Package ‘bioRad’

April 1, 2020

Title Biological Analysis and Visualization of Weather Radar Data
Version 0.5.1
Description Extract, visualize and summarize aerial movements of birds and insects from weather radar data. See <doi:10.1111/ecog.04028> for a software paper describing package and methodologies.
License MIT + file LICENSE
URL https://github.com/adokter/bioRad,
    https://adokter.github.io/bioRad
BugReports https://github.com/adokter/bioRad/issues
biocViews
Depends R (>= 3.5.0)
Imports assertthat, curl, fields, ggmap, ggplot2, graphics, methods,
    raster, rgdal, rhdf5, sp, stats, tidyr, utils, viridisLite,
    viridis, maptools
Suggests knitr, testthat
LazyData true
Encoding UTF-8
VignetteBuilder knitr
RoxygenNote 7.1.0
NeedsCompilation no
Author Adriaan M. Dokter [aut, cre] (<https://orcid.org/0000-0001-6573-066X>),
    Peter Desmet [aut] (<https://orcid.org/0000-0002-8442-8025>),
    Stijn Van Hoey [aut] (<https://orcid.org/0000-0001-6413-3185>),
    Bart Kranstauber [ctb] (<https://orcid.org/0000-0001-8303-780X>),
    Jurriaan Spaaks [ctb],
    Lourens Veen [ctb],
    Liesbeth Verlinden [ctb] (<https://orcid.org/0000-0003-1744-9325>),
    Hidde Leijnse [ctb] (<https://orcid.org/0000-0001-7835-4480>)
Maintainer Adriaan M. Dokter <amd427@cornell.edu>
Repository CRAN
Date/Publication 2020-04-01 17:30:09 UTC
R topics documented:

apply_mistnet ........................................ 3
as.data.frame.vp ......................................... 5
as.data.frame.vpts ......................................... 7
beam_distance ........................................... 9
beam_height ............................................. 10
beam_profile ........................................... 11
beam_profile_overlap .................................. 12
beam_range ............................................. 14
beam_width ............................................. 15
bind_into_vpts ........................................... 16
c.vp ...................................................... 17
calculate_param ......................................... 18
calculate_vp ............................................. 19
check_docker ............................................ 23
check_night .............................................. 23
composite_ppi ........................................... 25
dbz_to_eta ................................................ 27
download_basemap ........................................ 28
download_vpfiles .......................................... 29
eta_to_dbz ................................................ 30
example_scan ............................................ 31
example_vp ............................................... 32
example_vpts ............................................. 32
filter_vpts ............................................... 33
generate_elevation_angles ................................ 34
generate_odim_object_type .............................. 35
generate_param .......................................... 36
generate_quantity ........................................ 36
generate_scan ............................................ 38
integrate_profile ........................................ 39
integrate_to_ppi .......................................... 42
is.pvolfile ............................................... 46
is.vpfile ................................................. 47
map ......................................................... 47
nexrad_to_odim ........................................... 50
nyquist_velocity ......................................... 51
plot.ppi ................................................... 52
plot.scan .................................................. 53
plot.vp ..................................................... 55
plot.vpi .................................................... 56
plot.vpts .................................................. 58
project_as_ppi .......................................... 59
rcs ......................................................... 62
rcs<- ..................................................... 63
read_cajun .............................................. 64
read_pvolfile ............................................ 65
**Description**

Apply MistNet segmentation model to a polar volume file on disk, and load the resultant segmentation as a polar volume (`pvol`) object.

**Usage**

```r
apply_mistnet(
  file,
  pvolfile_out,
  verbose = FALSE,
  mount = dirname(file),
  load = TRUE,
  mistnet_elevations = c(0.5, 1.5, 2.5, 3.5, 4.5)
)
```

**Arguments**

- `file` character. File path for a radar polar volume.
- `pvolfile_out` character. Filename for the polar volume to be stored, including the MistNet segmentation results.
- `verbose` logical. When TRUE, pipe Docker stdout to R console. On Windows always TRUE.
mount character. String with the mount point (a directory path) for the Docker container.

load on completion load the data

mistnet_elevations numeric vector of length 5. Elevation angles to feed to the MistNet segmentation model, which expects exactly 5 elevation scans at 0.5, 1.5, 3.5, 3.5 and 4.5 degrees. Specifying different elevation angles may compromise segmentation results.

