Package ‘stars’

April 23, 2019

Title Spatiotemporal Arrays, Raster and Vector Data Cubes

Version 0.3-1

Description Reading, manipulating, writing and plotting
spatiotemporal arrays (raster and vector data cubes) in 'R', using 'GDAL'
bindings provided by 'sf', and 'NetCDF' bindings by 'ncmeta' and 'RNetCDF'.

License Apache License

URL https://github.com/r-spatial/stars/

BugReports https://github.com/r-spatial/stars/issues/

Additional_repositories http://pebesma.staff.ifgi.de

Depends R (>= 3.3.0), abind, sf (>= 0.7-3)

Imports methods, parallel, classInt (>= 0.2-2), rlang, units

Suggests PCICt, RNetCDF (>= 1.8-2), covr, dplyr (>= 0.7-0),
future.apply, ggforce, ggplot2, ggthemes, gstat, httr,
jsonlite, knitr, lwgeom, maps, ncmeta (>= 0.0.3), pbapply, plm,
raster, rmarkdown, sp, spacetime, testthat, viridis, xts, zoo,
starsdata

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 6.1.1

Collate 'init.R' 'stars.R' 'read.R' 'sf.R' 'dimensions.R' 'values.R'
'plot.R' 'tidyverse.R' 'transform.R' 'ops.R' 'write.R'
'raster.R' 'sp.R' ' spacetime.R' 'ncdf.R' 'proxy.R' 'factors.R'
'rasterize.R' 'subset.R' 'warp.R' 'aggregate.R' 'xts.R'
'intervals.R' 'geom.R'

NeedsCompilation no

Author Edzer Pebesma [aut, cre] (<https://orcid.org/0000-0001-8049-7069>),
Michael Sumner [ctb] (<https://orcid.org/0000-0002-2471-7511>),
Etienne Racine [ctb],
Adriano Fantini [ctb],
David Blodgett [ctb]
Maintainer  Edzer Pebesma <edzer.pebesma@uni-muenster.de>
Repository  CRAN
Date/Publication  2019-04-23 12:30:03 UTC

R topics documented:

aggregate.stars . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
as . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
c.stars . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
cut_stars . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
dplyr . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
geom_stars . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
make_intervals . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
ops_stars . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
plot.stars . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
read_ncdf . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
read_stars . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
redimension . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
stars_subset . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14
st_apply . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
st_as_sf . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
st_as_stars . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
st_contour . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
st_coordinates . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
st_crop . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
st_dimensions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
st_rasterize . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24
st_transform . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25
st_warp . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26
st_xy2sfc . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
write_stars . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28

Index  29

aggregate.stars  spatially or temporally aggregate stars object

Description

spatially or temporally aggregate stars object, returning a data cube with lower spatial or temporal resolution

Usage

## S3 method for class 'stars'
aggregate(x, by, FUN, ..., drop = FALSE,
  join = st_intersects, as_points = any(st_dimension(by) == 2, na.rm =
  TRUE), rightmost.closed = FALSE)
Arguments

x  object of class stars with information to be aggregated
by object of class sf, sfc, or a time class (Date, POSIXct, or PCICt) with aggregation geometry/time periods; if of class stars, it is converted to sfc by st_as_sfc(by, as_points = FALSE)
FUN aggregation function, such as mean
... arguments passed on to FUN, such as na.rm=TRUE
drop logical; ignored
join join function to find matches of x to by
as_points see st_as_sf: shall raster pixels be taken as points, or small square polygons?
rightmost.closed see findInterval

---

as Coerce stars object into a Raster raster or brick

Description

Coerce stars object into a Raster raster or brick

Arguments

from object to coerce

c.stars combine multiple stars objects, or combine multiple attributes in a single stars object into a single array

Description

combine multiple stars objects, or combine multiple attributes in a single stars object into a single array

Usage

## S3 method for class 'stars'
c(..., along = NA_integer_)

Arguments

... object(s) of class star: in case of multiple arguments, these are combined into a single stars object, in case of a single argument, its attributes are combined into a single attribute. In case of multiple objects, all objects should have the same dimensionality.
along integer; see read_stars
**Examples**

tif = system.file("tif/L7_ETMs.tif", package = "stars")
\nx = read_stars(tif)
\n(new = c(x, x))
\nc(new) # collapses two arrays into one with an additional dimension
\nc(x, x, along = 3)

---

**cut stars**

**cut methods for stars objects**

**Description**

cut methods for stars objects

**Usage**

### S3 method for class 'array'

cut(x, breaks, …)

### S3 method for class 'matrix'

cut(x, breaks, …)

### S3 method for class 'stars'

cut(x, breaks, …)

**Arguments**

- **x** see cut
- **breaks** see cut
- **...** see cut

**Details**

R’s factor only works for vectors, not for arrays or matrices. This is a work-around (or hack?) to keep the factor levels generated by `cut` and use them in plots.

**Value**

an array or matrix with a `levels` attribute; see details

**Examples**

tif = system.file("tif/L7_ETMs.tif", package = "stars")
\nx = read_stars(tif)
\ncut(x, c(0, 50, 100, 255))
\ncut(x[,1], c(0, 50, 100, 255))
\nplot(cut(x[,1], c(0, 50, 100, 255)))
\ntif = system.file("tif/L7_ETMs.tif", package = "stars")
x1 = read_stars(tif)
  (x1_cut = cut(x1, breaks = c(0, 50, 100, Inf)))  # shows factor in summary
plot(x1_cut[, , c(3, 6)])  # propagates through [] and plot

dplyr verbs for stars objects

Description

dplyr verbs for stars objects

Usage

filter.stars(.data, ...)

mutate.stars(.data, ...)

select.stars(.data, ...)

pull.stars(.data, var = -1)

as.tbl_cube.stars(x, ...)

slice.stars(.data, along, index, ..., drop = length(index) == 1)

Arguments

.data object of class stars

... see filter

var see pull

x object of class stars

along name or index of dimension to which the slice should be applied

index integer value(s) for this index

drop logical; drop dimensions that only have a single index?

