Package: windscape (via r-universe)

October 27, 2026

Type Package

Title Model Landscape Connectivity by Wind

Version 2.0.0

Description Model landscape connectivity by wind for ecological and evolutionary analyses. Summarize long time series of gridded wind data, downloaded from reanalyses or supplied by the user, into wind roses describing directional conductance between neighboring grid cells. Model dispersal through them with least-cost path and random walk algorithms, to map windsheds around sites and estimate pairwise connectivity among them, test hypotheses relating wind to ecological data, and visualize wind and connectivity with 'ggplot2'. The approach builds on Kling and Ackerly (2020) <doi:10.1038/s41558-020-0848-3> and Kling and Ackerly (2021) <doi:10.1073/pnas.2017317118>.

Encoding UTF-8

LazyData true

LazyDataCompression xz

LinkingTo Rcpp

Imports gdistance, Rcpp, raster, geosphere, ggplot2, igraph, magrittr, Matrix, methods, stats, tools, utils

URL https://matthewkling.github.io/windscape/, https://github.com/matthewkling/windscape

BugReports https://github.com/matthewkling/windscape/issues

Suggests knitr, maps, ncdf4, rmarkdown, scales, sf, testthat (>= 3.0.0), vegan, tidyverse, withr

VignetteBuilder knitr

Depends R (>= 4.0.0), terra (>= 1.8-42)

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

License MIT + file LICENSE

Config/Needs/website rmarkdown

Collate 'RcppExports.R' 'circ_sd.R' 'datasets.R' 'distance.R' 'download_wind_rose.R' 'downscale.R' 'geom_wind_arrow.R' 'geom_wind_rose.R' 'geom_wind_trail.R' 'ggplot.R' 'least_cost.R' 'mantel.R' 'wind_rose.R' 'wind_field.R' 'methods.R' 'ncar.R' 'net_flow.R' 'pairwise_matrices.R' 'pairwise_random_walk.R' 'particles.R' 'random_walk.R' 'random_walk_paths.R' 'rose.R' 'utils.R' 'weight_conductance.R' 'wind_graph.R' 'windscape-package.R' 'ws_summarize.R'

RoxygenNote 8.0.0

NeedsCompilation yes

Author Matthew Kling [aut, cre, cph]

Maintainer Matthew Kling <mattkling@berkeley.edu>

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

Date/Publication 2026-10-27 15:00:02 UTC

RemoteUrl https://github.com/cran/windscape

RemoteRef HEAD

RemoteSha 3e969923524a77aba1833ca88cf144beedbc1e8c

Contents

birch . . . 3
cell_distance . . . 4
check_cell_distance . . . 4
combine_roses . . . 5
download_land_mask . . . 6
download_wind_data . . . 7
download_wind_rose . . . 9
downscale . . . 10
fortify.windscape . . . 11
generate_particles . . . 13
geoCorrection,wind_graph,ANY-method . . . 14
geom_wind_arrow . . . 14
geom_wind_path . . . 17
geom_wind_rose . . . 18
geom_wind_trail . . . 21
iter_length . . . 24
least_cost . . . 25
least_cost_paths . . . 26
mantel_test . . . 28
mean,wind_series-method . . . 29
net_flow . . . 30
pairwise_least_cost . . . 32
pairwise_means . . . 33
pairwise_random_walk . . . 34

pairwise_ratios . . . 36
point_distance . . . 37
random_walk . . . 37
random_walk_paths . . . 43
rw_exit_prob . . . 45
rw_max_step . . . 47
rw_self_retention . . . 48
scale_fill_bearing . . . 49
subset_series . . . 50
weight_conductance . . . 51
wind_field . . . 52
wind_field-class . . . 52
wind_graph . . . 53
wind_graph-class . . . 54
wind_rose . . . 54
wind_rose-class . . . 56
wind_rose-grid . . . 56
wind_rose_cache . . . 57
wind_rose_catalog . . . 57
wind_series . . . 58
wind_series-class . . . 59
wind_times . . . 59
wind_trails . . . 60
windscape-layers . . . 62
windscape_example . . . 63
ws_summarize . . . 64

Index **66**

`birch` *Silver birch landscape genetic data from Tsuda et al. (2017)*

Description

An example landscape genetic dataset for the species silver birch (Betula pendula) across 30 sampling sites in Asia, originally published by Tsuda et al. (2017).

Usage

```
birch
```

Format

`birch` A list with 4 entries:

sites A matrix with two columns giving the longitude and latitude of each site.

div A numeric vector giving the allelic richness sampled at each site.

mig A square, asymmetric matrix with estimated gene flow rates for each pair of sites.

fst A square, symmetric matrix with Fst values for each pair of sites.

Source

Y. Tsuda, V. Semerikov, F. Sebastiani, G. G. Vendramin, M. Lascoux, Multispecies genetic structure and hybridization in the Betula genus across Eurasia. Molecular Ecology 26, 589-605 (2017). doi:10.1111/mec.13885

---

cell_distance *Pairwise distances between cell centroids*

Description

Calculate the pairwise distances between the centroids of the raster grid cells that a set of points fall into. This is useful as an alternative to the distances between the points themselves, if distances are going to be compared to other metrics that are based on cell centers (e.g. wind connectivity).

Usage

```
cell_distance(x, ll)
```

Arguments

x	SpatRaster
ll	two-column matrix of site coordinates

Value

Pairwise distances between the cell centroids, in km

Examples

```
rose <- windscape_example("wind_rose")
sites <- cbind(c(-110, -100, -95), c(35, 40, 45))
cell_distance(rose, sites)
```

---

check_cell_distance *Check how grid cells distort distances among sites*

Description

Compares pairwise distances among sites with distances between the centers of the grid cells the sites fall in, and prints a report. Connectivity models that work from cell to cell treat each site as the center of its cell: the random walk functions (`random_walk()`, `pairwise_random_walk()`), and the least-cost functions when sites are snapped to cell centers (`least_cost()`, `least_cost_paths()`, and `pairwise_least_cost(snap = TRUE)`). For these, sites separated by only a few cells have distorted distances and directions, and sites in the same cell can't be distinguished at all. `pairwise_least_cost()` with its default `snap = FALSE` places sites at their actual locations, so this check does not apply to it.

Usage

```
check_cell_distance(x, ll, return = FALSE)
```

Arguments

x	A SpatRaster on the grid used for connectivity modeling, e.g. a wind_rose.
ll	A two-column matrix of site coordinates.
return	Logical: return the matrix of ratios? Default FALSE, which only prints the report.

Details

Where many site pairs are affected, options are to use wind data on a finer grid, or to downscale() the wind rose (see its documentation for how downscaling changes random walk results).

Value

Prints the number of site pairs, the number (and percentage) in the same grid cell, and the distribution of discrepancies between cell and point distances, as percentages of point distance. If return = TRUE, also returns a matrix of the ratios of cell distances to point distances (NaN for a site with itself, and 0 for distinct sites in the same cell).

Examples

```
rose <- windscape_example("wind_rose")
sites <- cbind(c(-110, -109.9, -100), c(35, 35.1, 40)) # the first two share a cell
check_cell_distance(rose, sites)
```

---

combine_roses — *Combine wind roses built from different time periods*

Description

Combines wind roses for the same grid, built from different sets of time steps (e.g. separate months or years), into a single rose representing all of those time steps. Because a wind rose is an average over time steps, the result is the mean of the input roses weighted by their numbers of time steps, and equals the rose that would be built from all the time steps at once.

Usage

```
combine_roses(...)
```

Arguments

...	Two or more wind_rose objects, or a single list of them. They must share the same grid and the same trans function, and each must have a known number of time steps (n_steps), as roses built with wind_rose() do.

Value

A `wind_rose` whose `n_steps` is the total across the inputs.

See Also

`wind_rose()`, which uses this function to build roses from long series in chunks.

Examples

```
series <- windscape_example("wind_series")
n <- series@n_steps
first <- wind_rose(subset_series(series, steps = 1:(n %/% 2)))
second <- wind_rose(subset_series(series, steps = (n %/% 2 + 1):n))
both <- combine_roses(first, second)

# the same as building one rose from all the time steps
all.equal(terra::values(both), terra::values(wind_rose(series)))
```

---

download_land_mask *Download a land-water layer from NCAR*

Description

Downloads a land layer matching the grid of `download_wind_data()` data for the same `source` and bounding box, e.g. for weighting a wind rose to reduce conductance over water.

Usage

```
download_land_mask(source = c("era5", "cfsr", "cfsv2"), xlim, ylim)
```

Arguments

`source`	Data set: `"era5"` or `"cfsr"`. (Not yet available for `"cfsv2"`.)
`xlim`, `ylim`	Bounding box; see `download_wind_data()`.

Value

A single-layer `SpatRaster` named `"land"`. For ERA5 this is the land fraction of each cell, from 0 (all water) to 1 (all land); use e.g. `land >= 0.5` for a binary layer. For CFSR it is binary, 1 for land and 0 for water. (CFSR publishes no land mask, so it is derived from which cells have soil temperature data.)

Examples

```
land <- download_land_mask("era5", xlim = c(-125, -115), ylim = c(42, 49))
```

`download_wind_data` *Download hourly wind data from NCAR*

Description

Downloads gridded hourly wind data from the NCAR Geoscience Data Exchange (GDEX) for a bounding box and a set of months, optionally limited to particular days and hours, saving one GeoTIFF file per month. No account is needed. Data are clipped to the bounding box and the requested times on the server, so only the requested data are transferred.

Usage

```
download_wind_data(
  source = c("era5", "cfsr", "cfsv2"),
  level = "10m",
  xlim,
  ylim,
  years,
  months = 1:12,
  days = NULL,
  hours = NULL,
  dir = tempdir(),
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

`source` Data set to download:

- `"era5"` (the default): ECMWF ERA5 reanalysis, 0.25 degree grid, 1940 to present (NCAR data set d633000).
- `"cfsr"`: NCEP Climate Forecast System Reanalysis, ~0.31 degree Gaussian grid, 1979 to 2010 (d093001).
- `"cfsv2"`: NCEP Climate Forecast System version 2, the operational continuation of CFSR, ~0.2 degree Gaussian grid, 2011 to present (d094001).

`level` Height of the wind data. `"10m"` (the default) is wind 10 m above the ground, available for all sources. `"100m"` is available for ERA5 only. Pressure levels `"1000hPa"`, `"850hPa"`, `"700hPa"`, `"500hPa"`, and `"200hPa"` are available for CFSR and CFSv2 only.

`xlim` Numeric vector of length 2 giving the western and eastern limits of the bounding box, in degrees. Longitudes can be given either in the -180 to 180 range or the 0 to 360 range, and output files use the same convention. Boxes crossing the prime meridian (in -180 to 180 coordinates, e.g. `c(-10, 30)`) or the antimeridian (in 0 to 360 coordinates, e.g. `c(170, 200)`) are supported.

`ylim` Numeric vector of length 2 giving the southern and northern limits of the bounding box, in degrees, in the range -90 to 90.

years	Integer vector of years to download.
months	Integer vector of months (1-12) to download within each year. Defaults to all months.
days	Integer vector of days of the month (1-31) to download, by the UTC date of each time step. NULL (the default) downloads all days. Months with none of the requested days (e.g. day 31 in a 30-day month) are skipped.
hours	Integer vector of hours of the day (0-23, UTC) to download. NULL (the default) downloads all hours; for example, hours = seq(0, 21, 3) downloads every third hour, which reduces download time and file size about threefold. Days and hours refer to the UTC date and time of each time step, within each requested month. (CFSR and CFSv2 monthly files hold hourly forecasts valid from 01:00 on the first through 00:00 on the first of the following month. With days or hours, time steps are assigned to their calendar month, so 00:00 on the first comes from the previous month's file; without them, each month's file is downloaded as is, as in the pre-built roses of download_wind_rose().)
dir	Directory where monthly files are saved. Defaults to a temporary directory, which is deleted at the end of the R session; set it to a permanent location to keep the data and make use of caching across sessions.
overwrite	Logical. If FALSE (the default), months already downloaded to dir are skipped.
quiet	Logical. If TRUE, progress messages are suppressed.

Details

Each output file holds one month of data in wind_series layout: all u layers followed by all v layers, named like "u 2005-08-28 23:00:00" (UTC), in m/s, oriented to true east and north, on a longitude/latitude grid. Combine the files into one series with wind_series(), which keeps the data on disk, and summarize it with wind_rose().

Downloads are cached: a month whose file already exists in dir is not downloaded again unless overwrite = TRUE, so an interrupted download can be resumed by rerunning the same call. Large requests (many years, or a large region) can take a long time and use substantial disk space; selecting hours with hours (e.g. every third hour) reduces both. The server limits the size of each request (about 100 MB), so large requests are split into several, by time, and reassembled.

Requires the **ncdf4** package.

Value

A character vector of file paths, one per month, in chronological order (returned invisibly if all files were already cached). Files for a subset of days or hours have names ending in a tag recording the selection (e.g. _d28_h23), so they are cached separately from whole months.

See Also

wind_series() to load the files; wind_rose() to summarize them; download_land_mask() to download a matching land-water layer; download_wind_rose() for pre-built CFSR wind roses, which need no wind data downloads.

Examples

```
# every third hour of 10 m ERA5 wind for summer 2020, Pacific Northwest
files <- download_wind_data("era5", xlim = c(-125, -115), ylim = c(42, 49),
                            years = 2020, months = 6:8, hours = seq(0, 21, 3))
rose <- wind_rose(wind_series(files))

# a single hour: Hurricane Katrina, 2005-08-28 23:00 UTC
f <- download_wind_data("era5", xlim = c(-99, -78), ylim = c(17, 35),
                        years = 2005, months = 8, days = 28, hours = 23)
katrina <- wind_field(wind_series(f))
```

---

download_wind_rose *Download a pre-built wind rose*

Description

Downloads a pre-built global wind rose for any combination of years and calendar months, optionally cropped to a region, so that common analyses need no wind data downloads or rose building. Roses are built from hourly 10 m CFSR wind for 1979-2010 with `trans = 1`; see `wind_rose_catalog()` for what is available. For other data sets, heights, periods, or `trans` functions, download wind data with `download_wind_data()` and build a rose with `wind_rose()`.

Usage

