Package 'starsExtra'

October 6, 2026

Title Miscellaneous Functions for Working with 'stars' Rasters

Version 0.2.9

Description Miscellaneous functions for working with 'stars' objects, mainly single-band rasters. Currently includes functions for: (1) focal filtering, (2) detrending of Digital Elevation Models, (3) calculating flow length, (4) calculating the Convergence Index, (5) calculating topographic aspect and topographic slope.

Depends R (>= 3.5.0), sf, stars

Imports methods, parallel, mgcv, nngeo, units

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Suggests tinytest, knitr, rmarkdown, raster

VignetteBuilder knitr

URL https://michaeldorman.github.io/starsExtra/, https://github.com/michaeldorman/starsExtra/

BugReports https://github.com/michaeldorman/starsExtra/issues/

Config/roxygen2/version 8.1.0

NeedsCompilation yes

Author Michael Dorman [aut, cre]

Maintainer Michael Dorman <dorman@post.bgu.ac.il>

Repository CRAN

Date/Publication 2026-10-06 20:00:02 UTC

Contents

aspect 2
carmel 4
CI 4
dem 5

detrend . . . 6
dist_to_nearest . . . 7
extract2 . . . 8
flowlength . . . 9
focal2 . . . 10
focal2r . . . 12
footprints . . . 13
golan . . . 14
landsat . . . 15
layer_to_matrix . . . 15
layer_to_vector . . . 16
make_grid . . . 17
matrix_extend . . . 17
matrix_get_neighbors . . . 18
matrix_to_stars . . . 19
matrix_trim . . . 20
mode_value . . . 20
normalize_2d . . . 21
normalize_3d . . . 22
rgb_to_greyscale . . . 22
slope . . . 23
trim2 . . . 24
w_azimuth . . . 25
w_circle . . . 26

Index **27**

`aspect` *Calculate topographic aspect from a DEM*

Description

Calculates topographic aspect given a Digital Elevation Model (DEM) raster. Input and output are rasters of class `stars`, single-band (i.e., only ‘"x"‘ and ‘"y"‘ dimensions), with one attribute.

Usage

```
aspect(x, na_flag = -9999)
```

Arguments

`x`	A raster (class `stars`) with two dimensions: `x` and `y`, i.e., a single-band raster, representing a DEM.
`na_flag`	Value used to mark `NA` values in C code. This should be set to a value which is guaranteed to be absent from the input raster `x` (default is `-9999`).

Value

A `stars` raster with topographic slope, i.e., the azimuth where the terrain is tilted towards, in decimal degrees (0-360) clockwise from north. Aspect of flat terrain, i.e., where all values in the neighborhood are equal, is set to `-1`. Returned raster values are of class `units` (decimal degrees).

Note

Aspect calculation results in `NA` when at least one of the cell neighbors is `NA`, including the outermost rows and columns. Given that the focal window size in aspect calculation is 3*3, this means that the outermost one row and one column are given an aspect value of `NA`.

The raster must be in projected CRS, and units of x/y resolution are assumed to be the same as units of elevation (typically *meters*).

References

The topographic aspect algorithm is based on the *How aspect works* article in the ArcGIS documentation:

https://desktop.arcgis.com/en/arcmap/10.3/tools/spatial-analyst-toolbox/how-aspect-works.htm

Examples

```
# Small example
data(dem)
dem_aspect = aspect(dem)
plot(
  dem, text_values = TRUE, breaks = "equal",
  col = hcl.colors(11, "Spectral"), main = "input (elevation)"
)
plot(
  dem_aspect, text_values = TRUE, breaks = "equal",
  col = hcl.colors(11, "Spectral"), main = "output (aspect)"
)

# Larger example
data(carmel)
carmel_aspect = aspect(carmel)
plot(
  carmel, breaks = "equal",
  col = hcl.colors(11, "Spectral"), main = "input (elevation)"
)
plot(
  carmel_aspect, breaks = "equal",
  col = hcl.colors(11, "Spectral"), main = "output (aspect)"
)
```

carmel *Digital Elevation Model of Mount Carmel*

Description

A `stars` object representing a Digital Elevation Model (DEM) Digital Elevation Model of Mount Carmel, at 90m resolution

Usage

```
carmel
```

Format

A `stars` object with 1 attribute:

elevation Elevation above sea level, in meters

Examples

```
plot(carmel, breaks = "equal", col = terrain.colors(11))
```

CI *Calculate the Convergence Index (CI) from a slope raster*

Description

Calculates the Convergence Index (CI) given a topographic slope raster. Input and output are rasters of class `stars`, single-band (i.e., only ‘"x"‘ and ‘"y"‘ dimensions), with one attribute.