Examples

tif = system.file("tif/L7_ETMs.tif", package = "stars")
x1 = read_stars(tif)
library(dplyr)
x1 %>% slice("band", 2:3)
x1 %>% slice("x", 50:100)
geom_stars

*ggplot geom for stars objects*

**Description**

*ggplot geom for stars objects*

**Usage**

```
geom_stars(mapping = NULL, data = NULL, ..., downsample = 1,
  sf = FALSE)
```

**Arguments**

- `mapping`: see `geom_raster`
- `data`: see `geom_raster`
- `...`: see `geom_raster`
- `downsample`: downsampling rate: e.g. 3 keeps rows and cols 1, 4, 7, 10 etc.; a value of 1 does not downsample
- `sf`: logical; if TRUE rasters will be converted to polygons and plotted using `geom_sf`.

**Details**

`geom_stars` returns (a call to) either `geom_raster`, `geom_tile`, or `geom_sf`, depending on the raster or vector geometry; for the first to, an `aes` call is constructed with the raster dimension names and the first array as fill variable. Further calls to `coord_equal` and `facet_wrap` are needed to control aspect ratio and the layers to be plotted; see examples.

**Examples**

```r
system.file("tif/L7_ETMs.tif", package = "stars") %>% read_stars() -> x
library(ggplot2)
ggplot() + geom_stars(data = x) +
  coord_equal() +
  facet_wrap(~band) +
  theme_void() +
  scale_x_discrete(expand=c(0,0))+
  scale_y_discrete(expand=c(0,0))
```
**make_intervals**

create an intervals object

**Description**

create an intervals object, assuming left-closed and right-open intervals

**Usage**

make_intervals(start, end)

**Arguments**

- **start**: vector with start values, or 2-column matrix with start and end values in column 1 and 2, respectively
- **end**: vector with end values

**ops_stars**

*S3 Ops Group Generic Functions for stars objects*

**Description**

Ops functions for stars objects, including comparison, product and divide, add, subtract

**Usage**

```r
## S3 method for class 'stars'
Ops(e1, e2)
```

```r
## S3 method for class 'stars'
Math(x, ...)
```

```r
## S3 method for class 'stars_proxy'
Ops(e1, e2)
```

```r
## S3 method for class 'stars_proxy'
Math(x, ...)
```

**Arguments**

- **e1**: object of class stars
- **e2**: object of class stars
- **x**: object of class stars
- **...**: parameters passed on to the Math functions
plot.stars

Value

object of class stars

Examples

tif = system.file("tif/L7_ETMs.tif", package = "stars")
x = read_stars(tif)
x * x
x / x
x + x
x + 10
all.equal(x * 10, 10 * x)
tif = system.file("tif/L7_ETMs.tif", package = "stars")
x = read_stars(tif)
a = sqrt(x)
b = log(x, base = 10)

plot.stars  plot stars object, with subplots for each level of first non-spatial dimension

Description

plot stars object, with subplots for each level of first non-spatial dimension

Usage

## S3 method for class 'stars'
plot(x, y, ..., join_zlim = TRUE, main = names(x)[1],
     axes = FALSE, downsample = TRUE, nbreaks = 11,
     breaks = "quantile", col = grey(1:(nbreaks - 1)/nbreaks),
     key.pos = get_key_pos(x, ...), key.width = lcm(1.8),
     key.length = 0.618, reset = TRUE, box_col = grey(0.8),
     center_time = FALSE)

## S3 method for class 'stars_proxy'
plot(x, y, ...,
     downsample = get_downsample(dim(x)))
Arguments

- **x**: object of class `stars`
- **y**: ignored
- **...**: further arguments: for `plot`, passed on to `image.stars`; for `image`, passed on to `image.default` or `rasterImage`.
- **join_zlim**: logical; if `TRUE`, compute a single, joint `zlim` (color scale) for all subplots from `x`.
- **main**: character; subplot title prefix; use "" to get only time, use NULL to suppress subplot titles.
- **axes**: logical; should axes and box be added to the plot?
- **downsample**: logical; if `TRUE` will try to plot not many more pixels than actually are visible.
- **nbreaks**: number of color breaks; should be one more than number of colors. If missing and `col` is specified, it is derived from that.
- **breaks**: actual color breaks, or a method name used for `classIntervals`.
- **col**: colors to use for grid cells.
- **key.pos**: integer; side to plot a color key: 1 bottom, 2 left, 3 top, 4 right; set to NULL to omit key. Ignored if multiple columns are plotted in a single function call. Default depends on plot size, map aspect, and, if set, parameter asp.
- **key.width**: amount of space reserved for width of the key (labels); relative or absolute (using lcm).
- **key.length**: amount of space reserved for length of the key (labels); relative or absolute (using lcm).
- **reset**: logical; if `FALSE`, keep the plot in a mode that allows adding further map elements; if `TRUE` restore original mode after plotting; see details.
- **box_col**: color for box around sub-plots; use 0 to suppress plotting of boxes around sub-plots.
- **center_time**: logical; if `TRUE`, sub-plot titles will show the center of time intervals, otherwise their start.
- **band**: integer; which band (dimension) to plot.
- **attr**: integer; which attribute to plot.
- **asp**: numeric; aspect ratio of image.
- **rgb**: integer; specify three bands to form an rgb composite. Experimental: rgb color table; see Details.
- **maxColorValue**: numeric; passed on to `rgb`.
- **xlab**: character; x axis label.
- **ylab**: character; y axis label.
- **xlim**: x axis limits.
- **ylim**: y axis limits.
- **text_values**: logical; print values as text on image?
- **interpolate**: logical; when using `rasterImage` (rgb), should pixels be interpolated?
as_points logical; for curvilinear or sheared grids: parameter passed on to st_as_sf, determining whether raster cells will be plotted as symbols (fast, approximate) or small polygons (slow, exact)

logz logical; if TRUE, use log10-scale for the attribute variable. In that case, breaks and at need to be given as log10-values; see examples.