Details

MistNet is a deep convolutional neural network that has been trained using labels derived from S-band dual-polarization data across the US NEXRAD network.

It’s purpose is to screen out areas of precipitation in weather radar data, primarily legacy data for which dual-polarization data are not available.

Because the network has been trained on S-band data, it may not perform as well on C-band.

MistNet requires three single-polarization parameters as input: reflectivity (DBZH), radial velocity (VRADH), and spectrum width (WRADH), at 5 specific elevation angles (0.5, 1.5, 3.5, 3.5 and 4.5 degrees). Based on these data it can estimate a segmentation mask that identifies pixels with weather that should be removed when interested only in biological data.

MistNet will calculate three class probabilities (from 0 to 1, with 1 corresponding to a 100% probability) as additional scan parameters to the polar volume:

"BACKGROUND" class probability that no signal was detected above the noise level of the radar

"WEATHER" class probability that weather was detected

"BIOLOGY" class probability that biological scatterers were detected

These class probabilities are only available for the 5 input elevations used as input for the MistNet model. Based on all the class probabilities a final weather segmentation map calculated, stored as scan parameter CELL, which is available for all elevation scans.

"CELL" Final weather segmentation, with values > 1 indicating pixels classified as weather, and values equal to 1 indicating pixels that are located within 5 km distance of a weather pixels

A pixel is classified as weather if the class probability WEATHER > 0.45 or when the average class probability for rain across all five MistNet elevation scans at that spatial location > 0.45.

MistNet may run more slowly on Windows than on Linux or Mac OSX.

See Lin et al. 2019 for details.

Value

If parameter load is TRUE an object of class pvol on success. If parameter load is FALSE, TRUE on success.
References

Please also cite this publication when using MistNet:


Examples

```r
## Not run:
# download a NEXRAD file, save as KBGM_example
download.file(paste("https://noaa-nexrad-level2.s3.amazonaws.com/",
"2019/10/01/KBGM/KBGM20191001_000542_V06",
sep = ""), 
"~/KBGM_example")

# calculate MistNet segmentation:
mistnet_pvol <- apply_mistnet("~/KBGM_example")

# print summary info for the segmented elevation scan at 0.5 degree,
# verify new parameters BIOLOGY, WEATHER, BACKGROUND and CELL have been added:
my_scan <- get_scan(mistnet_pvol, 0.5)

# project the scan as a ppi:
my_ppi <- project_as_ppi(my_scan, range_max = 100000)

# plot the reflectivity parameter:
plot(my_ppi, param = "DBZH")

# plot the MistNet class probability [0-1] for weather
plot(my_ppi, param = "WEATHER")

# plot the MistNet class probability [0-1] for biology
plot(my_ppi, param = "BIOLOGY")

# plot the final segmentation result, with values >1 indicating
# areas classified as weather, and value 1 pixels that fall within an
# additional 5 km fringe around weather areas.
plot(my_ppi, param = "CELL")

# clean up:
file.remove("~/KBGM_example")

## End(Not run)
```

as.data.frame.vp

**Convert a vertical profile (vp) to a Data Frame**
Description

Converts a vertical profile to a Data Frame, and optionally adds information on sunrise/sunset, day/night and derived quantities like migration traffic rates.