Usage

```
CI(x, k, na.rm = FALSE, na_flag = -9999)
```

Arguments

`x`	A raster (class `stars`) with two dimensions: `x` and `y`, i.e., a single-band raster, representing aspect in decimal degrees clockwise from north, possibly including `-1` to specify flat terrain, such as returned by function `aspect`.
`k`	k Neighborhood size around focal cell. Must be an odd number. For example, `k=3` implies a 3*3 neighborhood.
`na.rm`	Should `NA` values be ignored when calculating CI? Default is `FALSE`, i.e., when at least one aspect value in the neighborhood is `NA` the CI is also set to `NA`.
`na_flag`	Value used to mark `NA` values in C code. This should be set to a value which is guaranteed to be absent from the input raster `x` (default is `-9999`).

Value

A `stars` raster with CI values.

Note

The raster is "padded" with `(k-1)/2` more rows and columns of `NA` values on all sides, so that the neighborhood of the outermost rows and columns is still a complete neighborhood. Those rows and columns are removed from the final result before returning it. Aspect values of `-1`, specifying flat terrain, are assigned with a CI value of `0` regardless of their neighboring values.

References

The Convergence Index algorithm is described in:

Thommeret, N., Bailly, J. S., & Puech, C. (2010). Extraction of thalweg networks from DTMs: application to badlands.

Examples

```
# Small example
data(dem)
dem_asp = aspect(dem)
dem_ci = CI(dem_asp, k = 3)
r = c(dem, round(dem_ci, 1), along = 3)
r = st_set_dimensions(r, 3, values = c("input (aspect)", "output (CI, k=3)"))
plot(r, text_values = TRUE, breaks = "equal", col = terrain.colors(10), mfrow = c(1, 2))

# Larger example
data(golan)
golan_asp = aspect(golan)
golan_ci = CI(golan_asp, k = 25)
plot(golan_asp, breaks = "equal", col = hcl.colors(11, "Spectral"), main = "input (aspect)")
plot(golan_ci, breaks = "equal", col = hcl.colors(11, "Spectral"), main = "output (CI, k=25)")
```

---

dem *Small Digital Elevation Model*

Description

A `stars` object representing a small 13*11 Digital Elevation Model (DEM), at 90m resolution

Usage

```
dem
```

Format

A `stars` object with 1 attribute:

elevation Elevation above sea level, in meters

Examples

```
plot(dem, text_values = TRUE, breaks = "equal", col = terrain.colors(11))
```

`detrend` *Detrend a Digital Elevation Model*

Description

Detrends a Digital Elevation Model (DEM) raster, by subtracting a trend surface. The trend is computed using `mgcv::gam` or `mgcv::bam` (when `parallel>1`) with formula `z ~ s(x, y)`.

Usage

```
detrend(x, parallel = 1)
```

Arguments

`x`	A two-dimensional `stars` object representing the DEM
`parallel`	Number of parallel processes. With `parallel=1` uses ordinary, non-parallel processing.

Value

A two-dimensional `stars` object, with two attributes:

- `resid` - the detrended result, i.e., "residual"
- `trend` - the estimated "trend" which was subtracted from the actual elevation to obtain `resid`

Examples

```
# Small example
data(dem)
dem1 = detrend(dem)
dem1 = st_redimension(dem1)
dem1 = st_set_dimensions(dem1, 3, values = c("resid", "trend"))
plot(round(dem1), text_values = TRUE, col = terrain.colors(11))

# Larger example 1
data(carmel)
carmel1 = detrend(carmel, parallel = 2)
carmel1 = st_redimension(carmel1)
carmel1 = st_set_dimensions(carmel1, 3, values = c("resid", "trend"))
```

```
plot(carmel1, col = terrain.colors(11))

# Larger example 2
data(golan)
golan1 = detrend(golan, parallel = 2)
golan1 = st_redimension(golan1)
golan1 = st_set_dimensions(golan1, 3, values = c("resid", "trend"))
plot(golan1, col = terrain.colors(11))
```

`dist_to_nearest` *Calculate raster of distances to nearest feature*

Description

Given a `stars` raster and an `sf` vector layer, returns a new raster with the distances of each cell centroid to the nearest feature in the vector layer.