```
download_wind_rose(
  year = NULL,
  month = NULL,
  ext = NULL,
  source = "cfsr",
  level = "10m",
  cache = is.null(ext),
  quiet = FALSE
)
```

Arguments

`year`	Integer vector of years to include. `NULL` (the default) includes all years.
`month`	Integer vector of calendar months (1-12) to include in each year. `NULL` (the default) includes all months. For example, `year = 1990:1999, month = 6:8` gives a rose for the summers of the 1990s.
`ext`	Optional region to crop to: a `SpatExtent`, an object `terra::ext()` accepts (e.g. a `SpatRaster` or `SpatVector`), or a numeric vector `c(xmin, xmax, ymin, ymax)`, in degrees, with longitudes from -180 to 180. The result includes every cell overlapping `ext`. Regions crossing the antimeridian are not supported; for those, use the global rose.
`source, level`	Wind data set and height. Currently only `"cfsr"` and `"10m"`.

cache	Logical. If TRUE, whole files are downloaded and kept in a cache on disk (see wind_rose_cache()), so later requests for them need no download; each file is about 15 MB. If FALSE, nothing is kept: when ext is given, only the cells within it are read over the internet, which is much faster than downloading whole files for small regions; otherwise files are downloaded to a temporary directory. Defaults to TRUE when ext is NULL and FALSE otherwise.
quiet	Logical. If TRUE, progress messages are suppressed.

Details

Pre-built roses are stored for single months, calendar years, calendar months across all years, and the full period. A request is met with the fewest of these, combined with combine_roses(), which weights them by their numbers of hours; the result equals the rose built from all of the requested hours at once.

Value

A wind_rose, with n_steps and trans recorded so it can be combined with other roses. Values are wind conductance in 1 / hours.

See Also

wind_rose_catalog() for the available files; wind_rose_cache() to manage the cache.

Examples

```
# long-term rose for the western US, read remotely
rose <- download_wind_rose(ext = c(-125, -100, 30, 50))

# summers of 1990-1992
summer <- download_wind_rose(year = 1990:1992, month = 6:8, ext = c(-125, -100, 30, 50))
```

---

downscale *Downscale a wind rose to higher spatial resolution*

Description

Disaggregates a wind rose to a finer grid with terra::disagg(), interpolating between cells, and multiplies conductance by fact. Conductance is a rate per hop between neighboring cells, and hops on the finer grid are fact times shorter, so this rescaling keeps flow (conductance times the distance to each neighbor) unchanged: wind moves material at the same speed on the downscaled grid as on the original. Downscaling adds no wind information. It smooths the wind field between the original cell centers, but does not represent real fine-scale variation within cells, such as that caused by terrain.

Usage

```
downscale(x, fact, method = "bilinear")
```

Arguments

x	A wind_rose.
fact	A single integer: the factor by which to increase resolution. Each cell becomes fact x fact cells.
method	Interpolation method, passed to terra::disagg(): "bilinear" (the default) interpolates between the original cell centers; "near" gives each new cell the value of the cell it came from.

Details

How much downscaling changes results, and whether it improves them, depends on the connectivity model.

Least-cost analyses. pairwise_least_cost() places sites at their actual locations within cells by default (snap = FALSE), so it handles nearby sites without downscaling, and downscaling changes its results little: typically by a few percent, reflecting the smoother interpolated wind field. Downscaling does reduce error for nearby sites when sites are snapped to cell centers, as in pairwise_least_cost(snap = TRUE), least_cost(), and least_cost_paths().

Random walk analyses. The random walk functions treat each site as its cell, so downscaling separates nearby sites into distinct cells (see check_cell_distance()). But the spread of a random walk is numerical diffusion that scales with cell size (see random_walk()), so downscaling also narrows the dispersal kernel: in one test, downscaling by a factor of 4 reduced the spread of a stream-mode walk by 30 to 50 percent. Downscaling therefore changes the model as well as the grid. Results at different resolutions are not directly comparable, so choose one resolution for an analysis and treat it as part of the model specification.

Value

A wind_rose with fact times as many rows and columns as x, and the same trans and n_steps. Memory use and run time for connectivity analyses grow with the number of cells, roughly fact^2.

Examples

```
rose <- windscape_example("wind_rose")
fine <- downscale(rose, 2)
dim(rose)
dim(fine)
```

---

fortify.windscape *Convert windscape objects to data frames for ggplot2*

Description

fortify() methods that convert windscape objects to tidy data frames, so they can be passed directly to ggplot2::ggplot() or a layer's data argument. They can also be called directly, to inspect or modify the data before plotting.

Usage

```
## S3 method for class 'wind_rose'
fortify(model, data, na.rm = TRUE, ...)

## S3 method for class 'wind_field'
fortify(model, data, na.rm = TRUE, ...)

## S3 method for class 'wind_series'
fortify(model, data, na.rm = TRUE, ...)

## S3 method for class 'random_walk'
fortify(model, data, na.rm = TRUE, ...)
```

Arguments

`model`	A windscape object: a `wind_rose`, `wind_field`, `wind_series`, or the result of `random_walk()`.
`data`	Not used.
`na.rm`	Logical: drop grid cells with missing values? Default `TRUE`.
`...`	Not used.

Value

A data frame with grid cell center coordinates `x` and `y`, plus:

- `wind_rose`: one column of conductance per direction, `SW`, `W`, `NW`, `N`, `NE`, `E`, `SE`, `S`, and `total`, their sum. For longitude/latitude roses, also per-cell flow summaries matching those computed by `geom_wind_rose()`: `speed`, the mean wind speed in km/h (the sum of the flows toward the eight neighbors, if `trans = 1` and wind speeds are in m/s); `net`, the speed of net flow in km/h; `bearing`, its direction in degrees clockwise from north; and `consistency`, `net / speed`, the steadiness of wind direction over time (near 1 where wind blows predominantly one way, near 0 where it has no net direction).
- `wind_field`: wind components `u` and `v`; `speed`, in the units of `u` and `v`; and `bearing`, the direction the wind blows toward, in degrees clockwise from north.
- `wind_series`: one row per grid cell and time step, with `step` (the time step's index), `time` (parsed from layer names where possible, otherwise `NA`), and `u`, `v`, `speed`, and `bearing` as for a `wind_field`.
- `random_walk()` result, pulse mode: one row per grid cell and recorded iteration, with `iteration`, `hours` (elapsed time), `airborne`, and `deposition`.
- `random_walk()` result, stream mode: `residence`, `deposition`, and for upwind walks, `origin` (the `flux` element is a `wind_field`; fortify it separately).

Examples

```
rose <- windscape_example("wind_rose")
head(ggplot2::fortify(rose))
```

generate_particles *Generate initial particle locations*

Description

Initialize a set of particle locations for use in `wind_trails()`.

Usage

```
generate_particles(
  x,
  n = 1000,
  sample = "random",
  equalarea = TRUE,
  epsg = 8857,
  expand = 0.25
)
```

Arguments

`x`	A wind_field object.
`n`	Positive integer giving the total number of particles in the grid. If `sample = "random"` the result will have this exact number of particles, while if `sample = "grid"` it will be approximate.
`sample`	Sampling scheme, either "random" (the default) to generate points at random locations, or "grid" to generate points on a regular grid.
`equalarea`	Logical indicating whether to generate points on an equal area basis (TRUE, the default) or in lon-lat space (FALSE, which over-samples higher-latitude regions).
`epsg`	EPSG code for the projection in which to generate points. Only used if `equalarea = TRUE` and `sample = "grid"`. The default is `8857`, the Equal Area projection.
`expand`	Proportion by which to expand the bounding box around `x` after projection, to capture the full domain in projected space. Only used if `equalarea = TRUE` and `sample = "grid"`.

Value

A two-column matrix of x (longitude) and y (latitude) values, with a row for each particle.

Examples

```
field <- windscape_example("wind_field")
p <- generate_particles(field, n = 100)
head(p)
```

geoCorrection,wind_graph,ANY-method

geoCorrection is not applicable to wind graphs

Description

Wind graph conductances already account for the distances between cell centers (see wind_rose), so geoCorrection must not be applied to them.

Usage

```
## S4 method for signature 'wind_graph,ANY'
geoCorrection(x, type, ...)
```

Arguments

x	A wind_graph.
type, ...	Ignored.

Value

Always an error.

geom_wind_arrow

Wind field arrows

Description

Draws an arrow for each block of grid cells in a wind field, showing the direction and speed of the block's mean wind vector. stat_wind_arrow() averages wind components over blocks and computes arrow positions; geom_wind_arrow() draws them.

Usage

```
geom_wind_arrow(
  mapping = NULL,
  data = NULL,
  stat = StatWindArrow,
  position = "identity",
  ...,
  res = 20,
  scale = 1,
  fixed_length = FALSE,
  pivot = 0.5,
  arrow = grid::arrow(length = grid::unit(0.12, "cm"), type = "closed"),
```

```
  saturation = 0.5,
  center = FALSE,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

stat_wind_arrow(
  mapping = NULL,
  data = NULL,
  geom = GeomWindArrow,
  position = "identity",
  ...,
  res = 20,
  scale = 1,
  fixed_length = FALSE,
  pivot = 0.5,
  arrow = grid::arrow(length = grid::unit(0.12, "cm"), type = "closed"),
  saturation = 0.5,
  center = FALSE,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Aesthetic mappings created by ggplot2::aes(). x and y (cell center coordinates, in degrees longitude and latitude) must be mapped, usually in the plot's main call: ggplot(field, aes(x, y)). The wind components u (eastward) and v (northward) are mapped automatically to columns of the same names, as produced by fortify() on a wind_field (see fortify.windscape). If your columns are named differently, map them in this layer's mapping rather than in ggplot2::ggplot().
data	A data frame, or a wind_field, which is converted with fortify(). Default is to inherit the plot's data. The data must have one wind vector per grid cell in each panel and group, so to plot a wind_series, select a time step with subset_series() or facet by time.
stat, geom	Use to override the default pairing of stat_wind_arrow() and geom_wind_arrow().
position	Position adjustment; see ggplot2::layer().
...	Other arguments passed to ggplot2::layer(), such as fixed aesthetics like color = "white" or linewidth = 1.
res	Approximate number of arrows along the longer side of the data's extent. Blocks of grid cells are sized to be approximately square in km, and the same blocks are used in every panel. Default 20.
scale	Multiplier for the length of all arrows. At the default of 1, an arrow at the 90th percentile of speed across the layer (including all panels) is 0.9 times the spacing between arrows, or with fixed_length = TRUE, every arrow is.

`fixed_length` Logical: give all arrows the same length, so that length shows only direction? Default `FALSE` (length proportional to speed). Use with a speed mapping such as `aes(color = after_stat(speed))` or `aes(linewidth = after_stat(speed))` to encode speed another way. Blocks with zero mean wind get no arrow.

`pivot` Position of the arrow relative to its block center, as the fraction of its length that lies behind (upwind of) the center: 0.5 (the default) centers the arrow on the block, 0 starts it at the center.

`arrow` Arrowhead specification created by `grid::arrow()`, or `NULL` for none.

`saturation` If a `consistency` aesthetic is mapped (e.g. `aes(consistency = after_stat(consistency))`), arrow colors are blended toward gray where consistency is below this value. Default 0.5; set to 0 to disable blending.

`center` Logical: draw a point at each block center? Default `FALSE`. With `pivot = 0` and `arrow = NULL`, this gives a point-and-spoke plot.

`na.rm`, `show.legend`, `inherit.aes` See `ggplot2::layer()`.

Details

Wind components are averaged as vectors, so the arrow shows the block's net wind. Arrows are built in longitude/latitude coordinates, scaled so that their directions and relative lengths are correct in km at each block's latitude. Use `ggplot2::coord_quickmap()` or `ggplot2::coord_sf()` with a longitude/latitude CRS.

Value

A ggplot2 layer.

Computed variables

- `bearing`: direction of the block's mean wind vector, in degrees clockwise from north.
- `speed`: speed of the mean wind vector, in the units of `u` and `v`. This sets arrow length.
- `mean_speed`: mean of the wind speeds of the block's cells.
- `consistency`: `speed / mean_speed`, the spatial coherence of wind direction within the block; 1 where all cells' winds point the same way, lower where they diverge, converge, or rotate (as in a cyclone's eye), meaning the arrow understates the block's winds.
- `n`: number of grid cells in the block.
- `x0`, `y0`: block center, the mean coordinates of its cells.

Examples

```
library(ggplot2)
katrina <- windscape_example("wind_field")
# map background variables in the raster layer, not in ggplot(), so that the arrow layer
# doesn't inherit them
ggplot(katrina, aes(x, y)) +
  geom_raster(aes(fill = speed)) +
  scale_fill_gradient(low = "gray95", high = "gray40", name = "speed") +
```

```
  geom_wind_arrow() +
  coord_quickmap()

# fixed-length arrows, with speed shown by color
ggplot(katrina, aes(x, y)) +
  geom_wind_arrow(aes(color = after_stat(speed)), fixed_length = TRUE) +
  scale_color_viridis_c() +
  coord_quickmap()

# arrows colored by direction
ggplot(katrina, aes(x, y)) +
  geom_wind_arrow(aes(color = after_stat(bearing)), linewidth = 0.8) +
  scale_color_bearing() +
  coord_quickmap()
```

`geom_wind_path` *Draw existing wind trails or paths*

Description

Draws trails or paths you already have as data, such as the output of `wind_trails()` or `least_cost_paths()`, as lines with an arrowhead at the downwind end. To compute and draw trails from a wind field in one step, use `geom_wind_trail()` instead.