Details

use of an rgb color table is experimental; see https://github.com/r-spatial/mapview/issues/208

when plotting a subsetted stars_proxy object, the default value for argument downsample will not be computed correctly, and it and has to be set manually.

Examples

tif = system.file("tiff/L7_ETMs.tif", package = "stars")
x = read_stars(tif)
image(x, col = grey((3:9)/10))
image(x, rgb = c(1,3,5)) # rgb composite

---

read_ncdf

Read NetCDF into stars object

Description

Read data from a file (or source) using the NetCDF library directly.

Usage

read_ncdf(.x, ..., var = NULL, nsub = NULL,
curvilinear = character(0), eps = 1e-12)

Arguments

.x NetCDF file or source

... ignored

var variable name or names (they must be on matching grids)

ncsub matrix of start, count columns (see Details)

curvilinear length two character vector with names of subdatasets holding longitude and latitude values for all raster cells.

eps numeric; dimension value increases are considered identical when they differ less than eps
Details

The following logic is applied to coordinates. If any coordinate axes have regularly spaced coordinate variables they are reduced to the offset/delta form with \( \text{affine} = c(0, 0) \), otherwise the values of the coordinates are stored and used to define a rectilinear grid.

If the data has two or more dimensions and the first two are regular they are nominated as the ‘raster’ for plotting.

If the curvilinear argument is used it specifies the 2D arrays containing coordinate values for the first two dimensions of the data read. It is currently assumed that the coordinates are 2D and that they relate to the first two dimensions in that order.

If var is not set the first set of variables on a shared grid is used.

start and count columns of ncsb must correspond to the variable dimension (nrows) and be valid index using var.get.nc convention (start is 1-based). If the count value is NA then all steps are included. Axis order must match that of the variable/s being read.

Examples

```r
f <- system.file("nc/reduced.nc", package = "stars")
read_ncdf(f)
read_ncdf(f, var = c("anom"))
read_ncdf(f, ncsb = cbind(start = c(1, 1, 1, 1), count = c(10, 12, 1, 1)))
```

```r
#' precipitation data in a curvilinear NetCDF
prec_file = system.file("nc/test_stageiv_xyt.nc", package = "stars")
prec = read_ncdf(prec_file, curvilinear = c("lon", "lat"))
```

```r
# plot(prec) ## gives error about unique breaks
## remove NAs, zeros, and give a large number
## of breaks (used for validating in detail)
qu_0.omit = function(x, ..., n = 22) {
  x = na.omit(x)
  c(0, quantile(x[x > 0], seq(0, 1, length.out = n)))
}
library(dplyr)
prec_slice = slice(prec, index = 17, along = "time")
plot(prec_slice, border = NA, breaks = qu_0.omit(prec_slice[[1]]), reset = FALSE)
nc = sf::read_sf(system.file("gpkg/nc.gpkg", package = "sf"), "nc.gpkg")
plot(st_geometry(nc), add = TRUE, reset = FALSE, col = NA)
```

Description

read raster/array dataset from file or connection
read_stars

Arguments

.x character vector with name(s) of file(s) or data source(s) to be read
... passed on to st_as_stars if curvilinear was set
options character; opening options
driver character; driver to use for opening file. To override fixing for subdatasets and autodetect them as well, use NULL.
sub integer or logical; sub-datasets to be read
quiet logical; print progress output?
NA_value numeric value to be used for conversion into NA values; by default this is read from the input file
along length-one character or integer, or list; determines how several arrays are combined, see Details.
RasterIO list with named parameters for GDAL’s RasterIO, to further control the extent, resolution and bands to be read from the data source; see details.
proxy logical; if TRUE, an object of class stars_proxy is read which contains array metadata only; if FALSE the full array data is read in memory.
curvilinear length two character vector with names of subdatasets holding longitude and latitude values for all raster cells.
normalize_path logical; if FALSE, suppress a call to normalizePath on .x

Details

In case .x contains multiple files, they will all be read and combined with c.stars. Along which dimension, or how should objects be merged? If along is set to NA it will merge arrays as new attributes if all objects have identical dimensions, or else try to merge along time if a dimension called time indicates different time stamps. A single name (or positive value) for along will merge along that dimension, or create a new one if it does not already exist. If the arrays should be arranged along one of more dimensions with values (e.g. time stamps), a named list can passed to along to specify them; see example.