Usage

```
dist_to_nearest(x, v, progress = TRUE)
```

Arguments

`x`	A `stars` layer, used as a "grid" for distance calculations
`v`	An `sf`, `sfc` or `sfg` object
`progress`	Display progress bar? The default is `TRUE`

Value

A `stars` raster with distances to nearest feature

Examples

```
# Sample 'sf' layer
x = st_point(c(0,0))
y = st_point(c(1,1))
x = st_sfc(x, y)
x = st_sf(x)
x = st_buffer(x, 0.5)

# Make grid
r = make_grid(x, res = 0.1, buffer = 0.5)
d = dist_to_nearest(r, x, progress = FALSE)

# Plot
plot(d, breaks = "equal", axes = TRUE, reset = FALSE)
plot(st_geometry(x), add = TRUE, pch = 4, cex = 3)
```

extract2 *Extract raster values by lines or polygons*

Description

Extract raster values by lines or polygons, summarizing for each feature using a function specified by the user. This function is aimed to reproduce (some of) the functionality of `raster::extract`.

Usage

```
extract2(x, v, fun, progress = TRUE, ...)
```

Arguments

x	A `stars` object
v	An `sf` layer that determines values to extract
fun	A function to summarize cell values per feature/band
progress	Display progress bar? The default is `TRUE`
...	Further arguments passed to `fun`

Value

A vector (single-band raster) or `matrix` (multi-band raster) with the extracted and summarized values

Examples

```
# Polygons
pol = st_bbox(landsat)
pol = st_as_sfc(pol)
set.seed(1)
pol = st_sample(pol, 5)
pol = st_buffer(pol, 100)
pol = c(pol, pol)

# Plot
plot(landsat[,,,1,drop=TRUE], reset = FALSE)
plot(pol, add = TRUE)

# Single-band raster
aggregate(landsat[,,,1,drop=TRUE], pol, mean, na.rm = TRUE)[[1]] ## Duplicated areas get 'NA'
extract2(landsat[,,,1,drop=TRUE], pol, mean, na.rm = TRUE, progress = FALSE)

# Multi-band example
extract2(landsat, pol, mean, na.rm = TRUE, progress = FALSE)

# Lines
lines = st_cast(pol, "LINESTRING")
```

```
# Single-band raster
extract2(landsat[,,,1,drop=TRUE], lines, mean, na.rm = TRUE, progress = FALSE)

# Multi-band example
extract2(landsat, lines, mean, na.rm = TRUE, progress = FALSE)
```

`flowlength` *Calculate flow length*

Description

Calculates flow length for each pixel in a Digital Elevation Model (DEM) raster. Inputs and output are rasters of class `stars`, single-band (i.e., only ‘"x"‘ and ‘"y"‘ dimensions), with one attribute.

Usage

```
flowlength(elev, veg, progress = TRUE)
```

Arguments

`elev`	A numeric `stars` raster representing a Digital Elevation Model (DEM).
`veg`	A matching logical `stars` raster representing vegetation presence. `TRUE` values represent vegetated cells where flow is absorbed (i.e. sinks), `FALSE` values represent cells where flow is unobstructed.
`progress`	Display progress bar? The default is `TRUE`

Value

A numeric `stars` raster where each cell value is flow length, in resolution units.

References

The algorithm is described in:

Mayor, A. G., Bautista, S., Small, E. E., Dixon, M., & Bellot, J. (2008). Measurement of the connectivity of runoff source areas as determined by vegetation pattern and topography: A tool for assessing potential water and soil losses in drylands. Water Resources Research, 44(10).