Usage

```r
## S3 method for class 'vp'
as.data.frame(
  x,
  row.names = NULL,
  optional = FALSE,
  quantities = names(x$data),
  suntime = TRUE,
  geo = TRUE,
  elev = -0.268,
  lat = NULL,
  lon = NULL,
  ...
)
```

Arguments

- **x**: An object of class `vp`.
- **row.names**: NULL or a character vector giving the row names for the data frame. Missing values are not allowed.
- **optional**: If FALSE then the names of the variables in the data frame are checked to ensure that they are syntactically valid variable names and are not duplicated.
- **quantities**: An optional character vector with the names of the quantities to include as columns in the data frame.
- **suntime**: Logical, when TRUE, adds sunrise/sunset and day/night information to each row.
- **geo**: Logical, when TRUE, adds latitude, longitude and antenna height of the radar to each row.
- **elev**: Sun elevation in degrees, see `sunrise/sunset`.
- **lat**: Radar latitude in decimal degrees. When set, overrides the latitude stored in x in `sunrise/sunset` calculations
- **lon**: Radar longitude in decimal degrees. When set, overrides the longitude stored in x in `sunrise/sunset` calculations.
- **...**: Additional arguments to be passed to or from methods.

Details

Note that only the "dens" quantity is thresholded for radial velocity standard deviation by `sd_vvp_threshold`. Note that this is different from the default `plot.vp`, `plot.vpts` and `get_quantity.vp` functions, where quantities "eta", "dbz", "ff", "u", "v", "w", "dd" are all thresholded by `sd_vvp_threshold`
Value

An object of class data.frame.

Examples

```r
# load an example vertical profile time series object
data(example_vp)

# print some summary information
grep("\", example_vp)

# convert the object to a data.frame
df <- as.data.frame(example_vp)

df <- as.data.frame(example_vp, suntime = FALSE)

df <- as.data.frame(example_vp, suntime = TRUE, lat = 50, lon = 4)
```

---

**as.data.frame.vpts** Convert a time series of vertical profiles (vpts) to a data frame

**Description**

Converts vertical profile time series (objects of class vpts) to a data frame, and optionally adds information on sunrise/sunset, day/night and derived quantities like migration traffic rates.

**Usage**

```r
## S3 method for class 'vpts'
as.data.frame(
x,
row.names = NULL,
optional = FALSE,
quantities = names(x$data),
suntime = TRUE,
geo = TRUE,
elev = -0.268,
lat = NULL,
lon = NULL,
...)
```
Arguments

- **x**: An object of class `vpts`.
- **row.names**: NULL or a character vector giving the row names for the data frame. Missing values are not allowed.
- **optional**: If FALSE then the names of the variables in the data frame are checked to ensure that they are syntactically valid variable names and are not duplicated.
- **quantities**: An optional character vector with the names of the quantities to include as columns in the data frame.
- **suntime**: Logical, when TRUE, adds sunrise/sunset and day/night information to each row.
- **geo**: Logical, when TRUE, adds latitude, longitude and antenna height of the radar to each row.
- **elev**: Sun elevation in degrees, see *sunrise/sunset*.
- **lat**: Radar latitude in decimal degrees. When set, overrides the latitude stored in x in *sunrise/sunset* calculations.
- **lon**: Radar longitude in decimal degrees. When set, overrides the longitude stored in x in *sunrise/sunset* calculations.
- **...**: Additional arguments to be passed to or from methods.

Details

Note that only the 'dens' quantity is thresholded for radial velocity standard deviation by `sd_vvp_threshold`. Note that this is different from the default `plot.vp`, `plot.vpts` and `get_quantity.vp` functions, where quantities "eta", "dbz", "ff", "u", "v", "w", "dd" are all thresholded by `sd_vvp_threshold`.

Value

An object of class `data.frame`.

Examples

```r
# load an example vertical profile time series object
data(example_vpts) example_vpts

# convert the object to a data.frame
df <- as.data.frame(example_vpts)

# do not compute sunrise/sunset information
df <- as.data.frame(example_vpts, suntime = FALSE)

# override the latitude/longitude information stored in the object
# when calculating sunrise / sunset
df <- as.data.frame(example_vpts, suntime = TRUE, lat = 50, lon = 4)

# print first then rows of data.frame to console:
df[1:10, ]
```
beam_distance

Calculate radar beam distance

Description

Calculates the distance as measured over the earth’s surface (the down range) for a given beam elevation and slant range.