RasterIO is a list with zero or more of the following named arguments: nXOff, nYOff (both 1-based: the first row/col has offset value 1), nXSize, nYSize, nBufXSize, nBufYSize, bands, coderesample. see https://www.gdal.org/classGDALDataset.html#a80d005ed10aefafa8a55dc539c2f69da for their meaning; bands is an integer vector containing the band numbers to be read (1-based: first band is 1) Note that if nBufXSize or nBufYSize are specified for downsampling an image, resulting in an adjusted geotransform. resample reflects the resampling method and has to be one of: "nearest_neighbour" (the default), "bilinear", "cubic", "cubic_spline", "lanczos", "average", "mode", or "Gauss".
redimension

Value

object of class stars

Examples

tif = system.file("tif/L7_ETMs.tif", package = "stars")
(x1 = read_stars(tif))
(x2 = read_stars(c(tif, tif)))
(x3 = read_stars(c(tif, tif), along = "band"))
(x4 = read_stars(c(tif, tif), along = "new_dimensions")) # create 4-dimensional array
x1o = read_stars(tif, options = "OVERVIEW_LEVEL=1")
t1 = as.Date("2018-07-31")
# along is a named list indicating two dimensions:
read_stars(c(tif, tif, tif, tif), along = list(foo = c("bar1", "bar2"), time = c(t1, t1+2)))

m = matrix(1:120, nrow = 12, ncol = 10)
dim(m) = c(x = 10, y = 12) # named dim
st = st_as_stars(m)
attr(st, "dimensions")$y$delta = -1
attr(st, "dimensions")$y$offset = 12
st
tmp = tempfile(fileext = ".tif")
write_stars(st, tmp)
(red <- read_stars(tmp))
read_stars(tmp, RasterIO = list(nXOff = 1, nYOff = 1, nXsize = 10, nYSize = 12,
 nBufXSize = 2, nBufYSize = 2)))[[1]]
(red <- read_stars(tmp, RasterIO = list(nXOff = 1, nYOff = 1, nXsize = 10, nYSize = 12,
 nBufXSize = 2, nBufYSize = 2)))
red[[1]] # cell values of subsample grid:
plot(st, reset = FALSE, axes = TRUE, ylim = c(-1,12.1), xlim = c(-1,10.1),
 main = "nBufXSize & nBufYSize demo", text_values = TRUE)
plot(st_as_sfc(red, as_points = TRUE), add = TRUE, col = 'red', pch = 16)
plot(st_as_sfc(st_as_stars(st), as_points = FALSE), add = TRUE, border = 'grey')
plot(st_as_sfc(red, as_points = FALSE), add = TRUE, border = 'green', lwd = 2)
file.remove(tmp)

redimension
redimension array, or collapse attributes into a new dimension

Description

redimension array, or collapse attributes into a new dimension

Usage

st_redimension(x, new_dims, along, ...)

## S3 method for class 'stars'
st_redimension(x, new_dims = st_dimensions(x),
along = list(new_dim = names(x)), ...)

## S3 method for class 'stars_proxy'
st_redimension(x, new_dims = st_dimensions(x),
   along = list(new_dim = names(x)), ...)

Arguments

x          object of class stars
new_dims   target dimensions
along      named list with new dimension name and values
...         ignored

stars_subset subset stars objects

Description

subset stars objects

Usage

## S3 method for class 'stars'
x[i = TRUE, ..., drop = FALSE,
crop = !is_curvilinear(x)]

## S3 replacement method for class 'stars'
x[i] <- value

Arguments

x          object of class stars
i          first selector: integer, logical or character vector indicating attributes to select,
or object of class sf or sfc used as spatial selector; see details
...        further (logical or integer vector) selectors, matched by order, to select on individual dimensions
drop       logical; if TRUE, degenerate dimensions (with only one value) are dropped
crop       logical; if TRUE and parameter i is a spatial geometry (sf or sfc) object, the extent (bounding box) of the result is cropped to match the extent of i using st_crop. Cropping curvilinear grids is not supported.
value      array of dimensions equal to those in x, or a vector or value that will be recycled to such an array
Details

if i is an object of class sf, sfc or bbox, the spatial subset covering this geometry is selected, possibly followed by cropping the extent. Array values for which the cell centre is not inside the geometry are assigned NA.

in an assignment (or replacement form, [<-]), argument i needs to be a stars object with dimensions identical to x, and value will be recycled to the dimensions of the arrays in x.

Examples

tif = system.file("tif/L7_ETMs.tif", package = "stars")
x = read_stars(tif)
x[,3:4] # select bands
x[,1:100,100:200,] # select x and y by range
x["L7_ETMs.tif"] # select attribute
xy = structure(list(x = c(293253.999046018, 296400.196497684), y = c(9113801.64775462, 911128.49619133)), c("x", "y"))
pts = st_as_sf(data.frame(do.call(cbind, xy)), coords = c("x", "y"), crs = st_crs(x))
image(x, axes = TRUE)
plot(st_as_sfc(st_bbox(pts)), col = NA, add = TRUE)
bb = st_bbox(pts)
(xx = x[bb])
image(xx)
plot(st_as_sfc(bb), add = TRUE, col = NA)
image(x)
pt = st_point(c(x = 290462.103109179, y = 9114202.32594085))
buf = st_buffer(st_sfc(pt, crs = st_crs(x)), 1500)
plot(buf, add = TRUE)
buf = st_sfc(st_polygon(list(st_buffer(pt, 1500)[[1]], st_buffer(pt, 1000)[[1]])),
crs = st_crs(x))
image(x[buf])
plot(buf, add = TRUE, col = NA)
image(x[buf, crop=FALSE])
plot(buf, add = TRUE, col = NA)

Description

st_apply apply a function to one or more array dimensions

Usage

## S3 method for class 'stars'
st_apply(X, MARGIN, FUN, ..., CLUSTER = NULL,
   PROGRESS = FALSE, FUTURE = FALSE, rename = TRUE)
st_as_sf

Convert stars object into an sf object

Description

Convert stars object into an sf object

Usage

```r
## S3 method for class 'stars'
st_as_sfc(x, ..., as_points,
    which = seq_len(prod(dim(x)[1:2])))

## S3 method for class 'stars'
st_as_sf(x, ..., as_points = FALSE, merge = FALSE,
    na.rm = TRUE, use_integer = is.logical(x[[1]]) || is.integer(x[[1]]),
    long = FALSE, connect8 = FALSE)
```
st_as_stars