Examples

```
# Example from Fig. 2 in Mayor et al. 2008

elev = rbind(
  c(8, 8, 8, 8, 9, 8, 9),
  c(7, 7, 7, 7, 9, 7, 7),
  c(6, 6, 6, 6, 6, 5, 7),
  c(4, 5, 5, 3, 5, 4, 7),
```

```
  c(4, 5, 4, 5, 4, 6, 5),
  c(3, 3, 3, 3, 2, 3, 3),
  c(2, 2, 2, 3, 4, 1, 3)
)
veg = rbind(
  c(TRUE,  TRUE,  TRUE,  TRUE,  FALSE, FALSE, TRUE),
  c(TRUE,  TRUE,  TRUE,  TRUE,  TRUE,  FALSE, FALSE),
  c(FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE),
  c(FALSE, TRUE,  FALSE, FALSE, FALSE, FALSE, TRUE),
  c(TRUE,  TRUE,  FALSE, FALSE, FALSE, FALSE, FALSE),
  c(TRUE,  TRUE,  TRUE,  FALSE, FALSE, FALSE, FALSE),
  c(FALSE, TRUE,  TRUE,  FALSE, FALSE, TRUE,  TRUE)
)
elev = matrix_to_stars(elev)
veg = matrix_to_stars(veg)

# Calculate flow length
fl = flowlength(elev, veg, progress = FALSE)

# Plot
plot(
  round(elev, 1), text_values = TRUE, breaks = "equal",
  col = terrain.colors(6), main = "input (elevation)"
)
plot(
  veg*1, text_values = TRUE, breaks = "equal",
  col = rev(terrain.colors(2)), main = "input (vegetation)"
)
plot(
  round(fl, 1), text_values = TRUE, breaks = "equal",
  col = terrain.colors(6), main = "output (flowlength)"
)

# Larger example
data(carmel)
elev = carmel
elev[is.na(elev)] = 0
veg = elev > 100
fl = flowlength(elev, veg, progress = FALSE)
plot(fl, breaks = "equal", col = hcl.colors(11), main = "flowlength (m)")
```

`focal2` *Apply a focal filter on a raster*

Description

Applies a focal filter with weighted neighborhood `w` on a raster. The weights (`w`) can be added to, subtracted from, multiplied by or divided with the raster values (as specified with `weight_fun`). The focal cell is then taken as the mean, sum, minimum or maximum of the weighted values (as

specified with fun). Input and output are rasters of class stars, single-band (i.e., only ‘"x"‘ and ‘"y"‘ dimensions), with one attribute.

Usage

```
focal2(
  x,
  w,
  fun = "mean",
  weight_fun = "*",
  na.rm = FALSE,
  mask = FALSE,
  na_flag = -9999
)
```

Arguments

x	A raster (class stars) with one attribute and two dimensions: x and y, i.e., a single-band raster.
w	Weights matrix defining the neighborhood size around the focal cell, as well as the weights. For example, matrix(1,3,3) implies a neighborhood of size 3*3 with equal weights of 1 for all cells. The matrix must be square, i.e., with an odd number of rows and columns.
fun	A function to aggregate the resulting values for each neighborhood. Possible values are: "mean", "sum", "min", "max", and "mode". The default is "mean", i.e., the resulting values per neighborhood are *averaged* before being assigned to the new focal cell value.
weight_fun	An operator which is applied on each pair of values comprising the cell value and the respective weight value, as in raster_value-weight. Possible values are: "+", "-", "*", "/". The default is "*", i.e., each cell value is *multiplied* by the respective weight.
na.rm	Should NA values in the neighborhood be removed from the calculation? Default is FALSE.
mask	If TRUE, pixels with NA in the input are set to NA in the output as well, i.e., the output is "masked" using the input (default is FALSE).
na_flag	Value used to mark NA values in C code. This should be set to a value which is guaranteed to be absent from the input raster x (default is -9999).

Value

The filtered stars raster.

Note

The raster is "padded" with (nrow(w)-1)/2 more rows and columns of NA values on all sides, so that the neighborhood of the outermost rows and columns is still a complete neighborhood. Those rows and columns are removed from the final result before returning it. This means, for instance, that the outermost rows and columns in the result will be NA when using na.rm=FALSE.

