

Package ‘spex’

July 23, 2025

Type Package

Title Spatial Extent Tools

Version 0.7.1

Description Functions to produce a fully fledged 'geo-spatial' object extent as a 'SpatialPolygonsDataFrame'. Also included are functions to generate polygons from raster data using 'quadmesh' techniques, a round number buffered extent, and general spatial-extent and 'raster-like' extent helpers missing from the originating packages. Some latitude-based tools for polar maps are included.

License GPL-3

Encoding UTF-8

LazyData true

Depends R (>= 3.2.5)

Imports methods, quadmesh, raster, reproj, sp, stats, crsmeta

RoxygenNote 7.1.0.9000

URL <https://mdsumner.github.io/spex/>

BugReports <https://github.com/mdsumner/spex/issues>

Suggests covr, testthat (>= 2.1.0)

ByteCompile TRUE

NeedsCompilation no

Author Michael D. Sumner [aut, cre]

Maintainer Michael D. Sumner <mdsumner@gmail.com>

Repository CRAN

Date/Publication 2020-06-20 09:50:02 UTC

Contents

spex-package	2
buffer_extent	2
extent	3

ice	3
latitudecircle	4
latmask	5
lux	6
polygonize	6
psf	7
spex	8
xlim	9

Index	11
--------------	-----------

spex-package	<i>spex</i>
--------------	-------------

Description

Tools for spatial extents that are agnostic regarding format (i.e. sp, sf, or raster). These functions fill some of the gaps within and between these packages for dealing with object extents in flexible ways. Generally, spex considers extents of raster cells, and extents of objects as first-class objects (with projection metadata). and provides helpers for latitudinal boundaries within projected data.

Spatial Extent

buffer_extent	Buffer an extent to a given whole number.
latitudecircle	Create a latitude circle in a chosen projection.
latmask	Mask a raster based on a minimum (or maximum) latitude.
polygonize	Convert raster cells to polygons.
qm_rasterToPolygons	The sf version of polygonize.
qm_rasterToPolygons_sp	The sp version of polygonize.
spex	A function to produce a fully fledged Spatial object extent.
xlim, ylim	Helper functions for extents.

buffer_extent	<i>Whole grain buffers</i>
---------------	----------------------------

Description

Ensure a raster extent aligns to a clean divisor.

Usage

buffer_extent(e1, e2)

Arguments

- e1 input `extent`
- e2 grain size

Details

This function is used to generate extents that have tidy boundaries, i.e. extents that align to a clean whole number like "10000".
(We can't use the S4 group generic because raster has set that specifically for use with '+' and '-'.)

Examples

```
library(raster)
buffer_extent(extent(0.1, 2.2, 0, 3), 2)

p <- par(xpd = NA)
plot(lux)
plot(extent(lux), lty = 2, add = TRUE, col = "grey")
plot(buffer_extent(lux, 0.1), add = TRUE)
abline(v = c(5.7, 6.6), h = c(49.4, 50.2))
title("boundaries on clean alignment to 0.1")
par(p)
```

extent	<i>Extent of simple features</i>
--------	----------------------------------

Description

This is the simplest of the missing "raster support" for the sf package, here using the xmin, xmax, ymin, ymax convention used by raster rather than the transpose version favoured in sp and sf.

Usage

```
extent_sf(x, ...)
```

Arguments

- x object with an extent
- ... unused

ice	<i>A raster data set with southern ocean sea ice concentration</i>
-----	--

Description

When first created this data set was from 2018-04-28.

latitudecircle	<i>Latitude circle</i>
----------------	------------------------

Description

Create a circular polygon using a latitude value in a map projection. The longitude range can be modified from global to give a portion of a circle.

Usage

```
latitudecircle(
  latitude = 0,
  crs = "+proj=stere +lon_0=0 +lat_0=-90 +lat_ts=-71 +ellps=WGS84",
  lonlim = c(-180, 180),
  nverts = 1800
)
```

Arguments

latitude	latitude value for the boundary, defaults to 0
crs	map projection to use, defaults to southern Polar Stereographic true scale at -71S
lonlim	the range of longitude to use, defaults to entire globe
nverts	total number of vertices to use, see Details

Details

The argument `nverts` controls the total number of vertices of the circle within a linearly within the `lonlim` range of longitudes at `latitude`

This is for use on classic polar projections centred on the north or the south pole, particularly Polar Stereographic and Lambert Azimuthal Equal Area but will also work with some caveats on other families and situations. We have not explored this more general use. Feel free to contact the maintainer if you have interest in less typical usage or find problems.

