Package: hexsmoothR (via r-universe)

May 17, 2026

Type Package

Title Hexagonal Grid Smoothing for Satellite Data

Version 0.2.0

Description Creates hexagonal grids and applies spatial smoothing to satellite raster data. Provides tools for extracting environmental variables from TIF files and applying Gaussian-weighted spatial smoothing using 'Rcpp' for performance. The workflow has two steps: (1) extract raster data into hexagonal grids, and (2) apply N-order neighbour smoothing with customisable weights.

License BSD_3_clause + file LICENSE

URL https://maxmlang.github.io/hexsmoothR/, https://github.com/MaxMLang/hexsmoothR

BugReports https://github.com/MaxMLang/hexsmoothR/issues

Encoding UTF-8

Depends R (>= 4.0.0)

LinkingTo Rcpp

Imports Rcpp (>= 1.0.0), sf (>= 1.0.0), terra (>= 1.6.0), exactextractr (>= 0.8.0)

Suggests testthat (>= 3.0.0), knitr, rmarkdown, Matrix, ggplot2, gridExtra, withr

VignetteBuilder knitr

RoxygenNote 7.3.2

SystemRequirements GDAL (>= 3.0), GEOS (>= 3.8), PROJ (>= 6.0), C++17

Config/testthat/edition 3

Config/pak/sysreqs libabsl-dev cmake libgdal-dev gdal-bin libgeos-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev

Repository https://maxmlang.r-universe.dev

Date/Publication 2026-04-17 01:19:15 UTC

RemoteUrl https://github.com/MaxMLang/hexsmoothR

RemoteRef HEAD

RemoteSha 989672087916e0d6204319ab165a9bf4104712f2

Contents

`compute_topology` *Compute spatial topology and weights for hexagonal smoothing*

Description

Finds spatial neighbours and computes Gaussian-based weights for smoothing. Neighbours are computed up to ‘neighbor_orders‘ orders (1st-order = touching, 2nd-order = neighbours of neighbours, etc.). Weights decay with order using a Gaussian kernel.

Usage

```
compute_topology(
  grid,
  projection_crs = NULL,
  neighbor_orders = 2,
  sigma = NULL,
  center_weight = 1,
  neighbor_weights_param = NULL,
  adaptive_sigma_factor = 0.5,
  sample_size = 100
)
```

Arguments

`grid`	sf object with polygonal geometries (or a list of sf objects).
`projection_crs`	CRS used for distance calculations. Default ‘NULL‘ selects an appropriate UTM zone via [‘get_utm_crs()‘].
`neighbor_orders`	Number of neighbour orders (positive integer).

`sigma` Gaussian bandwidth. ‘NULL‘ (default) auto-computes from ‘avg_distance‘ and ‘adaptive_sigma_factor‘.

`center_weight` Weight for the centre cell.

`neighbor_weights_param` Optional list of length ‘neighbor_orders‘ with per-order weights (overrides Gaussian computation).

`adaptive_sigma_factor` Scaling factor for auto-sigma.

`sample_size` Cells to sample for the average-distance estimate.

Value

List (or named list of lists if ‘grid‘ was a list) containing ‘neighbors‘, ‘weights‘, ‘avg_distance‘, ‘sigma‘, ‘grid_ids‘, ‘grid_indices‘, ‘neighbor_orders‘.

Weight computation

- ‘sigma = avg_distance * adaptive_sigma_factor‘ (auto-bandwidth) when ‘sigma‘ is ‘NULL‘. - For order N: weight ~ ‘exp(-N^2 / (2 * sigma^2))‘. - All weights (including ‘center_weight‘) are normalised to sum to 1.

`create_grid` *Create hexagonal or square grids for spatial analysis*

Description

Creates regular hexagonal or square grids over a study area. The function automatically handles coordinate-system transformations and ensures proper grid alignment.