References

The function interface was inspired by function `raster::focal`. The C code for this function is a modified and expanded version of the C function named `applyKernel` included with R package `spatialfil`.

Examples

```
# Small example
data(dem)
dem_mean3 = focal2(dem, matrix(1, 3, 3), "mean")
r = c(dem, round(dem_mean3, 1), along = 3)
r = st_set_dimensions(r, 3, values = c("input", "output (mean, k=3)"))
plot(r, text_values = TRUE, breaks = "equal", col = terrain.colors(11))

# Larger example
data(carmel)
carmel_mean15 = focal2(carmel, matrix(1, 15, 15), "mean")
r = c(carmel, carmel_mean15, along = 3)
r = st_set_dimensions(r, 3, values = c("input", "output (mean, k=15)"))
plot(r, breaks = "equal", col = terrain.colors(11))
```

`focal2r` *Apply a focal filter on a raster (R)*

Description

Applies a focal filter with neighborhood size `k*k` on a raster (class `stars`), using R code. This function is relatively slow, provided here mainly for testing purposes or for custom using functions which are not provided by `focal2`.

Usage

```
focal2r(x, w, fun, mask = FALSE, ...)
```

Arguments

`x`	A raster (class `stars`) with two dimensions: `x` and `y`, i.e., a single-band raster
`w`	Weights matrix defining the neighborhood size around the focal cell, as well as the weights. For example, `matrix(1,3,3)` implies a neighborhood of size 3*3 with equal weights of 1 for all cells. Focal cell values are multiplied by the matrix values before being passed to function `fun`. The matrix must be square, i.e., with an odd number of rows and columns.
`fun`	A function to be applied on each neighborhood, after it has been multiplied by the matrix. The function needs to accepts a vector (of length equal to `length(w)` and return a vector of length `1`

mask	If TRUE, pixels with NA in the input are set to NA in the output as well, i.e., the output is "masked" with the input (default FALSE)
...	Further arguments passed to fun

Value

The filtered stars raster

Note

The raster is "padded" with one more row/column of NA values on all sides, so that the neigborhood of the outermost rows and columns is still a complete 3x3 neighborhood. Those rows and columns are removed from the filtered result before returning it.

Examples

```
# Small example
data(dem)
dem1 = focal2r(dem, matrix(1,3,3), mean, na.rm = TRUE)
dem2 = focal2r(dem, matrix(1,3,3), min, na.rm = TRUE)
dem3 = focal2r(dem, matrix(1,3,3), max, na.rm = TRUE)
r = c(dem, round(dem1, 1), dem2, dem3, along = 3)
r = st_set_dimensions(r, 3, values = c("input", "mean", "min", "max"))
plot(r, text_values = TRUE, breaks = "equal", col = terrain.colors(10))
# Larger example
data(carmel)
carmel1 = focal2r(carmel, matrix(1,3,3), mean, na.rm = TRUE, mask = TRUE)
carmel2 = focal2r(carmel, matrix(1,9,9), mean, na.rm = TRUE, mask = TRUE)
carmel3 = focal2r(carmel, matrix(1,15,15), mean, na.rm = TRUE, mask = TRUE)
r = c(carmel, carmel1, carmel2, carmel3, along = 3)
r = st_set_dimensions(r, 3, values = c("input", "k=3", "k=9", "k=15"))
plot(r, breaks = "equal", col = terrain.colors(100))
```

---

footprints *Footprints*

Description

Calculates a polygon layer with the footprints of raster images.