Arguments

- **x**: object of class stars
- **as_points**: logical; should cells be converted to points or to polygons? See details.
- **which**: linear index of cells to keep (this argument is not recommended to be used)
- **merge**: logical; if TRUE, cells with identical values are merged (using GDAL_Polygonize or GDAL_FPolygonize); if FALSE, a polygon for each raster cell is returned; see details
- **na.rm**: logical; should missing valued cells be removed, or also be converted to features?
- **use_integer**: (relevant only if **merge** is TRUE): if TRUE, before polygonizing values are rounded to 32-bits signed integer values (GDALPolygonize), otherwise they are converted to 32-bit floating point values (GDALFPolygonize).
- **long**: logical; if TRUE, return a long table form sf, with geometries and other dimensions recycled
- **connect8**: logical; if TRUE, use 8 connectedness. Otherwise the 4 connectedness algorithm will be applied.

Details

If **merge** is TRUE, only the first attribute is converted into an sf object. If **na.rm** is FALSE, areas with NA values are also written out as polygons. Note that the resulting polygons are typically invalid, and use **st_make_valid** to create valid polygons out of them.

Examples

```r
tif = system.file("tif/L7_ETMs.tif", package = "stars")
x = read_stars(tif)
x = x[,,6] # a band with lower values in it
x[[1]][x[[1]] < 30] = NA # set lower values to NA
x[[1]] = x[[1]] < 100 # make the rest binary
x
(p = st_as_sf(x)) # removes NA areas
(p = st_as_sf(x[,,1], merge = TRUE)) # glues polygons together
all(st_is_valid(p)) # not all valid, see details
# plot(p, axes = TRUE)
(p = st_as_sf(x, na.rm = FALSE, merge = TRUE)) # includes polygons with NA values
# plot(p, axes = TRUE)
```

---

**st_as_stars**

*convert objects into a stars object*

Description

*convert objects into a stars object*
Usage

```
st_as_stars(.x, ...)  
## S3 method for class 'list'
st_as_stars(.x, ..., dimensions = NULL)

## Default S3 method:
st_as_stars(.x = NULL, ..., raster = NULL)

## S3 method for class 'stars'
st_as_stars(.x, ..., curvilinear = NULL, 
            crs = st_crs(4326))

## S3 method for class 'bbox'
st_as_stars(.x, ..., nx = 360, ny = 180, 
            deltax = diff(xlim)/nx, deltay = -diff(ylim)/ny, 
            xlim = .x[c("xmin", "xmax")], ylim = .x[c("ymin", "ymax")], 
            values = 0)

## S3 method for class 'sf'
st_as_stars(.x, ..., name = attr(.x, "sf_column"))

## S3 method for class 'Raster'
st_as_stars(.x, ...)

## S3 method for class 'stars_proxy'
st_as_stars(.x, ..., downsample = 0, 
            url = attr(.x, "url"), env = parent.frame())
```

Arguments

- `.x` object to convert
- `...` ignored
- `dimensions` object of class dimensions
- `raster` character; the names of the dimensions that denote raster dimensions
- `curvilinear` only for creating curvilinear grids: named length 2 list holding longitude and latitude matrices; the names of this list should correspond to raster dimensions to be replaced
- `crs` object of class `crs` with the coordinate reference system of the values in `curvilinear`; see details
- `nx` integer; number of cells in x direction
- `ny` integer; number of cells in y direction
- `deltax` numeric; cell size in x direction
- `deltay` numeric; cell size in y direction (negative)
- `xlim` length 2 numeric vector with extent (min, max) in x direction
**st_contour**

*Compute contour lines or sets*

### Description

Compute contour lines or sets

### Usage

```r
st_contour(x, na.rm = TRUE, contour_lines = FALSE,
            breaks = classInt::classIntervals(na.omit(as.vector(x[[1]])))$brks)
```

### Arguments

- `x` object of class stars
- `na.rm` logical; should missing valued cells be removed, or also be converted to features?
- `contour_lines` logical; if FALSE, polygons are returned (contour sets), otherwise contour lines
- `breaks` numerical; values at which to "draw" contour levels

### Details

- this function requires GDAL >= 2.4.0

### See Also

- for polygonizing rasters following grid boundaries, see `st_as_sf` with arguments `as_points=FALSE` and `merge=TRUE`
**st_coordinates** retrieve coordinates for raster or vector cube cells

**Description**

retrieve coordinates for raster or vector cube cells

**Usage**

```r
## S3 method for class 'stars'
st_coordinates(x, ..., add_max = FALSE, center = TRUE)
```

```r
## S3 method for class 'stars'
as.data.frame(x, ..., add_max = FALSE, center = NA)

as_tibble.stars(.x, ..., x = FALSE, add_max = FALSE, center = NA)
```

**Arguments**

- **x** object of class stars
- **...** ignored
- **add_max** logical; if TRUE, dimensions are given with a min (x) and max (x_max) value
- **center** logical; (only if add_max is FALSE): should grid cell center coordinates be returned (TRUE) or offset values (FALSE)? center can be a named logical vector or list to specify values for each dimension.
- **.x** object to be converted to a tibble

**st_crop** crop a stars object

**Description**

crop a stars object

**Usage**

```r
## S3 method for class 'stars_proxy'
st_crop(x, y, ..., crop = TRUE, epsilon = 0)
```

```r
## S3 method for class 'stars'
st_crop(x, y, ..., crop = TRUE, epsilon = 0,
  as_points = all(st_dimension(y) == 2, na.rm = TRUE))
```
Arguments

- **x**: object of class `stars`
- **y**: object of class `sf`, `sfc` or `bbox`; see Details below.
- **...**: ignored
- **crop**: logical; if TRUE, the spatial extent of the returned object is cropped to still cover `obj`, if FALSE, the extent remains the same but cells outside `y` are given NA values.
- **epsilon**: numeric; shrink the bounding box of `y` to its center with this factor.
- **as_points**: logical; if FALSE, treat `x` as a set of points, else as a set of small polygons. Default: TRUE if `y` is two-dimensional, else FALSE

Details

For raster, `st_crop` selects cells for which the cell centre is inside the bounding box; see the examples below.

