Package 'lacunr'

May 8, 2026

Type Package

Title Fast 3D Lacunarity for Voxel Data

Version 1.0.2

Date 2025-08-27

URL https://github.com/ElliottSmeds/lacunr

BugReports https://github.com/ElliottSmeds/lacunr/issues

Description Calculates 3D lacunarity from voxel data. It is designed for use with point clouds generated from Light Detection And Ranging (LiDAR) scans in order to measure the spatial heterogeneity of 3-dimensional structures such as forest stands. It provides fast 'C++' functions to efficiently bin point cloud data into voxels and calculate lacunarity using different variants of the gliding-box algorithm originated by Allain & Cloitre (1991) <doi:10.1103/PhysRevA.44.3552>.

License GPL (>= 3)

Encoding UTF-8

Imports Rcpp (>= 1.0.10), data.table, abind, ggplot2, rlang

Suggests knitr, lidR, rmarkdown, testthat (>= 3.0.0)

LinkingTo Rcpp, RcppArmadillo, RcppThread

RoxygenNote 7.3.2

Config/testthat/edition 3

Depends R (>= 2.10)

LazyData true

LazyDataCompression bzip2

VignetteBuilder knitr

NeedsCompilation yes

Author Elliott Smeds [aut, cre, cph] (ORCID: <https://orcid.org/0000-0003-1054-7491>), J. Antonio Guzmán Q. [cph] (Author of original version of voxelize() function)

Maintainer Elliott Smeds <elliott.alfred93@gmail.com>

Repository CRAN

Date/Publication 2025-08-27 23:40:02 UTC

Contents

`bounding_box` *Create voxel array*

Description

`bounding_box()` takes a table of voxel data and converts it into a 3-dimensional array, with the original voxels arranged in their correct spatial positions inside of a 3-D "box" of empty voxels. This array can be input directly into `lacunarity()` to generate lacunarity curves.

Usage

```
bounding_box(x, threshold = 0, edge_length = NULL)
```

Arguments

`x`	A '`lac_voxels`' object created by `voxelize()` (preferred), or a '`lasmetrics3d`' object created by `lidR::voxel_metrics()`. Users can alternatively supply a `data.table::data.table()` containing X, Y, Z, and N columns, in which case the argument `edge_length` is required.
`threshold`	An integer specifying the minimum number of points to use when determining if a voxel is occupied. The default is `0`. `bounding_box()` retains only those voxels where `x$N` is greater than `threshold`.
`edge_length`	a numeric vector of length 3, specifying the X, Y, and Z dimensions of each voxel. This argument should only be necessary when supplying voxel data generated by a function other than `voxelize()` or `lidR::voxel_metrics()`.

Details

bounding_box() relies on the spatial coordinates of the input data to determine the dimensions of the resulting array. Noisy point cloud data will often produce "outlier" voxels surrounding points that are far removed from the bulk of the point cloud. These can drastically alter the output, creating an array in which the occupied voxels are surrounded by a large region of empty space. It is highly recommended that users supply a cutoff value for threshold to ideally remove these outliers. More elaborate filtering tools for trimming the point data before voxelization are available in the lidR package.

Value

A 3-dimensional integer array containing values of 0 or 1, representing the occupancy of a given voxel. Occupied voxels (1) are arranged according to their relative positions in 3-dimensional space, and fully encapsulated within a rectangular volume of unoccupied voxels (0). The XYZ positions of the voxels are retained in the array dimnames.

Examples

```
# Basic usage ------------------------------------------------------------
# simulate a diagonal line of points with XYZ coordinates
pc <- data.frame(X = as.numeric(0:24),
                 Y = as.numeric(0:24),
                 Z = as.numeric(0:24))
# convert point data to cubic voxels of length 5
vox <- voxelize(pc, edge_length = c(5,5,5))
# convert to voxel array
box <- bounding_box(vox)

# Using lidR::voxel_metrics ----------------------------------------------
if (require("lidR")){
# reformat point data into rudimentary LAS object
las <- suppressMessages(lidR::LAS(pc))
# convert to voxels of length 5
vox <- lidR::voxel_metrics(las, ~list(N = length(Z)), res = 5)
# convert to voxel array
box <- bounding_box(vox)
}
```

---

glassfire *California oak forest stand before and after 2020 Glass Fire*

Description

This dataset contains point cloud data from two terrestrial LiDAR scans of a Northern California oak forest shortly before and after the Glass Fire, which burned some 27000 hectares of land in Sonoma and Napa counties between September 27 and October 20, 2020. The scans encompass an identical 24m by 24m rectangular plot at a study site within the Saddle Mountain Open Space Preserve in Sonoma County.