Value

SpatialPolygonsDataFrame

Examples

```
latitudecircle(seq(0, -65, by = -5))
library(raster)
plot(ice)
circ <- latitudecircle(-71, crs = projection(ice))
plot(circ, add = TRUE)
```

latmask	<i>Latitude mask for polar raster</i>
---------	---------------------------------------

Description

Mask out values based on latitude for a raster. This works by finding all cells at latitudes less than latitude and setting them to missing. If southern = FALSE the inequality is reversed, and all cells at latitudes greater than latitude are masked out.

Usage

```
latmask(x, latitude = 0, southern = TRUE, trim = FALSE, ...)
```

Arguments

x	a raster layer
latitude	maximum latitude (effectively a minimum latitude if southern = FALSE)
southern	flag for whether south-polar context is used, default is TRUE
trim	if TRUE runs <code>raster::trim</code> on the result, to remove NA margin
...	ignored currently

Details

The trim option allows for the result to be reduced to the common bounding box within which any row or column has a non-missing value.

Value

RasterLayer

See Also

[raster::trim](#), [latitudecircle](#)

Examples

```
library(raster)
plot(latmask(ice, -60))
plot(latmask(ice, -60, trim = TRUE))
ice[!ice > 0] <- NA
plot(ice)
plot(latmask(ice, -55, trim = TRUE))
```

lux

*The 'lux' Spatial Polygons from the 'raster' package.***Description**

The 'lux' Spatial Polygons from the 'raster' package.

Format

`SpatialPolygonsDataFrame`-class with columns:

- ID_1
- NAME_1
- ID_2
- NAME_2
- AREA

Examples

```
library(sp)
plot(lux)
```

polygonize

*Create a polygon layer from a raster.***Description**

This method uses the quadmesh to generate the coordinates, and creates a simple features layer. It's faster by turning off the checking done in the simple features package, but it's also faster than raster because it uses a dense mesh to generate the coordinates.

Usage

```
## S3 method for class 'RasterLayer'
polygonize(x, na.rm = TRUE, ...)

polygonize(x, ...)

qm_rasterToPolygons(x, na.rm = TRUE, ...)

## S3 method for class 'RasterStack'
polygonize(x, na.rm = TRUE, ...)

## S3 method for class 'RasterBrick'
polygonize(x, na.rm = TRUE, ...)

qm_rasterToPolygons_sp(x, na.rm = TRUE, ...)
```

Arguments

x	raster, brick or stack
na.rm	defaults to TRUE and will polygonize all the cells that are non-NA in any layer, set to FALSE to not remove any cells
...	arguments passed to methods, currently unused

Details

If na.rm is TRUE (the default) only cells that are not-NA across all layers are created. An exception to this is the empty raster raster::hasValues(x) is FALSE and all the cells will be turned into polygons - since this is what the whole scene is really for, easily creating polygons from a grid.

Value

simple features POLYGON layer, or SpatialPolygonsDataFrame

Warning

Please don't try this on large rasters (> ~1e5 cells), use quadmesh itself for efficient vector based use of a raster's coordinates. It will work reasonably on largish grids, but you won't want to try plotting them or perform operations on them, simple features is incredibly wasteful for objects like this.

Examples

```
#library(raadtools)
library(raster)
r <- raster(volcano)
r[sample(ncell(r), 3000)] <- NA
b <- brick(r, r*1.5)
psf <- qm_rasterToPolygons(r, na.rm = TRUE)
#psp <- qm_rasterToPolygons_sp(r)
#pspr <- rasterToPolygons(r)
#library(rbenchmark)
#benchmark(qm_rasterToPolygons(r), qm_rasterToPolygons_sp(r), rasterToPolygons(r), replications = 2)
#           test replications elapsed relative user.self sys.self user.child sys.child
# 1  qm_rasterToPolygons(r)           2  0.476    1.000      0.476  0.000         0         0
# 2 qm_rasterToPolygons_sp(r)           2  4.012    8.429      3.964  0.048         0         0
# 3   rasterToPolygons(r)              2  2.274    4.777      2.268  0.008         0         0
```

psf	<i>A polygon data set with sf class.</i>
-----	--

Description

A polygon data set with sf class.

spex

*Polygon extent***Description**

Create Spatial Polygons with projection metadata from a 'Spatial Extent'.