Usage

```
geom_wind_path(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  arrow = grid::arrow(length = grid::unit(0.1, "cm"), type = "closed"),
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

`mapping`	Aesthetic mappings created by `ggplot2::aes()`. `x` and `y` (coordinates, in degrees longitude and latitude) must be mapped, usually in the plot's main call. `group` and `t` are mapped automatically to the columns `trail` and `step`, as in `wind_trails()` and `least_cost_paths()` output: `group` identifies each trail, and `t` orders the points along it, from upwind to downwind. Map them in this layer if your columns are named differently.
`data`	A data frame of trail points. Default is to inherit the plot's data.
`stat`	The statistical transformation; the default, `"identity"`, draws the data as is.

position — Position adjustment; see ggplot2::layer().

... — Other arguments passed to ggplot2::layer(), such as fixed aesthetics like color = "white" or linewidth = 0.8.

arrow — Arrowhead drawn at the downwind end of each trail, created by grid::arrow(), or NULL for none.

na.rm, show.legend, inherit.aes — See ggplot2::layer().

Value

A ggplot2 layer.

See Also

geom_wind_trail() to compute trails from a wind field.

Examples

```
library(ggplot2)
katrina <- windscape_example("wind_field")
sites <- cbind(c(-92, -86), c(24, 30))
trails <- wind_trails(katrina, sites, hours = 12)

ggplot(trails, aes(x, y)) +
  geom_wind_path(aes(color = speed)) +
  coord_quickmap()
```

---

geom_wind_rose — *Wind rose glyphs: a map of local wind roses*

Description

Draws a star glyph for each block of grid cells in a wind rose, showing the strength of flow toward each of the eight neighbors. stat_wind_rose() does the computation: converting conductance to flow, averaging over blocks, and building glyph polygons. geom_wind_rose() draws them, coloring each glyph by its net flow direction and graying it out where flow has little net direction.

Usage

```
geom_wind_rose(
  mapping = NULL,
  data = NULL,
  stat = StatWindRose,
  position = "identity",
  ...,
  res = 15,
  scale = 1,
  normalize = FALSE,
```

```
  smooth = TRUE,
  saturation = 0.5,
  center = TRUE,
  bearing_scale = NULL,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

stat_wind_rose(
  mapping = NULL,
  data = NULL,
  geom = GeomWindRose,
  position = "identity",
  ...,
  res = 15,
  scale = 1,
  normalize = FALSE,
  smooth = TRUE,
  saturation = 0.5,
  center = TRUE,
  bearing_scale = NULL,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

`mapping`	Aesthetic mappings created by `ggplot2::aes()`. `x` and `y` (cell center coordinates, in degrees longitude and latitude) must be mapped, usually in the plot's main call: `ggplot(rose, aes(x, y))`. The eight conductance columns (`SW`, `W`, `NW`, `N`, `NE`, `E`, `SE`, `S`) are mapped automatically to columns of the same names, as produced by `fortify()` on a `wind_rose` (see fortify.windscape). If your columns are named differently, map them in this layer's `mapping` rather than in `ggplot2::ggplot()`: the automatic mappings are part of the layer, so they take precedence over plot-level ones.
`data`	A data frame, or a `wind_rose`, which is converted with `fortify()`. Default is to inherit the plot's data.
`stat, geom`	Use to override the default pairing of `stat_wind_rose()` and `geom_wind_rose()`.
`position`	Position adjustment; see `ggplot2::layer()`.
`...`	Other arguments passed to `ggplot2::layer()`, such as fixed aesthetics like `color = "black"`.
`res`	Approximate number of glyphs along the longer side of the data's extent. Blocks of grid cells are sized to be approximately square in km. The same blocks are used in every panel, so faceted glyphs are directly comparable. Default 15.

`scale` Multiplier for the size of all glyphs. At the default of 1, the longest ray of a typical large glyph (the 90th percentile across all glyphs in the layer, including all panels) is half the spacing between glyphs.

`normalize` Logical: scale each ray by the range of wind directions its neighbor receives? If `FALSE` (the default), rays show flow toward each neighbor, so the glyph shows how the grid's eight connections divide the wind. Away from the equator, neighbor bearings are unevenly spaced (the north and south neighbors lie closer to the diagonals), so even an isotropic wind draws as an east-west peanut. If `TRUE`, each ray is divided by the width of the range of directions allocated to that neighbor, in units of 45 degrees, giving flow per 45 degrees of wind direction: an isotropic wind draws as a circle at any latitude, which suits maps of wind regimes. Either way, directions are resolved only to the eight neighbor bearings, and the computed variables (`speed`, `net`, `bearing`, `consistency`) describe the unnormalized flows.

`smooth` Logical: join the ray tips with a smooth curve (the default) rather than straight lines? See Details.

`saturation` Consistency (see Computed variables) at and above which glyph fill is shown at full strength; glyphs with lower consistency are blended toward gray. Set to 0 to disable blending. Default 0.5.

`center` Logical: draw a point at each glyph's center? Default `TRUE`.

`bearing_scale` Logical: add `scale_fill_bearing()` to the plot? Default `TRUE` if `fill` is neither mapped nor set, otherwise `FALSE`.

`na.rm`, `show.legend`, `inherit.aes`
See `ggplot2::layer()`.

Details

Each glyph's eight rays point toward the eight neighbors, with length proportional to flow in that direction: conductance times the distance to the neighbor, averaged over the block, in km/h. Unlike conductance, which is a rate per grid cell, flow does not depend on cell size, so it can be averaged across cells and compared across latitudes. The flows sum to the mean wind speed, and their vector sum is the net flow (with `normalize = TRUE`, these hold only approximately for the drawn rays).

The outline passes through the tips of the eight rays. With `smooth = TRUE`, they are joined by a smooth curve: radius is interpolated as a function of bearing with a periodic, shape-preserving cubic (PCHIP). Between two adjacent rays the radius stays within the range of their lengths, so the outline adds no lobes that aren't in the rose. The curve between rays is a visual connector: the rose holds flow toward the eight neighbors only. With `smooth = FALSE`, the tips are joined by straight lines, giving a star-shaped polygon.

Glyphs are built in longitude/latitude coordinates, scaled so that their shapes are correct in km at each glyph's latitude. Use `ggplot2::coord_quickmap()` or `ggplot2::coord_sf()` with a longitude/latitude CRS so that the map's aspect ratio matches; in projected coordinates, glyph angles are distorted.

Value

A ggplot2 layer (with `bearing_scale = TRUE`, a list of a layer and a scale).

Computed variables

- `bearing`: direction of net flow, in degrees clockwise from north.
- `consistency`: net flow divided by total flow (mean resultant length); near 1 where wind blows predominantly one way, near 0 where it has no net direction.
- `speed`: total flow, the mean wind speed in km/h (if `trans = 1` in `wind_rose()` and wind speeds are in m/s).
- `net`: net flow speed, in km/h.
- `n`: number of grid cells in the glyph's block.
- `x0`, `y0`: glyph center, the mean coordinates of the cells in its block.

By default, `fill` is mapped to `after_stat(bearing)` and the `consistency` aesthetic to `after_stat(consistency)`.

Examples

```
library(ggplot2)
rose <- windscape_example("wind_rose")
ggplot(rose, aes(x, y)) +
  geom_wind_rose() +
  coord_quickmap()

# a background of mean wind speed: the glyphs use the fill scale, so show the background
# with alpha (or use a second fill scale, e.g. from the ggnewscale package)
ggplot(rose, aes(x, y)) +
  geom_raster(aes(alpha = speed), fill = "gray20") +
  geom_wind_rose() +
  coord_quickmap()

# faceting: glyphs use the same blocks and size scale in every panel
d <- rbind(cbind(fortify(rose), version = "original"),
           cbind(fortify(rose), version = "copy"))
ggplot(d, aes(x, y)) +
  geom_wind_rose(res = 10) +
  facet_wrap(~version) +
  coord_quickmap()
```

---

geom_wind_trail *Wind trails and streamlines*

Description

Draw trails that follow the airflow through a wind field, each with an arrowhead at its downwind end. Trails are computed from the field with `wind_trails()`, seeded on a regular grid (or at points you supply) and traced upwind and downwind; trails traced through a single wind field are streamlines. `geom_wind_trail()` and `stat_wind_trail()` make the same layer, differing only in which argument, `stat` or `geom`, can be changed. To draw trails you already have, such as `wind_trails()` or `least_cost_paths()` output, use `geom_wind_path()`.

Usage

```
geom_wind_trail(
  mapping = NULL,
  data = NULL,
  stat = StatWindTrail,
  position = "identity",
  ...,
  seeds = NULL,
  res = 20,
  fixed_length = FALSE,
  length = 1.5,
  hours = NULL,
  direction = c("both", "downwind", "upwind"),
  steps = 20,
  wrap = NULL,
  arrow = grid::arrow(length = grid::unit(0.1, "cm"), type = "closed"),
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

stat_wind_trail(
  mapping = NULL,
  data = NULL,
  geom = GeomWindTrail,
  position = "identity",
  ...,
  seeds = NULL,
  res = 20,
  fixed_length = FALSE,
  length = 1.5,
  hours = NULL,
  direction = c("both", "downwind", "upwind"),
  steps = 20,
  wrap = NULL,
  arrow = grid::arrow(length = grid::unit(0.1, "cm"), type = "closed"),
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Aesthetic mappings created by `ggplot2::aes()`. x and y (coordinates, in degrees longitude and latitude) must be mapped, usually in the plot's main call. The wind components u and v are mapped automatically to columns of the same names, as produced by `fortify()` on a `wind_field` (see fortify.windscape).
data	A `wind_field`, which is converted with `fortify()`, or a data frame like its

fortify() output. Default is to inherit the plot's data. The data must have one wind vector per grid cell in each panel and group, so to plot a wind_series, select a time step with subset_series() or facet by time.

stat, geom — Use to override the default pairing of stat and geom.

position — Position adjustment; see ggplot2::layer().

... — Other arguments passed to ggplot2::layer(), such as fixed aesthetics like color = "white" or linewidth = 0.8.

seeds — Starting points for trails. NULL (the default) seeds one trail at the center of each block of grid cells (see res). Alternatively, a two-column matrix of longitude and latitude, e.g. to trace trails from study sites; the same seeds are used in every panel.

res — Approximate number of seeds along the longer side of the data's extent, when seeds = NULL. Seeds are placed at the centers of blocks of grid cells that are approximately square in km. Also sets the spacing used by length. Default 20.

fixed_length — Logical: give all trails the same length, so they show only direction (streamlines)? Default FALSE: trails span the same transport time, so their length is proportional to wind speed, as for geom_wind_arrow(). In fields with very uneven speeds, TRUE shows flow structure more evenly, since trails in calm areas don't shrink.

length — Length of each trail, as a multiple of the spacing between seeds set by res. With fixed_length = FALSE, this is the length of a trail at the 90th percentile of wind speed, unless hours is given. Default 1.5.

hours — With fixed_length = FALSE, transport time in hours spanned by each trail, in place of length, so that trails show actual transport distance (assuming wind speeds in m/s). Default NULL.

direction — Trace trails "both" ways from each seed (the default), only "downwind", or only "upwind".

steps — Number of integration steps per trail; more gives smoother trails. Default 20.

wrap — Wrap trails across the field's edges; see wind_trails(). Default "neither".

arrow — Arrowhead drawn at the downwind end of each trail, created by grid::arrow(), or NULL for none.

na.rm, show.legend, inherit.aes — See ggplot2::layer().

Details

Trails are traced with wind_trails(), using bilinear interpolation of the wind field, and stop where they leave the field (unless wrapped), so trails seeded near the edges can be shorter; extend the field beyond the area of interest to avoid this. They are computed in longitude/latitude coordinates with correct directions and lengths in km; use ggplot2::coord_quickmap() or ggplot2::coord_sf() with a longitude/latitude CRS. Seeds and trail lengths are set once for the whole layer, so faceted panels are comparable.

Value

A ggplot2 layer.

Computed variables

- speed: wind speed at each point along the trail, in the units of u and v.
- t: integration step, negative upwind of the seed and positive downwind.
- hours (with fixed_length = FALSE) or km (with fixed_length = TRUE): signed transport time or distance from the seed.
- progress: position along the trail from its upwind end (0) to its downwind end (1).

Examples

```
library(ggplot2)
katrina <- windscape_example("wind_field")

# trails over wind speed, with length proportional to speed
ggplot(katrina, aes(x, y)) +
  geom_raster(aes(fill = speed)) +
  geom_wind_trail(color = "white") +
  coord_quickmap()

# streamlines: equal-length trails showing direction only
ggplot(katrina, aes(x, y)) +
  geom_wind_trail(fixed_length = TRUE) +
  coord_quickmap()

# 6 hours of transport from two sites
sites <- cbind(c(-92, -86), c(24, 30))
ggplot(katrina, aes(x, y)) +
  geom_raster(aes(fill = speed)) +
  geom_wind_trail(seeds = sites, hours = 6, steps = 50, color = "white") +
  coord_quickmap()

# color trails by a computed variable: transport time from the seed
ggplot(katrina, aes(x, y)) +
  geom_wind_trail(aes(color = after_stat(hours)), seeds = sites, hours = 6, steps = 50) +
  scale_color_gradient2() +
  coord_quickmap()
```

---

iter_length *Iteration step length of a random walk*

Description

Iteration step length of a random walk

Usage

```
iter_length(x)
```

Arguments

x	The list returned by random_walk(), or one of its elements.

Value

Length of one iteration (time step), in hours if trans = 1 and wind speeds are in m/s.

Examples

```
rose <- windscape_example("wind_rose")
rw <- random_walk(rose, cbind(-100, 40), iter = 10)
iter_length(rw)
```

---

least_cost *Least-cost travel time surface*

Description

Map the least-cost wind travel time (or flow rate, its inverse) between one or more sites and every grid cell across the domain: downwind, from the sites to each cell, or upwind, from each cell to the sites. This function is a wrapper around accCost.

Usage

```
least_cost(rose, sites, direction = "downwind", rate = FALSE, ...)
```

Arguments

rose	A wind_rose(), or a wind_graph built in advance whose direction matches direction; see pairwise_least_cost().
sites	A two-column matrix (or data frame) of point coordinates, or a SpatVector of points.
direction	Either "downwind" (the default), for travel from the sites to each cell, or "upwind", for travel from each cell to the sites.
rate	Whether to return values as "rates" instead of the default "cost distances". Rates are the inverse of cost distances, representing flow rather than travel time.
...	Further arguments passed to wind_graph(), such as wrap. Not allowed when rose is already a wind_graph.

Details

Paths are restricted to the eight neighbor directions, so cost distances are overestimated for routes between neighbor bearings; see pairwise_least_cost for magnitudes. Sites are snapped to the centers of their grid cells.

Value

A single-layer SpatRaster. With `rate = FALSE`, the layer is named `hours` and gives the travel time between each cell and the nearest site (in the given direction), in hours if `trans = 1` in `wind_rose()` and wind speeds are in m/s (otherwise in relative units). With `rate = TRUE`, it is named `rate` and gives the inverse. Sites outside the grid are ignored, with a warning.

Examples

```
rose <- windscape_example("wind_rose")
site <- cbind(-105, 40)
from_site <- least_cost(rose, site)
to_site <- least_cost(rose, site, direction = "upwind")
terra::plot(c(from_site, to_site), main = c("from the site", "to the site"))
```

---

least_cost_paths *Least-cost paths*

Description

Traces the least-cost (fastest) wind paths between sites, as sequences of grid cell centers. The result has the same structure as `wind_trails()` output, so it can be drawn with `geom_wind_path()`. Paths run between `sites` and the points in `to`: downwind, from the sites to the points, or upwind, from the points to the sites. Without `to`, the points are a regular grid across the domain, so the paths show the network of fastest routes downwind (or upwind) of the sites.

Usage