Usage

beam_distance(range, elev, k = 4/3, lat = 35, re = 6378, rp = 6357)

Arguments

- **range** numeric. Slant range in m, the length of the skywave path between target and the radar antenna.
- **elev** numeric. Beam elevation in degrees.
- **k** Standard refraction coefficient.
- **lat** Geodetic latitude of the radar in degrees.
- **re** Earth equatorial radius in km.
- **rp** Earth polar radius in km.

Details

depends on beam_height to calculate beam height.

Value

numeric. Beam distance (down range) in m.

Examples

# down range of the 5 degree elevation beam at a slant range of 100 km:
beam_distance(100000, 5)
beam_height

*Calculate radar beam height*

**Description**

Calculates the height of a radar beam as a function of elevation and range, assuming the beam is emitted at surface level.

**Usage**

`beam_height(range, elev, k = 4/3, lat = 35, re = 6378, rp = 6357)`

**Arguments**

- `range` numeric. Slant range in m, the length of the skywave path between target and the radar antenna.
- `elev` numeric. Beam elevation in degrees.
- `k` Standard refraction coefficient.
- `lat` Geodetic latitude of the radar in degrees.
- `re` Earth equatorial radius in km.
- `rp` Earth polar radius in km.

**Details**

To account for refraction of the beam towards the earth’s surface, an effective earth’s radius of $k \times$ (true radius) is assumed, with $k = 4/3$.

The earth’s radius is approximated as a point on a spheroid surface, with `re` the longer equatorial radius, and `rp` the shorter polar radius. Typically uncertainties in refraction coefficient are relatively large, making oblateness of the earth and the dependence of earth radius with latitude only a small correction. Using default values assumes an average earth’s radius of 6371 km.

**Value**

numeric. Beam height in m.

**Examples**

# beam height in meters at 10 km range for a 1 degree elevation beam:
beam_height(10000, 1)

# beam height in meters at 10 km range for a 3 and 5 degree elevation beam:
beam_height(10000, c(3, 5))

# define ranges from 0 to 1000000 meter (100 km), in steps of 100 m:
range <- seq(0, 100000, 100)

# plot the beam height of the 0.5 degree elevation beam:
plot(range, beam_height(range, 0.5), ylab = "beam height [m]", xlab = "range [m]")
beam_profile  

*Calculate vertical radiation profile*

**Description**

Calculate for a set of beam elevations elev the altitudinal normalized distribution of radiated energy by those beams.

**Usage**

```r
beam_profile(
  height,
  distance,
  elev,
  antenna = 0,
  beam_angle = 1,
  k = 4/3,
  lat = 35,
  re = 6378,
  rp = 6357
)
```

**Arguments**

- `height`: numeric. Height in meter.
- `distance`: numeric. Distance from the radar as measured along sea level (down range) in m.
- `elev`: numeric vector. Beam elevation(s) in degrees.
- `antenna`: numeric. Height of the center of the radar antenna in meters
- `beam_angle`: numeric. Beam opening angle in degrees, typically the angle between the half-power (-3 dB) points of the main lobe
- `k`: Standard refraction coefficient.
- `lat`: Geodetic latitude of the radar in degrees.
- `re`: Earth equatorial radius in km.
- `rp`: Earth polar radius in km.

**Details**

Beam profile is calculated using `beam_height` and `beam_width`. Returns a beam profile as a function of height relative to ground level.

Returns the normalized altitudinal pattern of radiated energy as a function of altitude at a given distance from the radar, assuming the beams are emitted at antenna level.
beam_profile_overlap

Calculate overlap between a vertical profile ('vp') of biological scatterers and the vertical radiation profile emitted by the radar

**Value**

numeric vector. Normalized radiated energy at each of the specified heights.