Usage

```
glassfire
```

Format

A data table with 1,000,000 rows and 4 columns: `X`, `Y`, `Z`, and `Year`

`X`,`Y`,`Z` The XYZ spatial positions of each point, in meters. X and Y denote the East-West and North-South horizontal positions, respectively, while Z denotes the vertical position

`Year` The year each LiDAR scan was taken, either 2020, immediately before the Glass Fire, or 2021, a few months after

Details

The original terrain topography has been removed using digital elevation model (DEM)-based height normalization, and ground points removed by clipping all points below 0.25m. The raw point cloud data were normalized via voxelization at a resolution of 0.125m, and the results further down-sampled to make the dataset more compact. The X, Y, and Z coordinates were generated from the original Easting, Northing, and elevation by subtracting their minimum values.

`glassfire` is technically encoded as a `lasmetrics3d` object from the `lidR` package. This class inherits from `data.table()`, but has the added benefit that it can be rendered as a 3D rgl plot using `lidR::plot.lasmetrics3d()`.

`lacunarity` *Calculate gliding-box lacunarity*

Description

Generates $\Lambda(r)$ lacunarity curves for a specified set of box sizes, using one of two versions of the gliding-box algorithm

Usage

```
lacunarity(x, box_sizes = "twos", periodic = FALSE)
```

Arguments

`x`	A 3-dimensional `array` of integer values
`box_sizes`	Which box sizes to use for calculating lacunarity: • `"twos"` (the default) returns box sizes for all powers of two less than or equal to the smallest dimension of `x`. • `"all"` calculates every possible box size up to the smallest dimension of `x`. • Alternatively, users may supply their own `vector` of custom box sizes. This vector must be of type `"numeric"` and can only contain positive values. Values which exceed the dimensions of `x` are ignored.

`periodic` A Boolean. Determines which boundary algorithm to use, the classic fixed boundary by Allain and Cloitre (default) or the periodic boundary algorithm introduced by Feagin et al. 2007. The latter is slightly slower but is more robust to edge effects.

Details

The raw $\Lambda(r)$ values depend on the proportion of occupied voxels within the data space. As a result, it is difficult to compare two spatial patterns with different occupancy proportions because the curves will begin at different y-intercepts. This is rectified by normalizing the curve, typically by log-transforming it and dividing by the lacunarity value at the smallest box size (i.e. $\log \Lambda(r) / \log \Lambda(1)$). `lacunarity()` outputs both normalized and non-normalized $\Lambda(r)$ curves for convenience.

The function also computes $\mathrm{H}(r)$, a transformed lacunarity curve introduced by Feagin 2003. $\mathrm{H}(r)$ rescales normalized $\Lambda(r)$ in terms of the Hurst exponent, where values greater than 0.5 indicate heterogeneity and values less than 0.5 indicate homogeneity. Where $\Lambda(r)$ describes a pattern's deviation from translational invariance, $\mathrm{H}(r)$ describes its deviation from standard Brownian motion.

Value

A `data.frame` containing box sizes and their corresponding $\Lambda(r)$, normalized $\Lambda(r)$, and $\mathrm{H}(r)$ values. Lacunarity is always computed for box size 1, even if the user supplies a custom `box_sizes` vector that omits it, as this value is required to calculate normalized lacunarity.

References

Allain, C., & Cloitre, M. (1991). Characterizing the lacunarity of random and deterministic fractal sets. *Physical Review A*, **44(6)**, 3552–3558. doi:10.1103/PhysRevA.44.3552.

Feagin, R. A. (2003). Relationship of second-order lacunarity, Hurst exponent, Brownian motion, and pattern organization. *Physica A: Statistical Mechanics and its Applications*, **328(3-4)**, 315-321. doi:10.1016/S03784371(03)005247.

Feagin, R. A., Wu, X. B., & Feagin, T. (2007). Edge effects in lacunarity analysis. *Ecological Modelling*, **201(3–4)**, 262–268. doi:10.1016/j.ecolmodel.2006.09.019.