```
least_cost_paths(
  rose,
  sites,
  to = NULL,
  direction = "downwind",
  pairs = c("all", "nearest", "matched"),
  n = 50,
  ...
)
```

Arguments

`rose`	A `wind_rose()`, or a `wind_graph` built in advance whose direction matches `direction`; see `pairwise_least_cost()`.
`sites, to`	Sites, and the points to trace paths to (downwind) or from (upwind): two-column matrices (or data frames) of longitude and latitude, or `SpatVectors` of points. If `to` is `NULL` (the default), the points are about `n` points on a regular grid across the domain, and points that can't be reached are dropped silently.

`direction` Either "downwind" (the default), for paths from sites to to, or "upwind", for paths from to to sites. With explicit to points, the two give the same paths for swapped arguments; direction matters most with the grid of points and with pairs = "nearest".

`pairs` Which pairs to trace: "all" (the default) traces a path between every site and every point in to; "nearest" traces one path for each site, to or from the point in to with the lowest travel time; "matched" pairs sites and to row by row, which requires them to have the same number of rows; it isn't available without to.

`n` Approximate number of grid points, when to is NULL.

`...` Further arguments passed to wind_graph(), such as wrap. Not allowed when rose is already a wind_graph.

Details

Paths are computed with gdistance::shortestPath(). Because wind is directional, the fastest path from A to B generally differs from the fastest path from B to A.

Paths move between neighboring grid cells, in eight directions, so they appear as segments at 45-degree angles, including occasional staircase patterns where the optimal route lies between two neighbor directions. This reflects the model's structure rather than the plot. Paths from one site form a tree: once two paths meet, they share the rest of their route back to the site, so shared segments show the main transport corridors. Because diagonal steps between cell centers can cross without passing through a common cell, two branches of the tree occasionally appear to cross.

Value

A data frame with one row per path vertex, ordered along each path in the direction of travel (from the site to the point for downwind paths, and from the point to the site for upwind paths):

- trail: line id, for drawing. Each path is one trail, unless it crosses the east-west seam of a wrapped global grid (see wrap in wind_graph()), where a new trail starts so that lines don't cross the map.
- site, to: row numbers of the path's site in sites and point in to, which together identify the path.
- step: vertex number along the path, starting at 0 at its upwind end.
- hours: cumulative travel time along the path, in hours (if trans = 1 in wind_rose() and wind speeds are in m/s). Its final value on each path equals the pair's travel time from pairwise_least_cost() (with snap = TRUE).
- x, y: longitude and latitude of the grid cell center.

Pairs with no path (e.g. because wind never blows from one toward the other), and sites and points outside the grid, are omitted with a warning. Pairs whose site and point fall in the same grid cell are omitted, since their path has no length.

Examples

```
rose <- windscape_example("wind_rose")
site <- cbind(-105, 40)
destinations <- cbind(c(-95, -115, -100, -110), c(45, 35, 32, 48))
paths <- least_cost_paths(rose, site, destinations)
head(paths)

library(ggplot2)
ggplot(paths, aes(x, y)) +
  geom_wind_path(aes(color = hours)) +
  coord_quickmap()

# the network of fastest routes from the site, and to it
down <- least_cost_paths(rose, site, n = 200)
up <- least_cost_paths(rose, site, n = 200, direction = "upwind")
ggplot(rbind(cbind(down, direction = "downwind"), cbind(up, direction = "upwind")),
       aes(x, y)) +
  geom_wind_path(aes(color = hours), arrow = NULL) +
  facet_wrap(~direction) +
  coord_quickmap()
```

---

mantel_test *Mantel test*

Description

This version of the Mantel test supports asymmetric matrices and multiple partial-Mantel controls, a combination of features that is not possible using other widely available Mantel functions such as mantel.partial.

Usage

```
mantel_test(
  x,
  y,
  z = NULL,
  nperm = 999,
  method = "pearson",
  alternative = c("two.sided", "less", "greater")
)
```

Arguments

x, y	Square matrices (can be asymmetric).
z	Optional list of one or more square control matrices if partial Mantel is desired.
nperm	Number of random permutations to perform (positive integer).
method	Correlation method (see cor for details).
alternative	A character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less".

Value

A list with the following components:

- `stat`: The correlation coefficient between `x` and `y`, measured on the data provided. This will be a partial correlation if `z` is specified.
- `quantile`: The quantile of the observed stat in the null distribution.
- `p.value`: Significance, derived from the combination of `quantile` and `alternative`.
- `perm`: A vector of null correlations based on permuted data.

Examples

```
# A convenience function to generate symmetric matrices:
sym_matrix <- function(x){
   n <- which(cumsum(1:100) == length(x)) + 1
   m <- matrix(0, n, n)
   m[lower.tri(m)] <- x
   m[upper.tri(m)] <- t(m)[upper.tri(t(m))]
   m
}

# Simulate some example data:
g <- runif(45)
w <- runif(45)
d <- runif(45)
e <- sample(0:1, 45, TRUE)
G <- sym_matrix(g) # e.g. a genetic matrix
W <- sym_matrix(w) # e.g. a wind matrix
D <- sym_matrix(d) # e.g. a distance matrix
E <- sym_matrix(e) # e.g. an environment matrix

# Basic function usage:
mantel_test(G, W)                      # simple first-order Mantel
mantel_test(G, W, list(D, E))          # partial Mantel
mantel_test(G, W, method = "kendall")  # use of alternative test statistic

# Demonstrate that the function matches vegan::mantel.partial output:
# (using only symmetric matrices, and just one control variable, since
# vegan does not work with asymmetric matrices or multiple controls)
if(requireNamespace("vegan", quietly = TRUE)){
  a <- mantel_test(G, W, list(D), nperm = 999)
  b <- vegan::mantel.partial(G, W, D, permutations = 999)
  c(a$stat, b$statistic)
  c(1 - a$quantile, b$signif) # will differ slightly due to randomization
}
```

---

mean,wind_series-method

Mean wind of a wind_series

Description

Averages the u and v components of a `wind_series` over its time steps, giving the mean wind vector in each cell as a `wind_field`. Long series stored in files are processed in blocks, without loading the whole series into memory.

Usage

```
## S4 method for signature 'wind_series'
mean(x, ..., na.rm = FALSE)
```

Arguments

`x`	A `wind_series`.
`...`	Not used.
`na.rm`	Logical: ignore missing values when averaging?

Details

The mean wind vector describes the net drift of the air over the series: where winds blow from many directions, it is short even if winds are strong. It is similar to, but not the same as, the `net_flow()` of a wind rose built from the series. Net flow describes the connectivity model rather than the wind itself: it is shaped by `trans`, and with `trans = 1` it is typically a few percent weaker than the mean wind, because the rose divides each wind between two neighbor directions. Use `mean()` to describe the wind, and `net_flow()` to describe the wind rose used for connectivity modeling.

Value

A `wind_field` of the mean u and v components.

See Also

`subset_series()` to average over selected time steps, such as one season.

Examples

```
series <- windscape_example("wind_series")
prevailing <- mean(series)
```

---

net_flow — *Net flow of a wind rose, as a wind field*

Description

Summarizes each grid cell of a wind rose as a single vector: the net flow, or the vector sum of the flows toward the cell's eight neighbors. This is the direction and rate at which the rose moves material on balance, as seen by the connectivity models, and is the vector form of the `bearing` and `net` statistics computed by `geom_wind_rose()` and `fortify()`. The result is a `wind_field()`, so it can be drawn with `geom_wind_arrow()` or `geom_wind_trail()`, or used anywhere else a wind field is.

Usage

```
net_flow(rose)
```

Arguments

rose	A wind_rose, on a longitude/latitude grid.

Details

Flow toward each neighbor is conductance times the distance to that neighbor, and net flow is the sum of these flows, each pointing toward its neighbor. Flow is used rather than conductance itself because conductance is a rate per grid cell: on a longitude/latitude grid, east and west neighbors are closer together toward the poles, so a vector sum of conductances would be distorted by latitude, while flow is not.

Net flow is also the local drift velocity of a random_walk() on the rose: the rate at which a particle in the cell moves, on balance, before any decay. (The walk's latitude correction adds equal conductance toward east and west neighbors, so it leaves drift unchanged.)

Net flow is not the same as the mean wind vector. wind_rose() allocates each wind observation between the two neighbors whose bearings bracket its direction, and the vector sum of the two parts is shorter than the original wind unless it blows straight toward a neighbor: by up to about 8% for winds halfway between neighbors. With trans = 1, net flow is therefore typically a few percent less than the mean wind speed, in nearly the same direction; where winds from different directions largely cancel, the two can differ more in both speed and direction. With other values of trans, observations are weighted by transformed speed, so net flow emphasizes the winds that matter for conductance (e.g. strong winds, for trans > 1). Where wind direction is variable, net flow can be small even when winds are strong; the total flow (the speed computed by fortify()) describes overall wind strength regardless of direction.

Value

A wind_field whose u and v layers are the eastward and northward components of net flow, in m/s if the rose was built with trans = 1 from wind speeds in m/s, like any other wind field, so that geom_wind_arrow(), geom_wind_trail(), and wind_trails() (including their hours arguments) treat it correctly. For other trans, units are those of the transformed speeds. The net statistic computed by fortify() for a wind rose is the same quantity in km/h. Cells that are NA in rose are NA in the result.

See Also

geom_wind_rose() for drawing the full rose, and the flux output of random_walk() for the net transport of dispersing material.

Examples

```
rose <- windscape_example("wind_rose")
f <- net_flow(rose)
f

library(ggplot2)
```

```
ggplot(f, aes(x, y)) +
  geom_wind_arrow() +
  coord_quickmap()
```

---

pairwise_least_cost *Pairwise least-cost travel times among sites*

Description

Calculate pairwise wind cost-distances (e.g. travel times) or flow rates (the inverse of cost-distances) among a set of sites, using the least cost path algorithm. For the random walk counterpart, see pairwise_random_walk().

Usage

```
pairwise_least_cost(rose, sites, snap = FALSE, rate = FALSE, ...)
```

Arguments

rose	A wind_rose(). The least-cost functions build a wind_graph() from it internally. Alternatively, a wind_graph built in advance, to save rebuilding it when making many calls on a large grid; its direction must match the analysis (always "downwind" here).
sites	A two-column matrix (or data frame) of point coordinates, or a SpatVector of points.
snap	Logical: snap sites to the centers of their grid cells? Default FALSE, which places sites at their actual locations within cells; see details.
rate	Whether to return values as "rates" instead of the default "cost distances". Rates are the inverse of cost distances, representing flow rather than travel time.
...	Further arguments passed to wind_graph(), such as wrap. Not allowed when rose is already a wind_graph.

Details

Least-cost travel on a wind graph moves between neighboring cell centers, so a grid alone places each site at the center of its cell. For sites a few cells apart or less, that distorts both the distance and the direction between them, and sites in the same cell would be zero hours apart. By default (snap = FALSE), sites are instead added to the graph at their actual locations. Each site is linked to the centers of its own and the eight surrounding cells, and directly to any other site in those cells, by edges whose travel time is computed exactly for the wind where the edge starts, treated as uniform along the edge. That wind is the cell's own at a cell center, and is interpolated between cell centers at a site. Paths then run from site to site through these edges and the grid. Results change continuously as sites move, rather than jumping at cell boundaries. With snap = TRUE, sites are snapped to cell centers, as in gdistance::costDistance().

The exact travel time through uniform wind is the continuum limit of least-cost travel on an eight-neighbor grid: the time to cover a displacement using the cheapest combination of the cell's eight

flow vectors (conductance times the displacement to each neighbor; see net_flow()), which uses at most two of them. Equivalently, the region reachable in one hour is the convex hull of the flow vectors. Site edges therefore share the grid's metric: for sites at cell centers, results are close to those from snap = TRUE, and slightly lower where a site edge combines two directions more efficiently than the grid can.

Paths are restricted to the eight neighbor directions, so cost distances are overestimated for routes between neighbor bearings. For uniform wind on square cells the maximum error is about 8 percent (routes 22.5 degrees off a grid axis). On a longitude/latitude grid, cells narrow east-west toward the poles and neighbor bearings become uneven, so the maximum error grows with latitude, to roughly 18 percent at 60 degrees. Site edges have the same directional bias, so that it is consistent across distances.

Unlike least_cost() and least_cost_paths(), this function has no direction argument: the matrix holds travel in both directions between every pair of sites. Row i gives travel downwind from site i, and column j gives travel upwind of site j.

Value

A square matrix with one row and column per site: the least-cost travel time (or, with rate = TRUE, its inverse) from the row's site to the column's site, in hours if trans = 1 in wind_rose() and wind speeds are in m/s. Small travel times mean strong connectivity. Because wind is directional, the matrix is generally asymmetric. The diagonal is zero. Sites outside the grid get NA, with a warning.

Examples

```
rose <- windscape_example("wind_rose")
sites <- cbind(lon = c(-110, -105, -100, -95), lat = c(40, 42, 38, 44))
pairwise_least_cost(rose, sites) # travel time from row site to column site, in hours
```

---

pairwise_means *Convert asymmetric pairwise matrix to symmetrical pairwise means*

Description

This function produces a symmetrical matrix with values representing the pairwise means of directional flows in an asymmetric input matrix. The output can be used in tests about the **strength** of direction-agnostic connectivity among nodes in a data set (e.g. the "isolation" hypothesis; Kling and Ackerly (2021)).

Usage

```
pairwise_means(x)
```

Arguments

x	A square matrix with entries representing directional edge weights between pairs of nodes.

Value

A symmetrical square matrix the same size as x, with values representing the pairwise means of directional flows in x.

References

Kling and Ackerly (2021). Global wind patterns shape genetic differentiation, asymmetric gene flow, and genetic diversity in trees. Proceedings of the National Academy of Sciences 118: e2017317118. doi:10.1073/pnas.2017317118

Examples

```
# directional flows among three sites, from row to column
x <- matrix(c(0, 2, 6,
              4, 0, 1,
              2, 3, 0), 3, byrow = TRUE)
pairwise_means(x)
```

---

pairwise_random_walk *Pairwise random walk connectivity among sites*

Description

For every pair of sites, computes how strongly particles released at one site reach the other under a stream-mode random walk (see random_walk()): by default, the probability that a particle released at the first site is deposited at the second. This is the random walk counterpart to pairwise_least_cost().

Usage