**Examples**

```r
# plot the beam profile, for a 0.5 degree elevation beam at 50 km distance from the radar:
plot(beam_profile(height = 0:4000, 50000, 0.5), 0:4000,
   xlab = "normalized radiated energy",
   ylab = "height [m]", main = "beam elevation: 0.5 deg, distance=50km"
)

# plot the beam profile, for a 2 degree elevation beam at 50 km distance from the radar:
plot(beam_profile(height = 0:4000, 50000, 2), 0:4000,
   xlab = "normalized radiated energy",
   ylab = "height [m]", main = "beam elevation: 2 deg, distance=50km"
)

# plot the combined beam profile for a 0.5 and 2.0 degree elevation beam
# at 50 km distance from the radar:
plot(beam_profile(height = 0:4000, 50000, c(0.5, 2)), 0:4000,
   xlab = "normalized radiated energy",
   ylab = "height [m]", main = "beam elevations: 0.5,2 deg, distance=50km"
)
```


description

Calculates the distribution overlap between a vertical profile ('vp') and the vertical radiation profile of a set of emitted radar beams at various elevation angles as given by `beam_profile`

**Usage**

```r
beam_profile_overlap(
  vp,
  elev,
  distance,
  antenna,
  zlim = c(0, 4000),
  noise_floor = -Inf,
  noise_floor_ref_range = 1,
  steps = 500,
  quantity = "dens",
  normalize = T,
  beam_angle = 1,
  k = 4/3,
)```
beam_profile_overlap

\[
\text{lat,}
\text{re} = 6378,
\text{rp} = 6357
\]

Arguments

- **vp**: a vertical profile of class vp
- **elev**: numeric vector. Beam elevation(s) in degrees.
- **distance**: the distance(s) from the radar along sea level (down range) for which to calculate the overlap in m.
- **antenna**: radar antenna height. If missing taken from \(vp\)
- **zlim**: altitude range in meter, given as a numeric vector of length two.
- **noise_floor**: The system noise floor in dBZ. The total system noise expressed as the reflectivity factor it would represent at a distance \(\text{noise}\_\text{floor}\_\text{ref}\_\text{range}\) from the radar. NOT YET IMPLEMENTED
- **noise_floor\_ref\_range**: the reference distance from the radar at which \(\text{noise}\_\text{floor}\) is expressed. NOT YET IMPLEMENTED
- **steps**: number of integration steps over altitude range \(\text{zlim}\), defining altitude grid size used for numeric integrations
- **quantity**: profile quantity to use for the altitude distribution, one of 'dens' or 'eta'.
- **normalize**: Whether to normalize the radiation coverage pattern over the altitude range specified by \(\text{zlim}\)
- **beam_angle**: numeric. Beam opening angle in degrees, typically the angle between the half-power (-3 dB) points of the main lobe
- **k**: Standard refraction coefficient.
- **lat**: radar latitude. If missing taken from \(vp\)
- **re**: Earth equatorial radius in km.
- **rp**: Earth polar radius in km.

Details

This function also calculates the overlap quantity in the output of `integrate_to_ppi`. Overlap is calculated as the Bhattacharyya coefficient \([\text{Bhattacharyya}\_\text{coefficient}][1]\) (i.e. distribution overlap) between the (normalized) vertical profile \(vp\) and the (normalized) radiation coverage pattern as calculated by \(\text{beam}\_\text{profile}\).

The current implementation does not (yet) take into account the system noise floor when calculating the overlap.

In the ODIM data model the attribute `/how/NEZ` or `/how/NEZH` specifies the system noise floor (the Noise Equivalent Z or noise equivalent reflectivity factor. the H refers to the horizontal channel of a dual-polarization radar). In addition, the attribute `/how/LOG` gives "security distance above mean noise level (dB) threshold value". This is equivalent to the log receiver signal-to-noise ratio, i.e. the dB above the noise floor for the signal processor to report a valid reflectivity value. We recommend
using NEZH+LOG for noise_floor, as this is the effective noise floor of the system below which no data will be reported by the radar signal processor.

Typical values are NEZH = -45 to -50 dBZ at 1 km from the radar. LOG is typically around 1 dB.

Need to evaluate beam by beam the returned signal relative to a uniform beam filling of at least NEZH + LOG If returned signal is lower, the gate is below noise level.

Value

A data.frame with columns distance and overlap.

Examples

# locate example volume file:
pvolfile <- system.file("extdata", "volume.h5", package = "bioRad")