Usage

```
spex(x, crs, byid = FALSE, .id, ..., clipboard = FALSE)

## Default S3 method:
spex(x, crs = NULL, byid = FALSE, .id, ..., clipboard = FALSE)

## S3 method for class 'sf'
spex(x, crs, byid = FALSE, .id, ..., clipboard = FALSE)

## S3 method for class 'sfc'
spex(x, crs, byid = FALSE, .id, ..., clipboard = FALSE)
```

Arguments

x	any object with a Extent-class
crs	a projection string
byid	return a separate object for every input sub-object (not yet implemented)
.id	optional name for output attribute name
...	arguments for methods
clipboard	WIP this special-case allows x to be the result of the leafem clipboard copy process

Details

Called with no arguments will return the extent of the current 'par("usr")' setting.

Called with a matrix, list, or data frame it will create an extent from a two columned thing.

Called with clipboard = TRUE and x will be treated as the JSON-ic output of the clipboard copy from leafem (WIP). If x is missing, it will be attempted to be read from the clipboard. Clipboard read cannot work on RStudio Server, so we allow the text value to be passed in. I.e. spex(clipboard = TRUE) will read from the clipboard, spex(tx, clipboard = TRUE) will read from tx with value like '{"_southWest":{"lat":-1.307259612275665,"lng":23.411865234375},"_north...}'.

This function is to replace a common pattern in spatial packages which is

- create an [Extent-class](#), a bounding box in xmin,xmax,ymin,ymax but without projection metadata
- coerce the Extent to [SpatialPolygons](#)

- restore the 'CRS', the "coordinate reference system", i.e. projection metadata
- elevate the object to be a [SpatialPolygonsDataFrame-class](#).

In short, this pattern exists because there is no projection metadata stored with either sp's [bbox](#) or raster's [Extent-class](#).

Value

'SpatialPolygonsDataFrame'

Warning

Please note that an extent converted to polygons consists of only four unique coordinates, and so this is not necessarily suited for projection transformations.

See Also

This pattern is displayed in the example code for [cover](#).

Examples

```
library(raster)
data(lux)
exlux <- spex(lux)

plot(lux)
plot(exlux, add = TRUE)

## put an extent and a CRS together
spex(extent(0, 1, 0, 1), crs = "+proj=laea +ellps=WGS84")
```

xlim	<i>Axis ranges from extent</i>
------	--------------------------------

Description

Functions xlim and ylim return the two-value counterparts of an extent.

Usage

```
xlim(x, ...)

## Default S3 method:
xlim(x, ...)

ylim(x, ...)

## Default S3 method:
ylim(x, ...)
```

Arguments

<code>x</code>	any object with an extent understood by <code>spex</code>
<code>...</code>	reserved for future methods

Details

Any projection metadata is dropped since this is a one-dimensional entity.

Index

bbox, [9](#)
buffer_extent, [2](#), [2](#)

cover, [9](#)

Extent (extent), [3](#)
extent, [3](#), [3](#)
extent_sf (extent), [3](#)

ice, [3](#)

latitudecircle, [2](#), [4](#), [5](#)
latmask, [2](#), [5](#)
lux, [6](#)

polygonize, [2](#), [6](#)
psf, [7](#)

qm_rasterToPolygons, [2](#)
qm_rasterToPolygons (polygonize), [6](#)
qm_rasterToPolygons_sp, [2](#)
qm_rasterToPolygons_sp (polygonize), [6](#)

raster::trim, [5](#)

SpatialPolygons, [8](#)
spex, [2](#), [8](#)
spex-package, [2](#)

xlim, [2](#), [9](#)

ylim, [2](#)
ylim (xlim), [9](#)