```
pairwise_random_walk(
  rose,
  sites,
  half_life = NULL,
  value = c("deposition", "residence"),
  density = TRUE,
  timescale = 1,
  latitude_correction = TRUE,
  wrap = NULL,
  chunk = 200
)
```

Arguments

rose	A wind_rose.
sites	A two-column matrix (or data frame) of site coordinates, or a SpatVector of points.

`half_life` Half-life of airborne mass, in hours of transport time; see `random_walk()`. Required for `value = "deposition"`, which is zero without deposition.

`value` `"deposition"` (the default) or `"residence"`; see Value.

`density` Logical: divide by the area of the destination site's grid cell, giving values per km^2? Default `TRUE`. Without this, values depend on cell size: a larger cell receives more of the particles passing nearby. On a longitude/latitude grid, cell area shrinks toward the poles (by about 25 percent from 30 to 50 degrees latitude), so raw values are biased toward lower-latitude destinations even within a single analysis. Dividing by area removes that bias, and also makes values comparable across grid resolutions (though see Details).

`timescale`, `latitude_correction`, `wrap`
See `random_walk()`. Results do not depend on `timescale`.

`chunk` Number of sites to solve for at once. Larger values are faster but use more memory.

Details

Each origin's row is the destination values of a stream-mode random walk released from that origin, so it equals the result of `random_walk(rose, origin, mode = "stream", ...)` read at each destination. All origins share one matrix factorization, so the cost grows slowly with the number of sites.

`density = TRUE` removes the direct effect of cell size on how much a destination receives, but the random walk's spread itself depends somewhat on grid resolution (see `random_walk()`), so values are not fully resolution-independent.

Value

A square matrix with one row and column per site, giving connectivity from the row's site (the origin) to the column's site (the destination). Large values mean strong connectivity, the opposite orientation from the travel times of `pairwise_least_cost()`. With `value = "deposition"`, entries are the probability that a particle released at the origin is deposited in the destination's grid cell (per km^2 if `density = TRUE`). With `value = "residence"`, they are the time, in hours, that a unit release at the origin spends airborne over the destination's cell (per km^2 if `density = TRUE`). Because wind is directional, the matrix is generally asymmetric. The diagonal is each site's self-connectivity: release that is deposited (or stays airborne) in its own cell, which can be large; see `rw_self_retention()`. Sites are treated as the centers of their grid cells, so sites in the same cell get identical rows and columns; see `check_cell_distance()`.

Examples

```
rose <- windscape_example("wind_rose")
sites <- cbind(c(-110, -105, -100, -95), c(40, 42, 38, 44))
pairwise_random_walk(rose, sites, half_life = 24)
```

`pairwise_ratios` *Convert asymmetric pairwise matrix to reciprocally symmetrical matrix*

Description

This function produces a "pairwise ratio matrix" that can be used in tests about asymmetric flows among nodes in a data set (e.g. the "asymmetry" and "diversity" hypotheses; Kling and Ackerly (2021)). The input data set can be an asymmetric matrix with entries representing directed edge weights (e.g. directional flows of wind or genes connecting each pair of populations or "nodes"), or it can be a vector of node attributes (e.g. properties of a site or population, such as genetic diversity). The result is a matrix that represents each pair's relative asymmetry, removing information about their absolute magnitudes.

Usage

```
pairwise_ratios(x, log = TRUE)
```

Arguments

`x`	Either a square matrix with entries representing directional edge weights between pairs of nodes, or a numeric vector with a value for each node.
`log`	Should ratios be log-transformed? Default is `TRUE`, since this generates a symmetrical frequency distribution desirable for most analyses.

Value

A square matrix the same size as `x`, with ones in the diagonal (zeros if `log = TRUE`) and reciprocal values in the upper and lower triangles. The diagonal is set regardless of the diagonal of `x`, which for a matrix of flows is typically zero. A pair with zero flow in one direction gives a ratio of 0 or `Inf` (`-Inf` or `Inf` if logged), and a pair with no flow in either direction gives `NaN`. If `x` is a matrix, the output represents the ratios of above-diagonal to below-diagonal values in `x`. If `x` is a vector, the output represents the ratios of every pairwise combination of `x` values; e.g. a matrix in which the value of cell `[2, 4]` equals `x[2] / x[4]`.

References

Kling and Ackerly (2021). Global wind patterns shape genetic differentiation, asymmetric gene flow, and genetic diversity in trees. Proceedings of the National Academy of Sciences 118: e2017317118. doi:10.1073/pnas.2017317118

Examples

```
# directional flows among three sites, from row to column
x <- matrix(c(0, 2, 6,
              4, 0, 1,
              2, 3, 0), 3, byrow = TRUE)
pairwise_ratios(x)
```

```
# ratios of a site attribute, such as genetic diversity
pairwise_ratios(c(0.2, 0.4, 0.8))
```

`point_distance` *Pairwise distances between points*

Description

A wrapper around distm, returning distances in km.

Usage

```
point_distance(ll)
```

Arguments

`ll`	two-column matrix of point coordinates

Value

Pairwise distances between the points, in km

Examples

```
sites <- cbind(c(-110, -100, -95), c(35, 40, 45))
point_distance(sites)
```

`random_walk` *Simulate wind dispersal by random walk*

Description

Estimates the dispersal of particles across a windscape with a Markov random walk on the grid. Each iteration, the "particle mass" in each grid cell is dispersed within its 9-cell neighborhood in proportion to wind conductance. Depending on the application, this mass could represent probability, numbers of seeds or spores, etc. Particles can be released once (`mode = "pulse"`) or continuously (`mode = "stream"`), and can be deposited along the way (`half_life`).

Usage

```
random_walk(
  rose,
  init,
  mode = c("pulse", "stream"),
  direction = c("downwind", "upwind"),
  half_life = Inf,
  timescale = 1,
  latitude_correction = TRUE,
  wrap = NULL,
  density = TRUE,
  iter = 100,
  record = iter,
  method = c("auto", "solve", "iterate"),
  tol = 1e-08,
  max_iter = 1e+05,
  flux = FALSE,
  source = NULL,
  progress = FALSE
)
```

Arguments

`rose` A `wind_rose`.

`init` Where particles are released: either a two-column matrix of coordinates, which places a mass of 1 in each cell containing a coordinate, or a single-layer SpatRaster of non-negative mass values with the same geometry as `rose`. In stream mode, `init` can be read either as a release rate (mass per hour) or as a one-time release; see Value. With `direction = "upwind"`, `init` instead gives the receptor locations (and weights) whose sources are traced.

`mode` Release mode. `"pulse"` (the default) releases `init` once and tracks the drifting, spreading cloud through time. `"stream"` releases particles continuously and returns the steady state, where release is balanced by deposition and loss across domain edges; equivalently, this is the all-time total for a single release (see Value). It is computed directly rather than by simulating forward.

`direction` `"downwind"` (the default) simulates where particles released at `init` go: their outbound windshed. `"upwind"` simulates where particles reaching `init` come from: its inbound windshed, or catchment. See Value and Details.

`half_life` Half-life of airborne mass, in hours of transport time (assuming `trans = 1` and wind speeds in m/s; see `wind_rose()`). Particles are deposited at the continuous rate `k = log(2) / half_life`, in the cell where they are airborne. Default `Inf`: no deposition.

`timescale` A value between 0 and 1 that scales the time step length (see `rw_max_step()`). The default of 1 uses the longest possible step, advancing the simulation farthest per iteration. Smaller values give more accurate pulse-mode dynamics or set the step to a desired duration. Stream-mode results do not depend on `timescale`.

latitude_correction
: Logical: correct the compression of east-west spread on longitude/latitude grids, where cells narrow toward the poles? Default TRUE. This adds conductance to east and west edges without changing drift, which shortens the time step (pulse mode needs about 1.3 times as many iterations at 45 degrees, 1.9 at 60). Has no effect on projected rasters. See Details.

wrap
: Logical: join the east and west edges of the grid, so that mass leaving one re-enters on the opposite side, and only the north and south edges absorb mass? The default, NULL, does so if rose is a global grid spanning all 360 degrees of longitude, where -180 and 180 are the same meridian, and not otherwise. TRUE or FALSE overrides this; TRUE on a longitude/latitude grid that isn't global gives a warning. As for wrap in wind_graph(), for the least-cost model.

density
: Logical: express results per km^2 rather than per grid cell? Default TRUE, matching pairwise_random_walk(). On a longitude/latitude grid, cell area shrinks toward the poles, so per-cell values are biased toward lower latitudes; per-km^2 values remove that bias and are comparable across grid resolutions. Downwind, each cell's values are divided by its own area. Upwind, values are per particle released from each origin, so they are instead divided by the area of the receptor (each receptor's own, if several); with a single receptor this rescales the whole map by a constant. flux is divided by each cell's area in either direction. Use density = FALSE for per-cell masses, which sum to totals (e.g. in the pulse-mode mass balance); equivalently, multiply per-km^2 results by terra::cellSize().

iter
: Pulse mode only: number of iterations (time steps) to simulate.

record
: Pulse mode only: integer vector of iterations to return, between 0 (the initial state) and iter. Default is the final iteration only.

method
: Stream mode only: how to compute the steady state. "solve" is exact, using a sparse LU solve. "iterate" repeats the stream update rule until the estimated error is below tol; it uses less memory for very large grids. "auto" (the default) uses "solve" for grids of up to 5e5 cells and "iterate" otherwise.

tol
: Stream mode with method = "iterate" only: convergence tolerance, as estimated L1 error relative to total airborne mass. For finite half_life the error estimate is a rigorous bound; for half_life = Inf it is extrapolated from the observed convergence rate.

max_iter
: Stream mode with method = "iterate" only: maximum number of iterations, after which a warning is given.

flux
: Stream mode only: logical: also return the net flux of material, the direction and rate at which it moves through each cell? Default FALSE. For upwind walks, requires a finite half_life. See Value.

source
: Upwind stream mode only: where material is released, used for origin and flux: either a two-column matrix of coordinates (a unit release in each cell containing a coordinate) or a single-layer SpatRaster of non-negative release per grid cell, like init. Default NULL releases uniformly per km^2 (each cell in proportion to its area). To use a map of release per km^2, multiply it by terra::cellSize().

progress
: Pulse mode only: logical: show a progress bar while iterating? Default FALSE.

Details

Transition probabilities:

The wind rose is converted to a simplex of nine probabilities for each grid cell, giving the rates at which particles remain in the cell or move to each of its eight neighbors. Probabilities of moving to a neighbor are proportional to conductance, and retention probabilities are scaled so that the cell with the highest total conductance has zero retention. This keeps conductance proportional across cells while maximizing the dispersal occurring at each iteration.

This is uniformization of the continuous-time Markov chain defined by the conductances, which are rates (in units of 1 / hours, if `trans = 1` and wind speeds are in m/s). With time step `t` and rate matrix `Q`, the transition matrix is `P = I + tQ`. Pulse iterations are therefore a first-order approximation of the continuous-time process at times `t`, `2t`, ...; reducing `timescale` improves their accuracy.

Deposition and the two release modes:

Each time step, airborne mass disperses and a fraction `lambda = kt / (1 + kt)` of it is deposited, where `k = log(2) / half_life`. This is the uniformization of transport and deposition together, so stream-mode results are exact solutions of the continuous-time process and independent of `timescale`. In pulse mode, mass halves after approximately `half_life` hours (exactly in the limit of small `timescale`).

The pulse update rule is `n <- (1 - lambda) * disperse(n)`, with `lambda * n` deposited in place each step before dispersal. The stream update rule is `n <- n0 + (1 - lambda) * disperse(n)`, with fixed point `n* = (I - (1 - lambda) P')^-1 n0`; residence is `(1 - lambda) * t * n*`. The two modes are linked exactly: stream-mode residence equals `(1 - lambda) * t` times the sum of the pulse-mode surfaces over all time steps, starting from step 0.

Upwind walks:

Write the downwind stream solution as `n* = G n0`, where entry `G[r, s]` is the airborne mass at cell `r` per unit released at cell `s`. A downwind walk from a source `s` gives column `s` of `G`: everywhere that source's particles go. An upwind walk to a receptor `r` gives row `r`: every source's contribution to that receptor. It is computed with the adjoint process, using the transpose of the downwind transition matrix (`P` in place of `P'`), not by reversing the wind, so for any source `s` and receptor `r`, the upwind value at `s` equals the downwind value at `r`. The same holds for each pulse iteration.

Domain edges:

Domain edges are absorbing: mass that disperses off the grid, or into NA cells, is lost and never returns. Values near edges are therefore biased low, because they receive no inflow from beyond the edge. On a global grid, the east and west edges are joined by default (see `wrap`), leaving only the north and south edges absorbing. In stream mode with finite `half_life`, the fraction of released mass lost across edges is reported as a message; the domain should be buffered by several decay lengths beyond the area of interest, until this fraction is small or the bias in the area of interest is acceptable. With `half_life = Inf`, edges are the only place stream-mode mass can leave, so the result measures connectivity within the chosen domain and depends on its extent; every valid cell must have a path to an edge or NA cell, or an error is raised. `rw_exit_prob()` maps the probability that mass released at each cell leaves the domain, which bounds the effect of the domain boundary on results for particles released there; the reported edge-loss fraction is this probability averaged over sources, weighted by release.

Latitude correction:

Conductances account for latitude (see `wind_rose()`), so the speed at which mass drifts is correct at all latitudes for winds aligned with a neighbor direction. However, the spread of mass in a nearest-neighbor walk is numerical diffusion whose magnitude scales with hop length. Because longitude/latitude cells narrow east-west toward the poles, east-west hops are shorter than north-south hops, and without correction east-west spread is compressed: for isotropic wind, to roughly 0.86, 0.70, and 0.49 times north-south spread at 30, 45, and 60 degrees latitude.

The correction scales each cell's east-west spread rate up to what the same rose would produce on square cells of the same north-south size, by adding equal conductance to its east and west edges. Because these two neighbors are exactly opposite, drift is unchanged. In tests against the same winds on square cells, the correction brings east-west spread to within about 8 percent of the square-cell value across a range of wind regimes at 45 and 60 degrees. It does not correct the orientation (east-west/north-south covariance) of spread for oblique winds, which remains underestimated at high latitude, nor the small drift bias for winds blowing between neighbor directions (up to about 8 percent at the equator, more at high latitude). The added conductance shortens the time step by an amount set by the highest-latitude cells in the domain. Spread remains dependent on grid resolution with or without the correction.

Value

A named list of two `wind_walk` rasters: `airborne` and `deposition` in pulse mode, or `residence` and `deposition` in stream mode. Masses are in the units of `init`, per km^2 by default or per grid cell with `density = FALSE` (see `density`), and hours assume `trans = 1` and wind speeds in m/s. Cells that are NA in `rose` are NA in the output.

Pulse mode. Each element has one layer per iteration in `record` (named `iter0`, `iter1`, ...):

- `airborne`: the mass airborne over each cell at that iteration.
- `deposition`: the cumulative mass deposited in each cell up to that iteration. Zero everywhere if `half_life = Inf`.

At every iteration, airborne mass plus deposition plus mass lost across domain edges equals the released mass (summing per-cell values, i.e. with `density = FALSE`). Use `iter_length()` to convert iterations to hours.

Stream mode. Each element is a single layer, and neither depends on `timescale`. The two differ in units by a factor of time:

- `residence`: airborne mass integrated over time, in mass-hours (units of `init` times hours).
- `deposition`: deposited mass, in units of `init`. It equals `k * residence`, where `k = log(2) / half_life` is the deposition rate per hour, so it is zero everywhere if `half_life = Inf`.

Stream results have two equivalent readings. They are the steady state when `init` is released continuously as a rate (mass per hour), in which case `residence` is the steady-state airborne mass and `deposition` the steady-state deposition rate (mass per hour). They are also the all-time totals for a single release of `init`, in which case `residence` is the total mass-hours spent airborne over each cell and `deposition` the total mass eventually deposited; for a unit release from one cell, `deposition` is the probability distribution of where a particle lands (a probability density per km^2 by default, or a probability per cell with `density = FALSE`).

Net flux. With `flux = TRUE`, the list also contains `flux`, a `wind_field()` whose `u` and `v` layers are the eastward and northward components of the net transport of material through each cell: the

flow from the cell to each neighbor minus the flow back, times the distance to that neighbor, halved (each flow is shared between two cells). Units are mass times km per hour (if `init` is a release rate, `trans = 1`, and wind speeds are in m/s); direction and relative magnitude are usually what matter. Draw it with `geom_wind_trail()` with `fixed_length = TRUE`, mapping flux magnitude to line width rather than length, e.g. `geom_wind_trail(aes(linewidth = after_stat(speed)), fixed_length = TRUE)` (a square-root scale, `scale_linewidth(trans = "sqrt")`, helps because flux spans orders of magnitude), or with `geom_wind_arrow()`. To show the routes material takes from the source(s), trace the flux with `random_walk_paths()`. Unlike the gradient of `residence` or `deposition`, which describes the shape of the windshed, net flux shows the transport that produces it: across a plume's flanks, for example, material moves mostly downwind, not sideways down the gradient. Each cell's net outflow equals its release minus its deposition.

For upwind walks, `flux` is the transport of the material that ends up at the receptor: material released per `source` (by default, uniformly, one unit per km^2 per hour) and eventually deposited at the receptor (weighted by `init`). It shows the routes by which the receptor's material arrives. Each cell's net outflow equals its release of such material (its release times its probability of eventual deposition at the receptor) minus the amount deposited in the cell, which is nonzero only at the receptor.

Relationship between modes. For the same `rose`, `init`, `half_life`, and `latitude_correction`, stream results are the totals that a pulse walk accumulates over all time, at any `timescale`. Stream `deposition` equals pulse `deposition` once the pulse has run until negligible mass remains airborne, and stream `residence` equals pulse `airborne` mass integrated over time (exactly, `(1 - lambda) * t` times its sum over iterations 0, 1, 2, ...; see Details). This holds because the walk is linear (particles move independently, so their contributions add) and time-invariant (the same `rose` applies at every time step), which is also why continuous and single releases give the same numbers.

Upwind walks. With `direction = "upwind"`, each cell's values describe particles released from that cell and their contribution to the receptor at `init` (weighted by its values, if a raster):

- Stream `deposition`: the probability that a particle released from the cell is deposited at the receptor. This maps where the receptor's deposited material comes from.
- Stream `residence`: the time, in hours, that a unit release from the cell spends airborne over the receptor.
- Pulse `airborne` at iteration `k`: the probability that a particle released from the cell is airborne over the receptor `k` time steps later; and pulse `deposition`: the probability it has been deposited there by then.
- Stream `origin` (with a finite `half_life`): the probability that a particle deposited at the receptor was released from the cell. Where `deposition` asks "if a particle were released here, would it reach the receptor?", `origin` asks "of the particles reaching the receptor, what share came from here?" By Bayes' rule, `origin` is `deposition` times release (see `source`), normalized to sum to 1 over cells, or with `density = TRUE`, to integrate to 1 per km^2. With the default uniform release per km^2, `origin` with `density = TRUE` is proportional to `deposition`. Shares are among sources inside the domain: material from beyond its edges is not represented. For a receptor spanning several cells, `origin` covers particles landing anywhere in it.

Examples