Usage

```
footprints(x)
```

Arguments

x	A character vector of raster file paths

Value

An sf layer with the footprints (i.e., bounding box polygons) of the rasters

Examples

```
# Create sample files
file1 = tempfile(fileext = ".tif")
file2 = tempfile(fileext = ".tif")
file3 = tempfile(fileext = ".tif")
r1 = landsat[,1:100, 1:100,]
r2 = landsat[,101:200, 101:200,]
r3 = landsat[,21:40, 51:120,]
write_stars(r1, file1)
write_stars(r2, file2)
write_stars(r3, file3)

# Calculate footprints
files = c(file1, file2, file3)
pol = footprints(files)
pol
```

---

golan *Digital Elevation Model of Golan Heights*

Description

A stars object representing a Digital Elevation Model (DEM) Digital Elevation Model of part of the Golan Heights and Lake Kinneret, at 90m resolution

Usage

```
golan
```

Format

A stars object with 1 attribute:

elevation Elevation above sea level, in meters

Examples

```
plot(golan, breaks = "equal", col = terrain.colors(11))
```

landsat *RGB image of Mount Carmel*

Description

A stars object representing an RGB image of part of Mount Carmel, at 30m resolution. The data source is Landsat-8 Surface Reflectance product.

Usage

```
landsat
```

Format

A stars object with 1 attribute:

refl Reflectance, numeric value between 0 and 1

Examples

```
plot(landsat, breaks = "equal")
```

layer_to_matrix *Get* stars *layer values as matrix*

Description

Extracts the values of a single layer in a stars object to a matrix.

Usage

```
layer_to_matrix(x, check = TRUE)
```

Arguments

x	A stars raster with one attribute and two dimensions, x and y, i.e., a single-band raster.
check	Whether to check (and fix if necessary) that input has one attribute, one layer and x-y as dimensions 1-2 (default is TRUE).

Value

A matrix with the layer values, having the same orientation as the raster (i.e., rows represent the y-axis and columns represent the x-axis).

Examples

```
data(dem)
m = layer_to_matrix(dem)
m
```

`layer_to_vector` *Get* `stars` *layer values as vector*

Description

Extracts the values of a single layer in a `stars` object to a vector. Cell values are ordered from top-left corner to the right.

Usage

```
layer_to_vector(x, check = TRUE)
```

Arguments

`x`	A raster (class `stars`) with two dimensions: `x` and `y`, i.e., a single-band raster.
`check`	Whether to check (and fix if necessary) that input has one attribute, one layer and x-y as dimensions 1-2 (default is `TRUE`).

Value

A vector with cell values, ordered by rows, starting from the top left corner (north-west) and to the right.

Examples

```
data(dem)
v = layer_to_vector(dem)
v
```

make_grid *Make 'stars' grid from 'sf' layer*

Description

Create 'stars' raster grid from bounding box of 'sf' vector layer, possibly buffered, with specified resolution.

Usage

```
make_grid(x, res, buffer = 0)
```

Arguments

x	An sf, sfc or sfg object
res	Required grid resolution, in CRS units of x
buffer	Buffer size around x (default is 0, i.e., no buffer)

Value

A stars raster with the grid, with all cell values equal to 1

Examples

```
# Sample 'sf' layer
x = st_point(c(0,0))
y = st_point(c(1,1))
x = st_sfc(x, y)
x = st_sf(x)

# Make grid
r = make_grid(x, res = 0.1, buffer = 0.5)
r[[1]][] = rep(1:3, length.out = length(r[[1]]))

# Plot
plot(r, axes = TRUE, reset = FALSE)
plot(st_geometry(x), add = TRUE, pch = 4, cex = 3, col = "red")
```

matrix_extend *Extend matrix*

Description

Adds n rows and columns with NA values on all sides of a matrix.

Usage

```
matrix_extend(m, n = 1, fill = NA)
```

Arguments

m	A matrix
n	By how many rows/columns to extend the matrix on each side?
fill	Fill value (default is NA)

Value

An extended matrix

Examples

```
m = matrix(1:6, nrow = 2, ncol = 3)
m
matrix_extend(m, 1)
matrix_extend(m, 2)
matrix_extend(m, 3)
```

---

matrix_get_neighbors *Get neighboring cell values for given matrix cell*

Description

Get the values of a k*k neighborhood, as vector and by row, given a matrix, k, and focal cell position (row and column).

Usage

```
matrix_get_neighbors(m, pos, k = 3)
```

Arguments

m	A matrix.
pos	The focal cell position, a numeric vector of length two of the form c(row, column).
k	Neighborhood size around the focal cell. For example, k=3 implies a neighborhood of size 3*3. Must be an odd positive integer.