Usage

```
create_grid(
  study_area,
  cell_size,
  type = c("hexagonal", "square"),
  projection_crs = NULL,
  id_column = NULL,
  return_crs = 4326,
  check_size = TRUE,
  max_cells = 1e+06
)
```

Arguments

study_area	sf object containing polygonal geometries defining the study area. May be in any CRS.
cell_size	Grid cell size (see Details for units).
type	‘"hexagonal"‘ (default) or ‘"square"‘.
projection_crs	CRS used for grid construction. Default ‘NULL‘ selects an appropriate UTM zone via [‘get_utm_crs()‘]. Pass an EPSG code or CRS string to override.
id_column	Optional column name in ‘study_area‘ for creating separate grids per unique value. If supplied, returns a named list of grids.
return_crs	CRS for the output grid (default WGS84). The grid is transformed to this CRS after creation.
check_size	Whether to warn when the grid is very large (default TRUE).
max_cells	Maximum number of cells allowed before stopping (default 1,000,000). Set to ‘NULL‘ to disable.

Details

Cell size units depend on the projection CRS: - Projected CRS (UTM, etc.): ‘cell_size‘ is in **metres**. - Geographic CRS (WGS84): ‘cell_size‘ is in **degrees**.

Use a projected CRS for real-world analysis. If ‘projection_crs‘ is ‘NULL‘ (the default), an appropriate UTM zone is chosen automatically using [‘get_utm_crs()‘].

Hexagons are pointy-topped; ‘cell_size‘ is the flat-to-flat distance (width between opposite edges).

Value

sf object containing the grid (or named list of sf objects when ‘id_column‘ is given). Each grid cell carries ‘grid_id‘ and ‘grid_index‘.

Examples

```
## Not run:
library(sf)
study_area <- st_sf(geometry = st_sfc(
  st_polygon(list(matrix(c(-5, 35, 5, 35, 5, 45, -5, 45, -5, 35),
                         ncol = 2, byrow = TRUE))),
  crs = 4326
))
hex_grid <- create_grid(study_area, cell_size = 1000, type = "hexagonal")

## End(Not run)
```

`extract_raster_data` *Extract raster data into a hexagonal grid*

Description

Primary function for extracting raster values into hexagonal grid cells. CRS transformations are handled automatically: each raster is reprojected (or the grid is reprojected to the raster's CRS) so that the underlying call to [‘exactextractr::exact_extract()‘] always sees matching CRSs.

Usage

```
extract_raster_data(
  raster_files,
  study_area = NULL,
  cell_size = NULL,
  hex_grid = NULL,
  sample_fraction = 1,
  random_seed = 42,
  fun = "mean"
)
```

Arguments

`raster_files`	Named character vector of file paths OR named list of ‘terra::SpatRaster‘ objects.
`study_area`	Optional sf polygon used for cropping each raster and to define the grid CRS.
`cell_size`	Hex cell size, in the units of the grid CRS. Required when ‘hex_grid‘ is not supplied.
`hex_grid`	Optional sf hexagonal grid to use instead of creating one.
`sample_fraction`	Fraction of grid cells to keep (default 1).
`random_seed`	Seed for reproducible sampling.
`fun`	Aggregation function passed to ‘exactextractr::exact_extract()‘ (default ‘"mean"‘).

Value

List with components

‘data‘ Data frame with ‘cell_id‘, ‘x‘, ‘y‘ and one column per raster.

‘hex_grid‘ The sf grid that was used (sampled, if applicable).

‘cell_size‘ The cell size used.

‘extent‘ Extent of the first raster (after cropping).

‘variables‘ Names of the rasters.

‘n_cells‘ Number of cells in ‘data‘.

Input flexibility

'raster_files' may be either a named character vector of file paths or a named list of 'terra::SpatRaster' objects. All inputs may be in different CRSs from one another and from the grid - the function handles cropping and transformation per raster.

CRS handling

- If 'study_area' is supplied, the grid is created in the study area's CRS. - Otherwise the grid is created in the first raster's CRS. - For each raster, the grid is transformed to the raster's CRS before extraction (so cell-size units are honoured exactly once, in the grid CRS).

---

find_hex_cell_size_for_target_cells

Find a hexagonal cell size that yields approximately a target number of cells

---

Description

Uses a binary search over 'cell_size_min' / 'cell_size_max' to find the flat-to-flat distance that produces approximately 'target_cells' hexagons.