```
library(ggplot2)
```

```
rose <- windscape_example("wind_rose")
site <- cbind(-105, 40)

# downwind windshed: where particles released at the site are deposited
w <- random_walk(rose, site, mode = "stream", half_life = 48, flux = TRUE)
ggplot(w, aes(x, y)) +
  geom_raster(aes(fill = deposition)) +
  scale_fill_viridis_c(trans = "log10", limits = c(1e-8, NA), oob = scales::squish) +
  coord_quickmap()

# net flux of the dispersing material, with magnitude shown as line width
ggplot(w$flux, aes(x, y)) +
  geom_wind_trail(aes(linewidth = after_stat(speed), alpha = after_stat(speed)),
                  fixed_length = TRUE) +
  scale_linewidth(trans = "sqrt", range = c(0.1, 2.5)) +
  scale_alpha(trans = "sqrt", range = c(0.05, 1)) +
  coord_quickmap()
```

`random_walk_paths` *Paths of material through a random walk*

Description

Traces the routes by which material moves in a random walk model: for a downwind walk, the paths from the source(s) to where material is deposited (or leaves the domain); for an upwind walk, the paths from where material originates to the receptor(s). The result is path data, like the output of `least_cost_paths()`, for drawing with `geom_wind_path()`. Where `least_cost_paths()` gives the single fastest route between places, these paths show the full spread of routes, weighted by how much material uses each.

Usage

```
random_walk_paths(rose, init, to = NULL, n = 50, ...)
```

Arguments

`rose`	A `wind_rose`.
`init`	Source(s) for a downwind walk, or receptor(s) for an upwind walk, as for `random_walk()`: a two-column matrix of point coordinates, or a raster of weights.
`to`	Optional points to trace paths to, as a two-column matrix (or data frame) of longitude and latitude, or a `SpatVector` of points. For a downwind walk, each path runs from a source to one of these points: the mean route by which material deposited there arrives. For an upwind walk, each path runs from one of these points to a receptor. If `NULL` (the default), `n` points are placed in proportion to where material ends up (see details).
`n`	Number of paths, when `to` is `NULL`.

... Further arguments passed to random_walk(), such as direction, half_life, latitude_correction, timescale, or wrap. The walk is always run in stream mode with flux, so mode, flux, density, iter, and record can't be supplied.

Details

Despite the name, these paths involve no randomness. They are not simulated random walks: individual particles in a random walk zigzag unpredictably, and no path here follows one. Instead, each path is a streamline of the walk's net flux of material (see flux in random_walk()), computed deterministically from the stream-mode solution. A streamline is the mean route of the material that travels along it, so the paths show where material goes on average. Running the function twice gives identical results.

How paths are placed. For a downwind walk, n end points are placed in proportion to where material ends up: deposition in each cell, plus material leaving the domain across its edges (with half_life = Inf, there is no deposition, so all paths end at the edges). End points are placed deterministically, by dividing the total into n equal shares and putting one end point at the middle of each share; several end points falling in one cell are spread across it on a regular grid. Each path is then traced backward along the flux until it reaches a source. Because each end point represents an equal share of the material, each path carries an equal share too: the density of paths shows where material goes, and each path's end shows where its share lands. For an upwind walk, start points are placed in proportion to the origin of material deposited at the receptor(s), and paths are traced forward along the flux to a receptor.

Paths to chosen points. With to, paths are traced from the given points instead, like least_cost_paths() to chosen destinations. Each path is still the mean route of the material moving through its end point, but the paths no longer carry equal shares of material, so their density has no meaning. Points the flux doesn't reach (e.g. upwind of the source, with a short half-life) can't be traced, and are dropped with a warning.

Since most material is typically deposited near its source, most sampled paths are short; use a larger n to resolve long-distance routes. With half_life = Inf, paths depend on where the domain edges are, as do all random walk results without deposition.

Value

A data frame with one row per point along each path, ordered from upstream to downstream: trail (line ID, for drawing), path (path ID; with to, the row of to the path was traced from), step (position along the path, from 0 at its upstream end), and x and y (coordinates). Each path is one trail, unless it crosses the east-west seam of a wrapped global grid (see wrap in random_walk()), where a new trail starts so that lines don't cross the map. For downwind walks, each path starts at a source and ends where its material is deposited or leaves the domain; for upwind walks, it starts where its material originates and ends at a receptor.

See Also

random_walk(); least_cost_paths() for fastest routes; geom_wind_path() to draw the paths.

Examples

```
library(ggplot2)
rose <- windscape_example("wind_rose")
```

```
site <- cbind(-105, 40)
p <- random_walk_paths(rose, site, n = 60, half_life = 48)

# paths, with a dot where each path's share of material is deposited
ends <- p[!duplicated(p$path, fromLast = TRUE), ]
ggplot(p, aes(x, y)) +
  geom_wind_path(arrow = NULL, alpha = 0.6) +
  geom_point(data = ends, size = 0.8) +
  coord_quickmap()
```

`rw_exit_prob` *Probability of exiting a random walk's domain*

Description

Computes, for each grid cell, the probability that particles released there leave the domain before being deposited: across its edges, into NA cells, or either. Because mass that leaves the domain is lost (see the Domain edges section of `random_walk()`), this measures how much each cell's results are affected by the choice of domain. See Details for how to use it.

Usage

```
rw_exit_prob(
  rose,
  half_life = Inf,
  exits = c("all", "edges", "na"),
  latitude_correction = TRUE,
  wrap = NULL,
  iter = NULL,
  timescale = 1
)
```

Arguments

`rose`	A `wind_rose`.
`half_life`	Half-life of airborne mass, in hours. Should match the value used for the random walk being assessed; see `random_walk()`. With the default `Inf`, every cell with a path to an exit has exit probability 1 (in the long run, all mass leaves), so the result is informative only with `iter` or for comparing `exits = "edges"` with `"na"`.
`exits`	Which exits to count. `"all"` (the default) counts mass leaving across domain edges or into NA cells, matching the edge-loss fraction reported by `random_walk()`. `"edges"` counts only mass leaving across the outer edges of the grid, and `"na"` only mass entering NA cells, e.g. to separate truncation of the domain from loss to a deliberately masked area. Both kinds of exit remain absorbing in every case: `"edges"` and `"na"` give the probability that mass is lost by that route, and

they sum to "all". When the grid wraps (see wrap), its east and west edges are joined, so only the north and south edges count.

latitude_correction, wrap
: Logical. Should match the values used for the random walk being assessed; see random_walk().

iter
: Number of pulse-mode iterations (time steps) over which to count exits. The default NULL gives the probability of ever exiting, which applies to stream-mode results and does not depend on timescale. An integer gives the probability of exiting within iter steps, which applies to pulse-mode results at that iteration.

timescale
: With iter only: time step scaling factor; see random_walk(). Should match the value used for the pulse-mode walk.

Details

Interpreting exit probabilities as error bounds:

Consider a random walk on a larger domain that contains this one, with any winds outside it. A particle released at cell x follows the same process in both domains until it first exits this one, so results for particles released at x can differ between the two only through particles that exit. This makes the exit probability h(x) a bound on the effect of the domain boundary, with no assumptions about winds outside the domain, for any quantity that is a probability over particles released at x:

- For an upwind walk (direction = "upwind"), stream-mode deposition at x (the probability that a particle released at x is deposited at the receptor) is off by at most h(x). The same holds for pulse-mode airborne and deposition at iteration iter, using h computed with that iter.
- For pairwise_random_walk(), values for source x are off by at most h(x).
- For a downwind walk from a unit source at x, the deposition map is off by at most h(x) in total, summed over all cells (with density = FALSE). This bounds the map as a whole, not the error at any particular cell. For a downwind walk with several sources, the edge-loss fraction reported by random_walk() is h averaged over sources, weighted by release.

Masking cells where h exceeds a tolerance therefore leaves cells whose values are accurate to within that tolerance. The bound is conservative, since it counts every particle that exits as fully misplaced, whereas some would return. It does not cover the error at individual cells of downwind maps, which comes from particles that exit and later return, nor the absence of sources beyond the domain in results that assume release everywhere (such as upwind origin with the default source); bounding these requires assumptions about the winds and sources outside the domain.

Exit probabilities are typically highest near the domain's downwind margins, where released mass is carried out of the domain, and fall off quickly toward upwind margins. The width of the affected band grows with half_life.

Computation:

Exit probabilities solve (I - (1 - lambda) P) h = (1 - lambda) l, where P is the transition matrix, lambda the per-step deposition fraction, and l the per-step probability of stepping out of the domain by the routes in exits. This is one sparse solve, the same size as a stream-mode walk. With iter, the recursion h <- (1 - lambda) * (l + P h) is applied iter times from h = 0.

Value

A single-layer SpatRaster named `exit_prob`, giving for each cell the probability that a particle released there exits by the routes in `exits` (within `iter` steps, if given) before being deposited. NA where `rose` is NA. Cells from which no exit can be reached have probability 0.

Examples

```
rose <- windscape_example("wind_rose")
h <- rw_exit_prob(rose, half_life = 24)
plot(h)

# mask an upwind walk to cells where the domain affects results by less than 1%
rw <- random_walk(rose, cbind(-100, 40), mode = "stream", direction = "upwind",
                  half_life = 24)
plot(terra::mask(rw$deposition, h > 0.01, maskvalues = TRUE))
```

---

rw_max_step — *Maximum iteration duration for a random walk*

Description

Calculate the maximum possible iteration length for a random walk simulation, which is the residence time of the grid cell with the greatest total conductance. Walks can proceed no faster than this.

Usage

```
rw_max_step(rose)
```

Arguments

`rose`	A `wind_rose`.

Value

A number: the residence time of the cell with the greatest total conductance, in hours for a rose built from wind speeds in m/s with `trans = 1`.

Examples

```
rose <- windscape_example("wind_rose")
rw_max_step(rose)
```

rw_self_retention — *Self-contribution of each source cell in a stream-mode random walk*

Description

Computes G_cc, the stream-mode residence time in cell c per unit of release from cell c (see random_walk()). This is each source's contribution to its own value, which can be subtracted for a leave-one-out surface, e.g. to avoid a site predicting itself.

Usage

```
rw_self_retention(
  rose,
  half_life = Inf,
  timescale = 1,
  latitude_correction = TRUE,
  wrap = NULL,
  cells = NULL,
  exact = FALSE,
  chunk = 200
)
```

Arguments

rose	A wind_rose.
half_life	Half-life of airborne mass, in hours. Must match the value used for the stream-mode walk; see random_walk().
timescale	Time step scaling factor; see random_walk(). Affects only the approximate values; exact values are independent of timescale.
latitude_correction, wrap	Logical. Must match the values used for the stream-mode walk; see random_walk().
cells	Cell numbers, or a two-column matrix of coordinates, at which to compute G_cc. Optional for the approximation (which defaults to every cell); required if exact = TRUE.
exact	Logical: compute exact values rather than the approximation? Exact values are a sparse solve per cell (sharing one factorization), practical for a set of occupied cells but not for every cell of a large grid. The approximation is a lower bound. Default FALSE.
chunk	With exact = TRUE: number of cells to solve for at once. Larger values are faster but use more memory.

Details

The approximation G_cc ~ (1 - lambda) * t / (1 - (1 - lambda) * p_cc) counts only paths in which mass never leaves the cell (p_cc is the retention probability; see random_walk()). It omits mass that leaves and returns, which is substantial in a nearest-neighbor walk, especially in fast cells where p_cc is near zero.

Value

G_cc in hours per unit of release: a SpatRaster of approximate values for every cell if cells is NULL, otherwise a numeric vector with one value per cell. For a leave-one-out surface, residence_loo = residence - init * G_cc and deposition_loo = log(2) / half_life * residence_loo, using the residence layer from random_walk() and init values at the source cells.

Examples

