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 SpatialPolygons, 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

```
## 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

`sp`	an object deriving from class Spatial, such as a SpatialPoints or SpatialPolygons
`time`	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
`endTime`	vector of class `POSIXct` holding end points of time intervals

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 colunns; 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

```
# 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 = STFDF(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 = rgdal::readOGR(fname)
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

```
  STF(sp, time, endTime = delta(time))
  STFDF(sp, time, data, endTime = delta(time))
  ## S4 method for signature 'STFDF'
x[i, j, ..., drop = is(x, "STFDF")]
  ## S4 method for signature 'STFDF,xts'
coerce(from, to, strict=TRUE)
  ## S4 method for signature 'STFDF,Spatial'
coerce(from, to)
```

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)
...	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

```
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 = STFDF(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

```
  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)
```

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 correspoinding 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

```
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

```
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 $y_ij=x_ij-m_.jmi./m$ with m the grand mean, $m_.j$ the temporal mean, $m_i.$ the spatial mean and m the grand mean.

Examples

```
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.STFDF(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	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?
range.expand	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 automically plots each time stamp in a panel. The returned value is 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 adviced – no checking is done. The returned value is 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 = "STFDF")`: plots object of class STFDF

stplot `signature(x = "STSDF")`: plots object of class STSDF

stplot `signature(x = "STI")`: plots object of class STI

stplot `signature(x = "STIDF")`: plots object of class STIDF

stplot `signature(x = "STT")`: plots object of class STT

stplot `signature(x = "STTDF")`: plots object of class STTDF

Note

vignette("spacetime") contains several examples

References

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

STSDF-class *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

```
  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 Spatial
`time`	object holding time information; see 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 = "STS",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

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

STTDF-class *Class "STTDF"*

Description

A class for spatio-temporal trajectory data

Usage

```
  ## 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
    l[[i]] = STI(sp, t)
}
stt= STT(l)
```

```
sttdf = STTDF(stt, data.frame(attr = rnorm(n*m), id = paste("ID", rep(1:m, each=n))))
x = as(stt, "STI")
stplot(sttdf, col=1:m, scales=list(draw=TRUE))
stplot(sttdf, by = "id")
stplot(sttdf[1])
stplot(sttdf[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(sttdf[pol])
  names(sttdf[pol]@traj)
  stplot(sttdf[1:2],col=1:2)
  stplot(sttdf[,t0])
  stplot(sttdf[,"2013"])
  stplot(sttdf[pol,"2013"])
  is.null(sttdf[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

```
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`.

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(y) = TRUE
timeMatch(x,y, returnList = FALSE, end.y = delta(y))
timeMatch(x,y, returnList = TRUE, end.y = delta(y))
#timeIsInterval(x) = TRUE
timeMatch(x,y, returnList = FALSE, end.x = delta(x), end.y = delta(y))
timeMatch(x,y, returnList = TRUE, end.x = delta(x), end.y = delta(y))
#timeIsInterval(y) = FALSE
timeMatch(x,y, returnList = FALSE, 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
y[1] = y[2]
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,FALSE),TI(y,TRUE))
#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)
```

```
#timeMatch(TI(x,TRUE), TI(y,TRUE), returnList = TRUE)
```

---

unstack *write STFDF to table forms*

Description

create table forms of STFDF objects

Usage

```
## 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

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

Index