Usage

```
find_hex_cell_size_for_target_cells(
  study_area,
  target_cells,
  cell_size_min = NULL,
  cell_size_max = NULL,
  tol = 0.05,
  max_iter = 20,
  projection_crs = NULL
)
```

Arguments

study_area	sf object (polygons).
target_cells	Desired number of hexagons (positive integer).
cell_size_min	Minimum cell size to try. Default 'NULL', in which case the search range is derived from the bounding box of 'study_area' so it works for either projected (metres) or geographic (degrees) input.
cell_size_max	Maximum cell size to try. See 'cell_size_min'.
tol	Convergence tolerance (fraction of 'target_cells').
max_iter	Maximum binary-search iterations.
projection_crs	Optional CRS for grid construction (passed to ['create_grid()']).

Value

Cell size (flat-to-flat distance) closest to ‘target_cells‘.

`get_utm_crs` *Get an appropriate UTM CRS for a study area*

Description

Determines the appropriate UTM coordinate reference system (CRS) for a given study area based on the longitude/latitude of its centroid.

Usage

```
get_utm_crs(study_area)
```

Arguments

`study_area`	sf object representing the study area. May be in any CRS; it will be transformed to WGS84 (EPSG:4326) internally for the centroid calculation.

Value

Character string with the UTM CRS, e.g. ‘"EPSG:32630"‘ for UTM 30N.

Examples

```
## Not run:
library(sf)
study_area <- st_sf(geometry = st_sfc(
  st_polygon(list(matrix(c(-5, 35, 5, 35, 5, 45, -5, 45, -5, 35),
                         ncol = 2, byrow = TRUE))),
  crs = 4326
))
get_utm_crs(study_area)  # "EPSG:32630"

## End(Not run)
```

`hex_flat_to_edge` *Convert between hexagon measurements*

Description

Helper functions to convert between different hexagon measurements. For a regular hexagon, the circumradius equals the edge length, so the "edge" and "circumradius" helpers are mathematically identical (provided for clarity at the call site).

Usage

```
hex_flat_to_edge(flat_to_flat)

hex_flat_to_circumradius(flat_to_flat)

hex_edge_to_flat(edge_length)

hex_circumradius_to_flat(circumradius)
```

Arguments

`flat_to_flat`	Flat-to-flat distance (between opposite edges).
`edge_length`	Edge length of the hexagon.
`circumradius`	Circumradius (centre to vertex).

Details

- ‘hex_flat_to_edge()‘: flat-to-flat distance to edge length - ‘hex_flat_to_circumradius()‘: flat-to-flat distance to circumradius - ‘hex_edge_to_flat()‘: edge length to flat-to-flat distance - ‘hex_circumradius_to_flat()‘: circumradius to flat-to-flat distance

Value

Numeric value in the same units as the input.

Examples

```
hex_flat_to_edge(1000)          # ~577.35
hex_flat_to_circumradius(1000)  # ~577.35
hex_edge_to_flat(577.35)        # ~1000
hex_circumradius_to_flat(577.35)# ~1000
```

`smooth_variables` *Apply spatial smoothing to variables on a hexagonal grid*

Description

Smooths variables using the topology produced by [‘compute_topology()‘]. Uses the compiled C++ implementation when available; if the C++ call fails for an unexpected reason, falls back to the pure-R implementation and re-raises clear validation errors.

Usage

```
smooth_variables(
  variable_values,
  neighbors,
  weights,
  hex_indices = NULL,
  var_names = NULL
)
```

Arguments

`variable_values`	Named list of numeric vectors (one per variable).
`neighbors`	List of neighbour lists (per order). Either the ‘neighbors‘ element of a topology, or the topology object itself.
`weights`	List with ‘center_weight‘ and ‘neighbor_weights‘. Either the ‘weights‘ element of a topology, or the topology object itself.
`hex_indices`	Integer vector of cell indices to process. Defaults to all cells.
`var_names`	Character vector of variable names. Defaults to ‘names(variable_values)‘.

Value

Named list with one entry per variable. Each entry contains:

- ‘raw‘: the original (unsmoothed) centre-cell values
- ‘weighted_combined‘: weighted average of centre + all neighbours
- ‘neighbors_<N><suffix>‘: mean of neighbours at order N (e.g. ‘neighbors_1st‘, ‘neighbors_2nd‘, ‘neighbors_3rd‘)

Index