# load the example polar volume file:
pvol <- read_pvolfile(pvolfile)

# let us use this example vertical profile:
data(example_vp)
example_vp

# calculate overlap between vertical profile of birds
# and the vertical radiation profile emitted by the radar:
bpo <- beam_profile_overlap(example_vp, get_elevation_angles(pvol), seq(0, 100000, 1000))

# plot the calculated overlap:
plot(bpo)

beam_range  Calculate radar beam range

Description

Calculates the range (i.e. slant range) given a distance measured along the earth’s surface (i.e. down range) and beam elevation.

Usage

beam_range(distance, elev, k = 4/3, lat = 35, re = 6378, rp = 6357)

Arguments

distance numeric. Distance from the radar as measured along sea level (down range) in m.
elev numeric. Beam elevation in degrees.
k Standard refraction coefficient.
beam_width

lat       Geodetic latitude of the radar in degrees.
re       Earth equatorial radius in km.
rp       Earth polar radius in km.

Details
depends on beam_height to calculate beam height.

Value
numeric. Beam range (slant range) in m.

Examples
# slant range of the 5 degree elevation beam at a down range of 100 km:
beam_range(100000, 5)

beam_width Calculate radar beam width

Description
Calculates the width of a radar beam as a function of range and beam angle.

Usage

beam_width(range, beam_angle = 1)

Arguments

range             numeric. Range (distance from the radar antenna) in m.
beam_angle         numeric. Beam opening angle in degrees, typically the angle between the half-
                    power (-3 dB) points of the main lobe

Value
numeric. Beam width in m.

Examples

' # beam width in meters at 10 km range:
beam_width(10000)

# define ranges from 0 to 100000 meter (100 km), in steps of 100 m:
range <- seq(0, 100000, 100)

# plot the beam width as a function of range:
plot(range, beam_width(range), ylab = "beam width [m]", xlab = "range [m]")
bind_into_vpts | Bind vertical profiles (vp) into time series (vpts)

Description

Binds vertical profiles (vp) into a vertical profile time series (vpts), sorted in time. Can also bind multiple vpts of a single radar into one vpts.

Usage

bind_into_vpts(x, ...)

## S3 method for class 'vp'
bind_into_vpts(...)

## S3 method for class 'list'
bind_into_vpts(x, ...)

## S3 method for class 'vpts'
bind_into_vpts(..., attributes_from = 1)

Arguments

x | A vp, vpts or a vector of these.
...
| A vp, vpts or a vector of these.
attributes_from
| integer. Which vpts to copy attributes from (default: first).

Value

A vpts for a single radar or a list of vpts for multiple radars. Input vp are sorted in time in the output vpts.

Methods (by class)

- vp: Bind multiple vp into a vpts. If vp for multiple radars are provided, a list is returned containing a vpts for each radar.
- list: Bind multiple vp objects into a vpts. If data for multiple radars is provided, a list is returned containing a vpts for each radar.
- vpts: Bind multiple vpts into a single vpts. Requires the input vpts to be from the same radar.
c.vp

Examples

# load example time series of vertical profiles:
data(example_vpts)

# split the vpts into two separate time series, one containing profile 1-10,
# and a second containing profile 11-20:
vpts1 <- example_vpts[1:10]
vpts2 <- example_vpts[11:20]

# use bind_into_vpts to bind the two together:
vpts1and2 <- bind_into_vpts(vpts1, vpts2)

# verify that the binded vpts now has 20 profiles, 10 from vpts1 and 10 from
# vpts2:
summary(vpts1and2)

# extract two profiles:
vp1 <- example_vpts[1]
vp1
vp2 <- example_vpts[2]
vp2

# bind the two profiles back into a vpts:
bind_into_vpts(vp1, vp2)

---

c.vp

Concatenate vertical profiles (vp) into a list of vertical profiles

Description

Concatenate vertical profiles (vp) into a list of vertical profiles

Usage

## S3 method for class 'vp'
c(...)

Arguments

... objects of class vp

Value

an object of class list
calculate_param  

**Calculate a new scan parameter**

**Description**

Calculates a new scan parameter from a combination of existing scan parameters. Useful for calculating quantities that are defined in terms of other basic radar moments, like linear reflectivity eta, depolarization ratio (Kilambi et al. 2018), or for applying clutter corrections (CCORH) to uncorrected reflectivity moments (TH), as in TH+CCORH.

**Usage**