Value

A vector with cell values, ordered by rows, starting from the top left corner of the neighborhood and to the right. When neighborhood extends beyond matrix bounds, only the "existing" values are returned.

Examples

```
m = matrix(1:12, nrow = 3, ncol = 4)
m
matrix_get_neighbors(m, pos = c(2, 2), k = 3)
matrix_get_neighbors(m, pos = c(2, 2), k = 5)
matrix_get_neighbors(m, pos = c(2, 2), k = 7)  # Same result
```

---

matrix_to_stars *Convert* matrix *to* stars

Description

Converts matrix to a single-band stars raster, conserving the matrix orientation where rows become the y-axis and columns become the y-axis. The bottom-left corner of the axis is set to (0,0) coordinate, so that x and y coordinates are positive across the raster extent.

Usage

```
matrix_to_stars(m, res = 1)
```

Arguments

m	A matrix
res	The cell size, default is 1

Value

A stars raster

Examples

```
data(volcano)
r = matrix_to_stars(volcano, res = 10)
plot(r)
```

matrix_trim *Trim matrix*

Description

Removes n rows and columns with NA values on all sides of a matrix.

Usage

```
matrix_trim(m, n = 1)
```

Arguments

m	A matrix
n	By how many rows/columns to trim the matrix on each side?

Value

A trimmed matrix, or NULL if trimming results in an "empty" matrix.

Examples

```
m = matrix(1:80, nrow = 8, ncol = 10)
m
matrix_trim(m, 1)
matrix_trim(m, 2)
matrix_trim(m, 3)
matrix_trim(m, 4)
```

mode_value *Mode*

Description

Find the mode (i.e., most common value) in a numeric vector. In case of ties, the first encountered value is returned.

Usage

```
mode_value(x, na_flag = -9999)
```

Arguments

x	A numeric or logical vector
na_flag	Value used to mark NA values in C code. This should be set to a value which is guaranteed to be absent from the input vector x (default is -9999).

Value

The mode, numeric vector of length 1

Examples

```
x = c(3, 2, 5, 5, 3, 10, 2, 5)
mode_value(x)
```

---

normalize_2d *Normalize a 2D 'stars' object*

Description

Check, and possibly correct, that the input stars object:

- Has exactly one attribute.
- Has exactly two dimensions.
- The dimensions are spatial, named x and y (in that order).

Usage

```
normalize_2d(x)
```

Arguments

x	A stars object

Value

A new stars object

Examples

```
# Small example
data(dem)
normalize_2d(dem)
```

normalize_3d *Normalize a 3D 'stars' object*

Description

Check, and possibly correct, that the input `stars` object:

- Has exactly one attribute.
- Has exactly three dimensions.
- The first two dimensions are spatial, named `x` and `y` (in that order).

Usage

```
normalize_3d(x)
```

Arguments

`x`	A `stars` object

Value

A new `stars` object

Examples

```
# Small example
data(landsat)
normalize_3d(landsat)
```

rgb_to_greyscale *Convert RGB to greyscale*

Description

Convert a 3-band RGB raster to 1-band greyscale raster. Based on `wvtool::rgb2gray`.

Usage

```
rgb_to_greyscale(x, rgb = 1:3, coefs = c(0.3, 0.59, 0.11))
```

Arguments

`x`	A three-dimensional `stars` object with RGB values
`rgb`	Indices of RGB bands, default is `c(1, 2, 3)`
`coefs`	RGB weights, default is `c(0.30,0.59,0.11)`

Value

A two-dimensional stars object greyscale values

Examples

```
data(landsat)
plot(landsat, rgb = 1:3)
landsat_grey = rgb_to_greyscale(landsat)
plot(landsat_grey, breaks = "equal")
```

---

slope *Calculate topographic slope from a DEM*

Description

Calculates topographic slope given a Digital Elevation Model (DEM) raster. Input and output are rasters of class stars, single-band (i.e., only ‘"x"‘ and ‘"y"‘ dimensions), with one attribute.

Usage

```
slope(x, na_flag = -9999)
```

Arguments

x	A raster (class stars) with two dimensions: x and y, i.e., a single-band raster, representing a DEM.
na_flag	Value used to mark NA values in C code. This should be set to a value which is guaranteed to be absent from the input raster x (default is -9999).