Examples

```r
lW = read_stars(system.file("tif/L7_ETMs.tif", package = "stars"))
d = st_dimensions(lW)

# area around cells 3:10 (x) and 4:11 (y):
offset = c(d[["x"]][offset, d[["y"]][offset])
res = c(d[["x"]][delta, d[["y"]][delta])
bb = st_bbox(c(xmin = offset[1] + 2 * res[1],
ymin = offset[2] + 11 * res[2],
xmax = offset[1] + 10 * res[1],
lW[bb]

plot(lW[,1:13,1:13,1], reset = FALSE)
image(lW[bb,,1], add = TRUE, col = sf.colors())
plot(st_as_sfc(bb), add = TRUE, border = 'green', lwd = 2)

# slightly smaller bbox:
bb = st_bbox(c(xmin = offset[1] + 2.1 * res[1],
ymin = offset[2] + 10.9 * res[2],
xmax = offset[1] + 9.9 * res[1],
lW[bb]

plot(lW[,1:13,1:13,1], reset = FALSE)
image(lW[bb,,1], add = TRUE, col = sf.colors())
plot(st_as_sfc(bb), add = TRUE, border = 'green', lwd = 2)

# slightly larger bbox:
bb = st_bbox(c(xmin = offset[1] + 1.9 * res[1],
xmax = offset[1] + 10.1 * res[1],
ymax = offset[2] + 2.9 * res[2]), crs = st_crs(lW))
lW[bb]
```

```r
plot(l7[,1:13,1:13,1], reset = FALSE)
image(l7[bb,,1], add = TRUE, col = sf.colors())
plot(st_as_sfc(bb), add = TRUE, border = 'green', lwd = 2)

# half a cell size larger bbox:
bb = st_bbox(c(xmin = offset[1] + 1.49 * res[1],
              ymin = offset[2] + 11.51 * res[2],
              xmax = offset[1] + 10.51 * res[1],
              ymax = offset[2] + 2.49 * res[2]), crs = st_crs(l7))

l7[bb]

plot(l7[,1:13,1:13,1], reset = FALSE)
image(l7[bb,,1], add = TRUE, col = sf.colors())
plot(st_as_sfc(bb), add = TRUE, border = 'green', lwd = 2)
```

**st_dimensions**

get dimensions from stars object

**Description**

get dimensions from stars object

**Usage**

```
st_dimensions(.x, ...)
```

## S3 method for class 'stars'
```
st_dimensions(.x, ...)
```

## S3 method for class 'array'
```
st_dimensions(.x, ...)
```

## Default S3 method:
```
st_dimensions(.x, ..., .raster, affine = c(0, 0),
               cell_midpoints = FALSE, point = FALSE)
```

```
st_set_dimensions(.x, which, values = NULL, point = NULL,
                   names = NULL, ...)
```

```
st_get_dimension_values(.x, which, ..., max = FALSE, center = NA)
```

**Arguments**

- `.x` object to retrieve dimensions information from
- `...` further arguments
- `.raster` length 2 character array with names (if any) of the raster dimensions
- `affine` numeric; specify parameters of the affine transformation
cell_midpoints logical; if TRUE AND the dimension values are strictly regular, the values are interpreted as the cell midpoint values rather than the cell offset values when calculating offset (i.e., the half-cell-size correction is applied); can have a value for each dimension, or else is recycled

point logical; does the pixel value (measure) refer to a point (location) value or to an pixel (area) summary value?

which integer or character; index or name of the dimension to be changed

values values for this dimension (e.g. sfc list-column)

names character; new names vector for (all) dimensions, ignoring which

max logical; if TRUE return the end, rather than the beginning of an interval

center logical; if TRUE return the center of an interval; if NA return the center for raster dimensions, and the start of intervals in other cases

Details
dimensions can be specified in two ways. The simplest is to pass a vector with numeric values for a numeric dimension, or character values for a categorical dimension. Parameter cell_midpoints is used to specify whether numeric values refer to the offset (start) of a dimension interval (default), or to the center; the center case is only available for regular dimensions. For rectilinear numeric dimensions, one can specify either a vector with cell borders (start values), or a data.frame with two columns named "start" and "end", with the respective interval start and end values. In the first case, the end values are computed from the start values by assuming the last two intervals have equal width.