```r
calculate_param(x, ...)
```

## S3 method for class 'pvol'

```r
calculate_param(x, ...)
```

## S3 method for class 'scan'

```r
calculate_param(x, ...)
```

**Arguments**

- `x`  
  an object of class `pvol` or class `scan`

- `...`  
  an expression defining the new scan parameter in terms of existing scan parameters

**Value**

an object of the same class as `x`, either class `pvol` or class `scan`

**Methods (by class)**

- `pvol`: Calculate a new scan parameter for all scans in a polar volume.
- `scan`: Calculate a new scan parameter for a scan

**References**

  [https://doi.org/10.1175/JTECH-D-17-0175.1](https://doi.org/10.1175/JTECH-D-17-0175.1)
Examples

# locate example volume file:
pvolfile <- system.file("extdata", "volume.h5", package = "bioRad")

# load the file:
example_pvol <- read_pvolfile(pvolfile)

# calculate linear reflectivity ETA from reflectivity factor DBZH:
radar_wavelength <- example_pvol$attributes$how$wavelength
example_pvol <- calculate_param(example_pvol, ETA = dbz_to_eta(DBZH, radar_wavelength))

# add depolarization ratio (DR) as a scan parameter (see Kilambi 2018):
example_pvol <- calculate_param(example_pvol, DR = 10 * log10((ZDR + 1 - 2 * ZDR^0.5 * RHOHV) /
(ZDR + 1 + 2 * ZDR^0.5 * RHOHV)))

# calculate_param operates on both pvol and scan objects:
calculate_param(example_scan, DR = 10 * log10((ZDR + 1 - 2 * ZDR^0.5 * RHOHV) /
(ZDR + 1 + 2 * ZDR^0.5 * RHOHV)))


calculate vp

Calculate a vertical profile (vp) from a polar volume (pvol)

Description

Calculates a vertical profile of biological scatterers (vp) from a polar volume (pvol) using the algorithm vol2bird (Dokter et al. 2011).

Usage

calculate_vp(
  file,             
  vpfile = "",      
  pvolfile_out = "", 
  autoconf = FALSE, 
  verbose = FALSE, 
  mount = dirname(file[1]), 
  sd_vvp_threshold, 
  rcs = 11,         
  dual_pol = FALSE, 
  rho_hv = 0.95,    
  elev_min = 0,     
  elev_max = 90,    
  azim_min = 0,     
  azim_max = 360,   
  range_min = 5000, 
  range_max = 35000, 
  n_layer = 20L,    
  h_layer = 200,    
)
dealias = TRUE,
nyquist_min = if (dealias) 5 else 25,
dbz_quantity = "DBZH",
mistnet = FALSE,
local_install,
pvolfile
)

Arguments

file string or a vector of strings with radar file(s) for a radar polar volume. Provide either a single file containing a polar volume, or multiple files with single scans/sweeps. Data format should be either ODIM format, which is the implementation of the OPERA data information model in HDF5 format, or a format supported by the RSL library, or Vaisala IRIS (IRIS RAW) format.

vpfile character. Filename for the vertical profile to be generated in ODIM HDF5 format (optional).

pvolfile_out character. Filename for the polar volume to be generated in ODIM HDF5 format (optional, e.g. for converting RSL formats to ODIM).

calculate_vp.autoconf logical. When TRUE, default optimal configuration settings are selected automatically, and other user settings are ignored.

verbose logical. When TRUE, pipe Docker stdout to R console. On Windows always TRUE.

mount character. String with the mount point (a directory path) for the Docker container.

calculate_vp.sd_vvp_threshold numeric. Lower threshold in radial velocity standard deviation (profile quantity sd_vvp) in m/s. Biological signals with sd_vvp < sd_vvp_threshold are set to zero. Defaults to 2 m/s for C-band radars and 1 m/s for S-band radars if not specified.

calculate_vp.rcs numeric. Radar cross section per bird in cm^2.