Value

A stars raster with topographic slope, i.e., the azimuth where the terrain is tilted towards, in decimal degrees (0-360) clockwise from north.

Note

Slope calculation results in NA when at least one of the cell neighbors is NA, including the outermost rows and columns. Given that the focal window size in slope calculation is 3*3, this means that the outermost one row and one column are given an slope value of NA.

The raster must be in projected CRS, and units of x/y resolution are assumed to be the same as units of elevation (typically *meters*).

References

The topographic slope algorithm is based on the *How slope works* article in the ArcGIS documentation:

https://desktop.arcgis.com/en/arcmap/10.3/tools/spatial-analyst-toolbox/how-slope-works.htm

Examples

```
# Small example
data(dem)
dem_slope = slope(dem)
plot(
  dem, text_values = TRUE, breaks = "equal",
  col = hcl.colors(11, "Spectral"), main = "input (elevation)"
)

plot(
  dem_slope, text_values = TRUE, breaks = "equal",
  col = hcl.colors(11, "Spectral"), main = "output (slope)"
)
# Larger example
data(carmel)
carmel_slope = slope(carmel)
plot(
  carmel, breaks = "equal",
  col = hcl.colors(11, "Spectral"), main = "input (elevation)"
)
plot(
  carmel_slope, breaks = "equal",
  col = hcl.colors(11, "Spectral"), main = "output (slope)"
)
```

---

trim2 *Remove empty outer rows and columns*

Description

Removes complete outer rows and columns which have NA values.

Usage

```
trim2(x)
```

Arguments

x	A two-dimensional stars object

Value

A new stars object with empty outer rows and columns removed

Examples

```
# Single-band example
data(dem)
dem[[1]][1,] = NA
dem1 = trim2(dem)

# Multi-band example
data(landsat)
landsat[[1]][1:100,,] = NA
landsat1 = trim2(landsat)
```

---

w_azimuth *Create matrix with azimuths to center*

Description

Creates a `matrix` with directions (i.e., azimuth) to central cell, of specified size `k`. The matrix can be used as weight matrix when calculating the convergence index (see Examples).

Usage

```
w_azimuth(k)
```

Arguments

`k`	Neighborhood size around focal cell. Must be an odd number. For example, `k=3` implies a 3*3 neighborhood.

Value

A `matrix` where each cell value is the azimuth from that cell towards the matrix center.

Examples

```
m = w_azimuth(3)
m
m = w_azimuth(5)
m
```

w_circle *Create matrix with circular weight pattern*

Description

Creates a `matrix` with where a circular pattern is filled with values of `1` and the remaining cells are filled with values of `0` (see Examples).

Usage

```
w_circle(k)
```

Arguments

`k`	Neighborhood size around focal cell. Must be an odd number. For example, `k=3` implies a 3*3 neighborhood.

Value

A `matrix` with a circular pattern.

Examples

```
m = w_circle(3)
image(m, asp = 1, axes = FALSE)
m = w_circle(5)
image(m, asp = 1, axes = FALSE)
m = w_circle(15)
image(m, asp = 1, axes = FALSE)
m = w_circle(35)
image(m, asp = 1, axes = FALSE)
m = w_circle(91)
image(m, asp = 1, axes = FALSE)
m = w_circle(151)
image(m, asp = 1, axes = FALSE)
```

Index

∗ **datasets**
carmel, 4
dem, 5
golan, 14
landsat, 15

aspect, 2, *4*

carmel, 4
CI, 4

dem, 5
detrend, 6
dist_to_nearest, 7

extract2, 8

flowlength, 9
focal2, 10
focal2r, 12
footprints, 13

golan, 14

landsat, 15
layer_to_matrix, 15
layer_to_vector, 16

make_grid, 17
matrix_extend, 17
matrix_get_neighbors, 18
matrix_to_stars, 19
matrix_trim, 20
mode_value, 20

normalize_2d, 21
normalize_3d, 22

rgb_to_greyscale, 22

slope, 23

trim2, 24

w_azimuth, 25
w_circle, 26