Value
the dimensions attribute of x, of class dimensions

Examples

```r
x = read_stars(system.file("tif/L7_ETMs.tif", package = "stars"))
# Landsat 7 ETM+ band semantics: https://landsat.gsfc.nasa.gov/the-enhanced-thematic-mapper-plus/
# set bands to values 1,2,3,4,5,7:
(x1 = st_set_dimensions(x, "band", values = c(1,2,3,4,5,7), names = "band_number", point = TRUE))
# set band values as bandwidth
rbind(c(0.45,0.515), c(0.525,0.605), c(0.63,0.69), c(0.775,0.90), c(1.55,1.75), c(2.08,2.35)) #> units::set_units("um") -> bw # or: units::set_units(um) -> bw
# set bandwidth midpoint:
(x2 = st_set_dimensions(x, "band", values = 0.5 * (bw[,1]+bw[,2]),
  names = "bandwidth_midpoint", point = TRUE))
# set bandwidth intervals:
(x3 = st_set_dimensions(x, "band", values = make_intervals(bw), names = "bandwidth"))
```
Description

rasterize simple feature geometries

Usage

\texttt{st\_rasterize(sf, template = st\_as\_stars(st\_bbox(sf), values = NA\_real\_, ...), file = tempfile(), driver = "GTiff", options = character(0), ...)}

Arguments

- \texttt{sf} object of class \texttt{sf}
- \texttt{template} stars object with desired target geometry
- \texttt{file} temporary file name
- \texttt{driver} driver for temporary file
- \texttt{options} character; options vector for GDALRasterize arguments passed on to \texttt{st\_as\_stars}

Examples

demo(nc, echo = FALSE, ask = FALSE)
(x = st_rasterize(nc)) # default grid:
plot(x, axes = TRUE)
# a bit more customized grid:
(x = st_rasterize(nc, st_as_stars(st_bbox(nc), nx = 100, ny = 50, values = NA_real_)))
plot(x, axes = TRUE)
(ls = st_sf(a = 1:2, st_sfc(st_linestring(rbind(c(0, 1, 0), c(1.1, 1))),
    st_linestring(rbind(c(0, 0.05), c(1, 0.05))))))
(grd = st_as_stars(st_bbox(ls), nx = 10, ny = 10, xlim = c(0, 1.0), ylim = c(0, 1),
    values = NA_real_))
# the following two plots suggests a half-gridcell-shift problem:
sf_extSoftVersion("GDAL")
plot(st_rasterize(ls, grd), axes = TRUE, reset = FALSE) # ALL_TOUCHED=FALSE
plot(ls, add = TRUE, col = "red")
plot(st_rasterize(ls, grd, options = "ALL_TOUCHED=TRUE"), axes = TRUE, reset = FALSE)
plot(ls, add = TRUE, col = "red")
# add lines to existing 0 values, summing values in case of multiple lines:
(grd = st_as_stars(st_bbox(ls), nx = 10, ny = 10, xlim = c(0, 1.0), ylim = c(0, 1), values = 0))
r = st_rasterize(ls, grd, options = c("MERGE_ALG=ADD", "ALL_TOUCHED=TRUE"))
plot(r, axes = TRUE, reset = FALSE)
plot(ls, add = TRUE, col = "red")
**st_transform**

Transfrom features, or warp/resample grids in stars objects to a new coordinate reference system.

**Description**

Transform features, or warp/resample grids in stars objects to a new coordinate reference system.

**Usage**

```r
# S3 method for class 'stars'
st_transform(x, crs, ...)
st_transform_proj.stars(x, crs, ...)
```

**Arguments**

- `x`: object of class `stars`, with either raster or simple feature geometries
- `crs`: object of class `crs` with target CRS
- `...`: ignored

**Details**

For simple feature dimensions, `st_transform` is called, leading to lossless transformation. For gridded spatial data, a curvilinear grid with transformed grid cell (centers) is returned. To convert this to a regular grid in the new CRS, use `st_warp`.

**See Also**

`st_warp`

**Examples**

```r
g = system.file("tif/geomatrix.tif", package = "stars")
(x = read_stars(g))
new = st_crs(4326)
y = st_transform(x, new)
plot(st_transform(st_as_sfc(st_bbox(x)), new), col = NA, border = 'red')
plot(st_as_sfc(y, as_points=FALSE), col = NA, border = 'green', axes = TRUE, add = TRUE)
image(y, col = heat.colors(12), add = TRUE)
plot(st_as_sfc(y, as_points=TRUE), pch=3, cex=.5, col = 'blue', add = TRUE)
plot(st_transform(st_as_sfc(x, as_points=FALSE), new), add = TRUE)
```
**st_warp**  
*Warp (resample) grids in stars objects to a new grid, possibly in an new coordinate reference system*

**Description**

Warp (resample) grids in stars objects to a new grid, possibly in a new coordinate reference system.

**Usage**

```r
st_warp(src, dest, ..., crs = NA_crs_, cellsize = NA_real_, segments = 100, use_gdal = FALSE, options = character(0), no_data_value = NA_real_, debug = FALSE, method = "near")
```

**Arguments**

- **src**: object of class `stars` with source raster
- **dest**: object of class `stars` with target raster geometry
- **...**: ignored
- **crs**: coordinate reference system for destination grid, only used when `dest` is missing
- **cellsize**: cellsize in target coordinate reference system
- **segments**: (total) number of segments for segmentizing the bounding box before transforming to the new `crs`
- **use_gdal**: logical; if TRUE, use gdalwarp, through `gdal_utils`
- **options**: character vector with options, passed on to gdalwarp
- **no_data_value**: value used by gdalwarp for no_data (NA) when writing to temporary file
- **debug**: logical; if TRUE, do not remove the temporary gdalwarp destination file, and print its name
- **method**: character; see details for options; methods other than `near` only work when `use_gdal`=TRUE

**Details**

- method should be one of near, bilinear, cubic, cubicspline, lanczos, average, mode, max, min, med, q1 or q3; see https://github.com/r-spatial/stars/issues/109