```
rose <- windscape_example("wind_rose")
sites <- cbind(c(-110, -100), c(35, 40))
rw_self_retention(rose, half_life = 24, cells = sites, exact = TRUE)
```

---

scale_fill_bearing *Color scales for wind direction*

Description

Cyclic color scales for compass bearings (degrees clockwise from north), using the same hues as the windscape direction color wheel. Bearings outside 0-360 are wrapped, so 0 and 360 (or -90 and 270) get the same color. By default the legend shows the eight compass directions; pass breaks and labels to change this.

Usage

```
scale_fill_bearing(..., chroma = 90, luminance = 65, aesthetics = "fill")

scale_colour_bearing(..., chroma = 90, luminance = 65, aesthetics = "colour")

scale_color_bearing(..., chroma = 90, luminance = 65, aesthetics = "colour")
```

Arguments

...	Other arguments passed to ggplot2::continuous_scale(), such as name or guide.
chroma, luminance	HCL chroma and luminance of the colors. Defaults match the windscape rose plots.
aesthetics	The aesthetics to which the scale applies.

Value

A ggplot2 scale.

Examples

```
library(ggplot2)
d <- data.frame(x = 1:8, bearing = seq(0, 315, 45))
ggplot(d, aes(x, 1, fill = bearing)) + geom_tile() + scale_fill_bearing()
```

subset_series *Select time steps from a wind_series*

Description

Selects time steps from a `wind_series` by month, hour of day, date range, or index, keeping the u and v layers of each step together. Use it to build wind roses for the times that matter for dispersal, such as a flowering season or the hours when propagules are released.

Usage

```
subset_series(
  x,
  months = NULL,
  hours = NULL,
  start = NULL,
  end = NULL,
  steps = NULL
)
```

Arguments

`x`	A `wind_series`, with times in its layer names (see `wind_times()`) if selecting by `months`, `hours`, `start`, or `end`.
`months`	Integer vector of months (1-12) to keep.
`hours`	Integer vector of hours of the day (0-23, UTC) to keep.
`start`, `end`	Keep time steps from `start` to `end`, inclusive. Each can be a `Date`, a POSIXct time, or a character string like `"2000-06-01"` or `"2000-06-01 12:00:00"` (interpreted as UTC). A date without a time includes that whole day, so `end = "2000-06-30"` keeps all of June 30.
`steps`	Time steps to keep, as integer indices or a logical vector with one element per time step, for selections the other arguments don't cover. Can be combined with them, e.g. `steps = speed > 10`, given a vector `speed`.

Details

Criteria are combined: a time step is kept only if it meets all of them. Times are in UTC. For hours of the day in local solar time, note that local time is about UTC plus longitude / 15 hours (e.g. UTC - 7 hours at 105 degrees west), so `hours` for a large region selects different local times in different places.

Value

A `wind_series` containing the selected time steps, in their original order.

See Also

wind_times()

Examples

```
series <- windscape_example("wind_series")

# summer only
summer <- subset_series(series, months = 6:8)
range(wind_times(summer))

# afternoons in the first half of the year
subset_series(series, hours = 18:23, end = "2000-06-30")
```

---

weight_conductance *Weight a wind rose's conductances*

Description

This function adjusts the conductance values in a wind rose, multiplying them by the values of a secondary raster data set w, which can be used to integrate non-wind factors into a wind connectivity analysis.

Usage

```
weight_conductance(rose, w)
```

Arguments

rose	A wind_rose.
w	A single-layer SpatRaster with values to be multiplied by rose, on the same grid as rose.

Details

As a hypothetical example, to incorporate a decreased but nonzero likelihood of dispersal over inhospitable areas, w could be a raster layer with 0.1 indicating water or mountains and 1.0 elsewhere. This would have the effect of down-weighting conductance over water or mountains by 90%.

Value

A wind_rose, with conductance values weighted by w.

Examples

```
rose <- windscape_example("wind_rose")

# halve conductance west of 105 degrees W
w <- terra::ifel(terra::init(rose[[1]], "x") < -105, 0.5, 1)
weighted <- weight_conductance(rose, w)
```

---

wind_field *Create a wind_field*

Description

Creates a `wind_field`, the wind across a grid at a single moment, from a two-layer raster or file, or from one time step of a `wind_series`. To summarize a whole series as a single field, use mean() or `net_flow()`.

Usage

```
wind_field(x, step = NULL)
```

Arguments

`x`	A `SpatRaster` with two layers holding the u and v wind components, the path to a raster file with those two layers, or a `wind_series`. Data must be on a longitude/latitude grid, with components in m/s, oriented to true east and north.
`step`	For a `wind_series` with more than one time step, the time step to use, as an integer index (see `wind_times()` for the time of each step).

Value

A `wind_field` object, which is a particular form of `SpatRaster`.

See Also

mean() for the time-mean wind of a series.

Examples

```
series <- windscape_example("wind_series")
field <- wind_field(series, step = 1)
```

---

wind_field-class *An S4 object class representing a wind field*

Description

An S4 object class representing a wind field

wind_graph *Build a wind connectivity graph*

Description

Constructs the directed network that the least-cost functions (least_cost(), least_cost_paths(), and pairwise_least_cost()) find paths through. Those functions build the graph from a wind rose automatically, so you don't normally need to call this function. Building the graph yourself can save time when making many least-cost calls on a large grid: pass the graph in place of the rose.

Usage

```
wind_graph(x, direction = "downwind", wrap = NULL)
```

Arguments

x	A wind_rose.
direction	Either "downwind" (the default) or "upwind": whether links follow the wind, or run against it.
wrap	Logical: join the east and west edges of the grid, linking cells across them? The default, NULL, does so if x is a global grid spanning all 360 degrees of longitude, where -180 and 180 are the same meridian, and not otherwise. TRUE or FALSE overrides this; TRUE on a longitude/latitude grid that isn't global gives a warning.

Details

The graph links each grid cell to its eight neighbors. Each link's conductance is the mean, over its two cells, of the rose's conductance in the link's direction. In a downwind graph, links point the way the wind carries material; an upwind graph is the same network with every link reversed, for measuring travel toward a site.

Value

A wind_graph: a gdistance Transition-class object, with its direction recorded.

Examples

```
rose <- windscape_example("wind_rose")
graph <- wind_graph(rose)
sites <- cbind(c(-110, -100), c(40, 40))
pairwise_least_cost(graph, sites)
```

wind_graph-class *An S4 object class representing a wind graph*

Description

An S4 object class representing a wind graph

wind_rose *Build or load a wind rose*

Description

A wind rose summarizes a time series of wind fields as the average wind conductance from each grid cell toward each of its eight neighbors. Given a `wind_series`, `wind_rose()` builds one; given a raster or file holding a saved wind rose, it loads it.

Usage

```
wind_rose(
  x,
  trans = 1,
  n_steps = NA_integer_,
  filename = NULL,
  overwrite = FALSE
)
```

Arguments

`x`	A `wind_series`, to build a wind rose from; or a saved wind rose to load, as an 8-layer `SpatRaster` or the path to a raster file (e.g. one written with `terra::writeRaster()`). A saved rose's layers must hold conductance toward the southwest, west, northwest, north, northeast, east, southeast, and south neighbors, in that order.
`trans`	Either a non-negative number indicating the power to raise wind speeds to, or an elementwise function of wind speed (it may be applied to many cells and time steps at once, so its result for each speed must not depend on the others); see details. When loading a saved rose, the `trans` it was built with, which is recorded with the rose (e.g. for `combine_roses()`); if not given, it is read from the file's metadata where recorded there (as in files from `download_wind_rose()`), and otherwise defaults to 1.
`n_steps`	When loading a saved rose, the number of time steps it summarizes, needed to combine it with other roses using `combine_roses()`; if not given, it is read from the file's metadata where recorded there. Ignored when building a rose, which records its number of time steps automatically.

`filename`	When building a rose, an optional file path to write the result to, as a raster file (e.g. a GeoTIFF). The file records the rose's number of time steps, and `trans` if it is a number, so `wind_rose(filename)` reloads it with no other arguments.
`overwrite`	Logical. Whether to overwrite an existing `filename`.

Details

For each time step, the wind in each cell is divided between the two neighbors whose directions bracket the wind direction, in proportion to how closely the wind points toward each, and its transformed speed (see `trans`) is converted to conductance toward each neighbor; conductance is then averaged over all time steps. Long series are processed in chunks of time steps, so a series spanning many files (see `wind_series()`) can be summarized without loading it all into memory; the result is identical to processing it at once. To build a rose from selected time steps, such as a season or time of day, select them first with `subset_series()`. To build many roses (e.g. one per month), run separate `wind_rose()` calls in parallel processes, e.g. with `parallel::mclapply()`.

The `trans` parameter defines the transformation function used to convert wind speed into conductance. If a numeric value is supplied, the function speed^trans is used. A value of trans = 0 will ignore speed, assigning weights based on direction only; trans = 1 assumes conductance is proportional to windspeed, trans = 2 assumes it's proportional to aerodynamic drag, and trans = 3 assumes it's proportional to force. Any intermediate value can also be used. Any elementwise function, transforming each speed independently of the others, can also be supplied; for example, to model seed dispersal for a species that only releases seeds when winds exceed 10 m/s, we could specify a threshold function `trans = function(x){x[x < 10] <- 0; return(x)}`. Calm time steps (zero wind speed) have no direction, so they contribute no conductance whatever `trans` is, though they still count toward the average; with `trans = 0`, a rose's total conductance thus reflects the fraction of time steps that are not calm.

Grid geometry: windscape works on longitude/latitude grids with square cells. Distances and bearings to each cell's neighbors are computed on the ellipsoid at that cell's latitude, so conductance accounts for the narrowing of cells toward the poles. Projected grids are not currently supported. If source data are projected, reproject them to longitude/latitude before building a `wind_series`, rotating u and v to true east and north if they are defined relative to the projected grid (as in some reanalysis products).

Value

A `wind_rose` object. This is an 8-layer raster stack, where each layer is wind conductance from the focal cell to one of its neighbors (clockwise starting in the SW). If input windspeeds are in m/s and `trans = 1`, values are in (1 / hours). When building a rose, cells missing wind data at any time step are `NA` in all eight layers.

See Also

`combine_roses()` to combine roses built from different time periods.

Examples

```
series <- windscape_example("wind_series")
rose <- wind_rose(series)
```

```
# save and reload; the file records n_steps and trans
f <- tempfile(fileext = ".tif")
rose <- wind_rose(series, filename = f)
rose2 <- wind_rose(f)
```

`wind_rose-class` *An S4 object class representing a wind rose*

Description

An S4 object class representing a wind rose

`wind_rose-grid` *Changing the grid of a wind rose*

Description

A wind rose's conductances are rates of flow between neighboring cells, so they depend on cell size. Changing the grid with `terra::aggregate()`, `terra::disagg()`, `terra::resample()`, or `terra::project()` would average or interpolate conductances without accounting for the new cell size, giving wrong results, so these operations are not allowed on a `wind_rose`. To refine a rose's grid, use `downscale()`. To use a coarser or different grid, change the grid of the `wind_series` (e.g. with `terra::aggregate()`) and build a new rose from it.

Usage

```
## S4 method for signature 'wind_rose'
aggregate(x, ...)

## S4 method for signature 'wind_rose'
disagg(x, ...)

## S4 method for signature 'wind_rose,ANY'
resample(x, y, ...)

## S4 method for signature 'wind_rose'
project(x, ...)
```

Arguments

`x`	A `wind_rose`.
`y, ...`	Not used.

Value

These methods signal an error.

wind_rose_cache *Manage the wind rose download cache*

Description

Lists, or deletes, the pre-built wind rose files that download_wind_rose() has saved to the cache. The cache is in the user cache directory given by tools::R_user_dir("windscape", "cache"); set options(windscape.cache_dir = ...) to use a different location.

Usage

```
wind_rose_cache(clear = FALSE)
```

Arguments

clear	Logical. If TRUE, delete all cached wind rose files.

Value

A data frame of cached files, with columns file, release, bytes, and path (invisibly, listing the deleted files, if clear = TRUE).

See Also

download_wind_rose()

Examples

```
wind_rose_cache()
```

wind_rose_catalog *Catalog of pre-built wind roses*

Description

Lists the pre-built global wind roses available for download with download_wind_rose(). Currently these are built from 10 m CFSR wind for 1979-2010, at CFSR's native resolution (about 0.31 degrees), with trans = 1.

Usage

```
wind_rose_catalog()
```

Value

A data frame with one row per file, and columns:

- `source`, `level`: wind data set and height (as in `download_wind_data()`).
- `tier`: the period the file covers: `"month"` (a single year-month), `"year"` (a calendar year), `"month_of_year"` (a calendar month across all years), or `"all"` (the full period).
- `year`, `month`: the year and month the file covers, or `NA` where it spans several.
- `first`, `last`: the first and last year-months covered.
- `n_steps`: the number of hourly time steps summarized.
- `trans`: the wind speed transformation the rose was built with (see `wind_rose()`).
- `file`, `bytes`, `md5`: file name, size in bytes, and MD5 checksum.
- `release`, `url`: the release the file belongs to, and its download URL.

See Also

`download_wind_rose()`

Examples

```
catalog <- wind_rose_catalog()
table(catalog$tier)
```

---

wind_series *Create a wind_series*

Description

Creates a `wind_series`, a time series of wind fields, from a raster, a list of rasters, or one or more raster files, such as the monthly files saved by `download_wind_data()`. Multiple inputs are combined into one series, in the order given, without reading the data into memory until needed.

Usage

```
wind_series(x, order = c("uuvv", "uvuv"))
```

Arguments

`x`	A multi-layer `SpatRaster` with layers containing u and v wind components; a list of them; or a character vector of paths to raster files. Each input must contain an equal number of u and v layers, arranged as given by `order`, and all inputs must share the same grid. Data must be on a longitude/latitude grid with square cells (equal x and y resolution in degrees), with u and v components in m/s, oriented to true east and north; see `wind_rose()`.
`order`	Either `"uuvv"`, the default, indicating that each input has all u components followed by all v components, or `"uvuv"`, indicating alternating u and v components.

Value

A `wind_series` object, a `SpatRaster` with all the u layers of all inputs, followed by all the v layers, in time order.

See Also

`subset_series()` and `wind_times()` for working with time steps.

Examples