Examples

```
# generate array
a <- array(data = rep(c(1,0), 125), dim = c(5,5,5))
# calculate lacunarity with default options
lacunarity(a)
# supply custom vector of box sizes
lacunarity(a, box_sizes = c(1,3,5))
# calculate lacunarity at all box sizes using the periodic boundary algorithm
lacunarity(a, box_sizes = "all", periodic = TRUE)
```

lac_plot *Plot lacunarity curve(s)*

Description

Plot lacunarity curve(s)

Usage

```
lac_plot(..., log = TRUE, group_names = NULL)

lacnorm_plot(..., log = TRUE, group_names = NULL)

hr_plot(..., group_names = NULL)
```

Arguments

...	One or more data.frames containing lacunarity curve data. Must contain columns named $box_size, $lacunarity, $lac_norm, and $H_r.
log	A Boolean. TRUE (default) displays the axes on a logarithmic scale, FALSE displays them on a linear scale. For lacnorm_plot() this only controls the x axis, as normalized lacunarity is by definition on a log scale.
group_names	A character vector containing labels for any data.frames passed to .... These labels will appear on the plot legend. If group_names is left empty, the legend uses the names of the data.frames as supplied to ...

Value

A ggplot object displaying the lacunarity or H(r) curve(s). If multiple curves are supplied, their ordering in the plot legend will reflect the order they were listed in the function call.

Examples

```
# generate array
a <- array(data = rep(c(1,0), 125), dim = c(5,5,5))
# calculate lacunarity at all box sizes
lac_curve <- lacunarity(a, box_sizes = "all")
# plot raw lacunarity curve
lac_plot(lac_curve)
```

pad_array *Add padding to 3D array*

Description

`pad_array()` adds additional rows, columns, or slices to a 3-dimensional array, increasing the array's dimensions by the desired amount and filling the new space with a uniform value. It is intended for adding empty or occupied space to the edges of a 3D spatial map.

Usage

```
pad_array(a, x = 0, y = 0, z = 0, fill = 0)
```

Arguments

`a`	A 3-dimensional `array` of numeric values
`x`	A positive or negative integer, denoting the number of rows to add to the array. The sign dictates which side of the array to pad. Default is zero.
`y`	A positive or negative integer, denoting the number of columns to add to the array. The sign dictates which side of the array to pad. Default is zero.
`z`	A positive or negative integer, denoting the number of "slices" to add to the array. The sign dictates which side of the array to pad. Default is zero.
`fill`	The desired value to fill the array padding with. Default is zero.

Details

`pad_array()` uses the signs of `x`, `y`, and `z` to determine where to add padding. Negative values are prepended before the array's lower indices (what one might call the "top", "front", or "beginning" of the array), while positive values are appended after the upper indices (the "bottom", "back", or "end" of the array).

Value

A 3-dimensional `array` with the desired padding added. The padded portions are labelled using their `dimnames`. If no padding has been specified, the function returns the original array.

Examples

```
# generate array
a <- array(data = rep(c(1,0), 125), dim = c(5,5,5))
# add two rows of zeroes to top of array
pad <- pad_array(a, x = -2)
# add one row of zeroes to bottom of array, and two columns to beginning
pad <- pad_array(a, x = 1, y = -2)
```

`voxelize` *Voxelize point cloud*

Description

Bins point cloud data into 3D pixels, otherwise known as 'voxels'.

Usage

```
voxelize(x, edge_length, threads = 1L)
```

Arguments

`x`	A `data.frame` or `data.table::data.table()` with columns containing the X, Y, and Z coordinates of every point. Any additional columns are ignored.
`edge_length`	A numeric `vector` of length `3`, containing values for the desired X, Y, and Z dimensions of each voxel.
`threads`	The number of threads to use for computing the voxel data. Default is 1.

Value

A data object of class '`lac_voxels`', which inherits from `data.table::data.table()`. The output contains 4 columns: X, Y, Z, and N. The first three columns encode the spatial coordinates of each voxel while the fourth denotes the total number of points they contain.

Examples

```
# simulate a diagonal line of points with XYZ coordinates
pc <- data.frame(X = 0:99, Y = 0:99, Z = 0:99)
# convert point data to cubic voxels of length 5
voxelize(pc, edge_length = c(5,5,5))
```

Index