For gridded spatial data (dimensions x and y), see figure; the existing grid is transformed into a regular grid defined by `dest`, possibly in a new coordinate reference system. If `dest` is not specified, but `crs` is, the procedure used to choose a target grid is similar to that of `projectRaster` (currently only with method='nbg'). This entails: (i) the envelope (bounding box polygon) is transformed into the new `crs`, possibly after segmentation (red box); (ii) a grid is formed in this new `crs`, touching the transformed envelope on its East and North side, with (if `cellsize` is not given) a `cellsize` similar to the cell size of `src`, with an extent that at least covers `x`; (iii) for each cell center of this new grid, the matching grid cell of `x` is used; if there is no match, an NA value is used.
Examples

```r
geomatrix = system.file("tif/geomatrix.tif", package = "stars")
(x = read_stars(geomatrix))
new_crs = st_crs(4326)
y = st_warp(x, crs = new_crs)
plot(st_transform(st_as_sfc(st_bbox(x)), new_crs), col = NA, border = 'red')
plot(st_as_sfc(y, as_points=FALSE), col = NA, border = 'green', axes = TRUE, add = TRUE)
image(y, add = TRUE, nbreaks = 6)
plot(st_as_sfc(y, as_points=TRUE), pch=3, cex=-.5, col = 'blue', add = TRUE)
plot(st_transform(st_as_sfc(x, as_points=FALSE), new_crs), add = TRUE)
```

---

**st_xy2sfc**

*replace x y raster dimensions with simple feature geometry list (points, or polygons = rasterize)*

Description

replace x y raster dimensions with simple feature geometry list (points, or polygons = rasterize)

Usage

```r
st_xy2sfc(x, as_points, ..., na.rm = TRUE)
```

Arguments

- `x` object of class stars
- `as_points` logical; if TRUE, generate points at cell centers, else generate polygons
- `...` arguments passed on to `st_as_sfc`
- `na.rm` logical; remove cells with all missing values?

Value

object of class stars with x and y raster dimensions replaced by a single sfc geometry list column containing either points or square polygons
write_stars

**write stars object to gdal dataset (typically: to file)**

**Description**
write stars object to gdal dataset (typically: to file)

**Usage**

```r
write_stars(obj, dsn, layer, ...)  

## S3 method for class 'stars'
write_stars(obj, dsn, layer = 1, ...,  
  driver = detect.driver(dsn), options = character(0),  
  type = "Float32", NA_value = NA_real_, update = FALSE)

## S3 method for class 'stars_proxy'
write_stars(obj, dsn, layer = 1, ...,  
  driver = detect.driver(dsn), options = character(0),  
  type = "Float32", NA_value = NA_real_, chunk_size = c(dim(obj)[1], 
  floor(2.5e+07/dim(obj)[1])), progress = TRUE)
```

**Arguments**

- **obj**: object of class `stars`
- **dsn**: gdal dataset (file) name
- **layer**: attribute name; if missing, the first attribute is written
- **...**: passed on to `gdal_write`
- **driver**: driver driver name; see `st_drivers`
- **options**: character vector with options
- **type**: character; output binary type, one of: `Byte` for eight bit unsigned integer, `UInt16` for sixteen bit unsigned integer, `Int16` for sixteen bit signed integer, `UInt32` for thirty two bit unsigned integer, `Int32` for thirty two bit signed integer, `Float32` for thirty two bit floating point, `Float64` for sixty four bit floating point.
- **NA_value**: non-NA value that should represent R’s NA value in the target raster file; if set to NA, it will be ignored.
- **update**: logical; if TRUE, an existing file is being updated
- **chunk_size**: length two integer vector with the number of pixels (x, y) used in the read/write loop; see details.
- **progress**: logical; if TRUE, a progress bar is shown

**Details**
write_stars first creates the target file, then updates it sequentially by writing blocks of chunk_size.
Index

[.stars(stars_subset), 14
[<-.stars(stars_subset), 14

aes, 6
aggregate.stars, 2
apply, 16
as, 3
as.data.frame.stars(st_coordinates), 20
as.tbl_cube.stars(dplyr), 5
as_tibble.stars(st_coordinates), 20
c.stars, 3, 12
classIntervals, 9
coerce.stars, Raster-method (as), 3
coord_equal, 6
cut, 4
cut.array(cut_stars), 4
cut.matrix(cut_stars), 4
cut.stars(cut_stars), 4
cut_stars, 4
dplyr, 5

dplyr, 5

filter, 5
filter.stars(dplyr), 5
findInterval, 3
gdal_utils, 26
gdal_write, 28
geom_raster, 6
gem_sf, 6
gem_stars, 6
geom_tile, 6

image.stars(plot.stars), 8

make.intervals, 7
makeCluster, 16
Math.stars(ops_stars), 7
Math.stars_proxy(ops_stars), 7

mutate.stars(dplyr), 5

normalizePath, 12

Ops.stars(ops_stars), 7
Ops.stars_proxy(ops_stars), 7
ops.stars, 7
plot.stars, 8
plot.stars_proxy(plot.stars), 8
projectRaster, 26
pull, 5
pull.stars(dplyr), 5
rasterImage, 9
read_ncdf, 10
read_stars, 3, 11
redimension, 13
rgb, 9

select.stars(dplyr), 5
slice.stars(dplyr), 5
st_apply, 15
st_as_sf, 3, 10, 16, 19
st_as_sfc.stars(st_as_sf), 16
st_as_stars, 12, 17, 24
st_contour, 19
st_coordinates, 20
st_crop, 14, 20
st_dimensions, 22
st_drivers, 28
st_get_dimension_values
(st_dimensions), 22
st_make_valid, 17
st_rasterize, 24
st_redimension(redimension), 13
st_set_dimensions(st_dimensions), 22
st_transform, 25, 25
st_transform_proj.stars(st_transform),
25
st_warp, 25, 26
st_xy2sfc, 27
stars_subset, 14
var.get.nc, 11
write_stars, 28