```
series <- windscape_example("wind_series")
series

# combine series stored in separate files, such as monthly files from download_wind_data()
files <- c(tempfile(fileext = ".tif"), tempfile(fileext = ".tif"))
terra::writeRaster(subset_series(series, steps = 1:48), files[1])
terra::writeRaster(subset_series(series, steps = 49:96), files[2])
wind_series(files)
```

---

wind_series-class — *An S4 object class representing a wind field time series*

Description

An S4 object class representing a wind field time series

---

wind_times — *Get the time of each step in a wind_series*

Description

Returns the time of each step of a `wind_series`, parsed from its layer names (like `"u 2000-01-01 06:00:00"`, as written by `download_wind_data()`) or, if the names hold no times, from the times set with `terra::time()`.

Usage

```
wind_times(x)
```

Arguments

`x`	A `wind_series`.

Value

A POSIXct vector (UTC), with one element per time step.

See Also

subset_series() to select time steps.

Examples

```
series <- windscape_example("wind_series")
head(wind_times(series))
```

---

wind_trails — *Trace particle trails through a wind field*

Description

Traces the paths of particles carried by a wind field, upwind and/or downwind from a set of starting points. Each step, particles move in the direction of the local wind (interpolated bilinearly), either by the distance the wind carries them in that time (hours), or by a fixed distance (distance), giving trails that show direction only. Trails through a single wind field are streamlines.

Usage

```
wind_trails(
  x,
  seeds,
  hours = NULL,
  distance = NULL,
  steps = 100,
  direction = c("both", "downwind", "upwind"),
  wrap = NULL,
  sf = FALSE,
  ...
)
```

Arguments

x	A wind_field, created with wind_field().
seeds	Starting points: either a two-column matrix of longitude and latitude, or a single integer giving the number of points to generate with generate_particles(), in which case further arguments to that function can be passed in ....
hours	Transport time in hours spanned by each trail, assuming wind speeds in x are in m/s. Particles move at the local wind speed, so trail length is proportional to speed. Supply exactly one of hours or distance.
distance	Length in km of each trail. Particles move at the same speed regardless of wind speed, so trails show only direction. Supply exactly one of hours or distance.
steps	Number of integration steps per trail. More steps give smoother and more accurate trails. Default 100.

direction	Trace trails "both" ways from each seed (the default; half of hours or distance upwind and half downwind), only "downwind", or only "upwind".
wrap	Wrap particles that leave the field across its edges back in on the opposite side: "neither", "horizontal", "vertical", or "both"; TRUE and FALSE are shorthand for "horizontal" and "neither". Trails end where they leave the field across an edge that doesn't wrap. The default, NULL, wraps horizontally if x is a global field spanning all 360 degrees of longitude, where -180 and 180 are the same meridian, and not otherwise.
sf	Logical: return trails as an sf object of linestrings instead of a data frame? Requires the sf package. Default FALSE.
...	Further arguments to generate_particles(), used only if seeds is an integer.

Value

A data frame with one row per particle position, ordered along each trail from upwind to downwind:

- trail: trail id. Each particle's path is one trail, unless it wraps across the field's edge, which starts a new trail so that lines don't cross the map.
- particle: particle id, the row of seeds it started from.
- step: integration step, negative upwind of the seed and positive downwind.
- hours (if hours was given): signed transport time from the seed. Or km (if distance was given): signed distance from the seed.
- x, y: longitude and latitude.
- speed: wind speed at the particle's position, in the units of x.

With sf = TRUE, an sf object with one linestring per trail, with trail and particle columns and the elapsed hours or km as the linestrings' M coordinate.

Examples

```
katrina <- windscape_example("wind_field")
seeds <- cbind(c(-92, -86), c(24, 30))

# where does air at these points come from and go to, over 12 hours?
tr <- wind_trails(katrina, seeds, hours = 12)
head(tr)

library(ggplot2)
ggplot(tr, aes(x, y)) +
  geom_wind_path(aes(color = speed)) +
  coord_quickmap()
```

windscape-layers *Selecting and combining layers of windscape objects*

Description

wind_series, wind_field, and wind_rose objects are SpatRasters whose layers have a fixed structure (u and v layers, or conductance toward eight neighbors). Selecting layers with [[or terra::subset(), or combining objects with c(), generally breaks that structure, so these operations return a plain SpatRaster. To select time steps while keeping a series intact, use subset_series(); to take one time step as a field, use wind_field(); and to combine series, use wind_series() with a list of series or files.

Usage

```
## S4 method for signature 'wind_series,ANY,ANY'
x[[i, j, ...]]

## S4 method for signature 'wind_field,ANY,ANY'
x[[i, j, ...]]

## S4 method for signature 'wind_rose,ANY,ANY'
x[[i, j, ...]]

## S4 method for signature 'wind_series'
subset(x, subset, ...)

## S4 method for signature 'wind_field'
subset(x, subset, ...)

## S4 method for signature 'wind_rose'
subset(x, subset, ...)

## S4 method for signature 'wind_series'
c(x, ...)

## S4 method for signature 'wind_field'
c(x, ...)

## S4 method for signature 'wind_rose'
c(x, ...)
```

Arguments

x	A wind_series, wind_field, or wind_rose.
i, subset	Layers to select, as for a SpatRaster.
j	Not used.

...	Other arguments passed to the `SpatRaster` method, or for `c()`, other objects to combine.

Value

A `SpatRaster`.

Examples

```
series <- windscape_example("wind_series")
class(series[[1:2]]) # a plain SpatRaster
class(subset_series(series, steps = 1:2)) # a wind_series
```

`windscape_example` *Example wind data sets*

Description

Load one of the example data sets shipped with windscape, as a ready-to-use object.

Usage

```
windscape_example(name = c("wind_series", "wind_rose", "wind_field"))
```

Arguments

`name`	Which data set to load: "wind_series", "wind_rose", or "wind_field".

Details

All three are derived from 10 m hourly wind from the Climate Forecast System Reanalysis (CFSR), on CFSR's native grid of approximately 0.32 degrees:

"wind_series" A `wind_series` for the western and central United States (120-90 W, 30-50 N), with 96 time steps: every sixth hour on the 1st and 15th of each month in 2000. Values are rounded to 0.1 m/s. This is for demonstrating the `wind_series` -> `wind_rose` workflow; a real analysis would use a longer and denser time series.

"wind_rose" A `wind_rose` (with `trans = 1`) for the same region, built from all 576 hours on those days. Because it uses six times as many observations as `"wind_series"`, it is a better estimate of the region's wind regime, and is the better choice for demonstrating connectivity analyses.

"wind_field" A `wind_field` for Hurricane Katrina in the Gulf of Mexico, at 2005-08-28 23:00 UTC (99-78 W, 17-35 N), rounded to 0.1 m/s.

Value

An object of the class named by `name`.

Source

Climate Forecast System Reanalysis (Saha et al. 2010), via the NSF NCAR Research Data Archive.

Examples

```
ws <- windscape_example("wind_series")
rose <- windscape_example("wind_rose")
katrina <- windscape_example("wind_field")
```

`ws_summarize` *Summary statistics for a windshed*

Description

A "windshed" is a raster layer representing upwind or downwind connectivity for a given point. This function computes a range of statistics summarizing a windshed, such as the size, centroid location, and isotropy of the wind accessibility surface. It is useful primarily for comparing the windshed properties of multiple different sites.

Usage

```
ws_summarize(x, origin, radius = NULL)
```

Arguments

`x`	SpatRaster representing wind accessibility, with higher values indicating greater accessibility. (For example, this could be the output of `random_walk()`, or `least_cost(..., rate = TRUE)`.)
`origin`	Coordinates of center point (matrix with 2 columns and 1 row).
`radius`	Optional windshed radius, in km; cells farther from the origin than this will be excluded from the summary statistics.

Value

A named vector of summary statistics:

- `centroid_x`: Longitude of windshed "centroid". The centroid is the mean of all lon-lat coordinates, weighed by the wind accessibility values in `x`.
- `centroid_y`: Latitude of windshed centroid.
- `centroid_distance`: Distance from origin to windshed centroid, in km.
- `centroid_bearing`: Bearing from origin to windshed centroid, in degrees.
- `windshed_distance`: Mean distance from origin, in km. This is the average distance from the origin to every grid cell, weighted by accessibility.
- `windshed_bearing`: Mean bearing from origin, in degrees. This is the circular-mean bearing from the origin to every other grid cell, weighted by accessibility.

- `windshed_isotropy`: Circular standard deviation of the bearing from the origin to every other grid cell, weighted by accessibility. Low values indicate that accessible sites are concentrated in a narrow range of compass directions, while high values indicate a wide distribution of directions. The range of possible values is 0-1.
- `windshed_size`: Mean accessibility. This is the average of all accessibility values, weighted to correct for grid distortions. Higher values indicate landscapes with greater overall wind accessibility.
- `windshed_landarea`: A relative measure of the land area covered by grid cells with nonzero wind accessibility.

Index

∗ **datasets**
birch, 3
[[,wind_field,ANY,ANY-method (windscape-layers), 62
[[,wind_rose,ANY,ANY-method (windscape-layers), 62
[[,wind_series,ANY,ANY-method (windscape-layers), 62

accCost, *25*
aggregate,wind_rose-method (wind_rose-grid), 56

birch, 3
build_wind_graph (wind_graph), 53

c,wind_field-method (windscape-layers), 62
c,wind_rose-method (windscape-layers), 62
c,wind_series-method (windscape-layers), 62
cell_distance, 4
check_cell_distance, 4
check_cell_distance(), *11*, *35*
combine_roses, 5
combine_roses(), *10*, *54*, *55*
cor, *28*

disagg,wind_rose-method (wind_rose-grid), 56
distm, *37*
download_land_mask, 6
download_land_mask(), *8*
download_wind_data, 7
download_wind_data(), *6*, *9*, *58*, *59*
download_wind_rose, 9
download_wind_rose(), *8*, *54*, *57*, *58*
downscale, 10
downscale(), *5*, *56*

fortify.random_walk (fortify.windscape), 11
fortify.wind_field (fortify.windscape), 11
fortify.wind_rose (fortify.windscape), 11
fortify.wind_series (fortify.windscape), 11
fortify.windscape, 11, *15*, *19*, *22*

gdistance::costDistance(), *32*
gdistance::shortestPath(), *27*
generate_particles, 13
generate_particles(), *60*, *61*
geoCorrection, *14*
geoCorrection,wind_graph,ANY-method, 14
geom_wind_arrow, 14
geom_wind_arrow(), *23*, *30*, *31*, *42*
geom_wind_path, 17
geom_wind_path(), *21*, *26*, *43*, *44*
geom_wind_rose, 18
geom_wind_rose(), *12*, *30*, *31*
geom_wind_trail, 21
geom_wind_trail(), *17*, *18*, *30*, *31*, *42*
GeomWindArrow (geom_wind_arrow), 14
GeomWindRose (geom_wind_rose), 18
GeomWindTrail (geom_wind_trail), 21
ggplot2::aes(), *15*, *17*, *19*, *22*
ggplot2::continuous_scale(), *49*
ggplot2::coord_quickmap(), *16*, *20*, *23*
ggplot2::coord_sf(), *16*, *20*, *23*
ggplot2::ggplot(), *11*, *15*, *19*
ggplot2::layer(), *15*, *16*, *18–20*, *23*
grid::arrow(), *16*, *18*, *23*

iter_length, 24
iter_length(), *41*

least_cost, 25

least_cost(), *4*, *11*, *33*, *53*
least_cost_paths, 26
least_cost_paths(), *4*, *11*, *17*, *21*, *33*, *43*, *44*, *53*

mantel.partial, *28*
mantel_test, 28
mean(), *52*
mean,wind_series-method, 29

net_flow, 30
net_flow(), *30*, *33*, *52*

pairwise_least_cost, *25*, 32
pairwise_least_cost(), *4*, *11*, *25–27*, *34*, *35*, *53*
pairwise_means, 33
pairwise_random_walk, 34
pairwise_random_walk(), *4*, *32*, *39*, *46*
pairwise_ratios, 36
point_distance, 37
project,wind_rose-method (wind_rose-grid), 56

random_walk, 37
random_walk(), *4*, *11*, *12*, *25*, *31*, *34*, *35*, *43–46*, *48*, *49*
random_walk_paths, 43
random_walk_paths(), *42*
resample,wind_rose,ANY-method (wind_rose-grid), 56
rw_exit_prob, 45
rw_exit_prob(), *40*
rw_max_step, 47
rw_max_step(), *38*
rw_self_retention, 48
rw_self_retention(), *35*

scale_color_bearing (scale_fill_bearing), 49
scale_colour_bearing (scale_fill_bearing), 49
scale_fill_bearing, 49
scale_fill_bearing(), *20*
stat_wind_arrow (geom_wind_arrow), 14
stat_wind_rose (geom_wind_rose), 18
stat_wind_trail (geom_wind_trail), 21
StatWindArrow (geom_wind_arrow), 14
StatWindRose (geom_wind_rose), 18
StatWindTrail (geom_wind_trail), 21
subset,wind_field-method (windscape-layers), 62
subset,wind_rose-method (windscape-layers), 62
subset,wind_series-method (windscape-layers), 62
subset_series, 50
subset_series(), *15*, *23*, *30*, *55*, *59*, *60*, *62*

terra::cellSize(), *39*
terra::disagg(), *10*, *11*
Transition-class, *53*

weight_conductance, 51
wind_field, 52
wind_field(), *30*, *41*, *60*, *62*
wind_field-class, 52
wind_graph, 53
wind_graph(), *25*, *27*, *32*, *39*
wind_graph-class, 54
wind_rose, *14*, 54
wind_rose(), *5*, *6*, *8*, *9*, *21*, *25–27*, *31–33*, *38*, *41*, *58*
wind_rose-class, 56
wind_rose-grid, 56
wind_rose_cache, 57
wind_rose_cache(), *10*
wind_rose_catalog, 57
wind_rose_catalog(), *9*, *10*
wind_series, 58
wind_series(), *8*, *55*, *62*
wind_series-class, 59
wind_times, 59
wind_times(), *50–52*, *59*
wind_trails, 60
wind_trails(), *13*, *17*, *21*, *23*, *26*, *31*
windrose_rasters (wind_rose), 54
windscape-layers, 62
windscape_example, 63
ws_summarize, 64