calculate_vp.dual_pol logical. When TRUE use dual-pol mode, in which meteorological echoes are filtered using the correlation coefficient rho_hv. When FALSE use single polarization mode based only on reflectivity and radial velocity quantities.

calculate_vp.rho_hv numeric. Lower threshold in correlation coefficient used to filter meteorological scattering.

calculate_vp.elev_min numeric. Minimum scan elevation in degrees.

calculate_vp.elev_max numeric. Maximum scan elevation in degrees.

calculate_vp.azim_min numeric. Minimum azimuth in degrees clockwise from north.

calculate_vp.azim_max numeric. Maximum azimuth in degrees clockwise from north.

calculate_vp.range_min numeric. Minimum range in m.

calculate_vp.range_max numeric. Maximum range in m.

calculate_vp.n_layer numeric. Number of altitude layers in the profile.

calculate_vp.h_layer numeric. Width of altitude layers in meter.
### calculate_vp

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>dealias</strong></td>
<td>logical. Whether to dealias radial velocities; this should typically be done when the scans in the polar volume have low Nyquist velocities (below 25 m/s).</td>
</tr>
<tr>
<td><strong>nyquist_min</strong></td>
<td>numeric. Minimum Nyquist velocity of scans in m/s for scans to be included in the analysis.</td>
</tr>
<tr>
<td><strong>dbz_quantity</strong></td>
<td>character. One of the available reflectivity factor quantities in the ODIM radar data format, e.g. DBZH, DBZV, TH, TV.</td>
</tr>
<tr>
<td><strong>mistnet</strong></td>
<td>logical. Whether to use MistNet segmentation model.</td>
</tr>
<tr>
<td><strong>local_install</strong></td>
<td>(optional) String with path to local vol2bird installation, see details.</td>
</tr>
<tr>
<td><strong>pvolfile</strong></td>
<td>deprecated argument renamed to <strong>file</strong>.</td>
</tr>
</tbody>
</table>

### Details

Requires a running Docker daemon (unless a local installation of vol2bird is specified with **local_install**). Common arguments set by users are **file**, **vpfile**, **autoconf** and **mount**.

Turn on **autoconf** to automatically select the optimal parameters for a given radar file. The default for C-band data is to apply rain-filtering in single polarization mode, as well as dual polarization mode when available.

The default for S-band data is to apply precipitation filtering in dual-polarization mode.

Arguments that sometimes require non-default values are: **rcs**, **sd_vvp_threshold**, **range_max**, **dual_pol**, **dealias**.

Other arguments are typically left at their defaults.

**azim_min** and **azim_max** only affects reflectivity-derived estimates in the profile (DBZH, eta, dens), not radial-velocity derived estimates (u, v, w, ff, dd, sd_vvp), which are estimated on all azimuths at all times. **azim_min**, **azim_max** may be set to exclude an angular sector with high ground clutter.

**range_max** may be extended up to 40,000 m for volumes with low elevations only, in order to extend coverage to higher altitudes.

For altitude layers with a VVP-retrieved radial velocity standard deviation value below the threshold **sd_vvp_threshold**, the bird density **dens** is set to zero (see vertical profile **vp** class). This threshold might be dependent on radar processing settings. Results from validation campaigns so far indicate that 2 m/s is the best choice for this parameter for most C-band weather radars, which is used as the C-band default. For S-band, the default threshold is 1 m/s. The algorithm has been tested and developed for altitude layers with **h_layer** = 200 m. Smaller widths are not recommended as they may cause instabilities of the volume velocity profiling (VVP) and dealiasing routines, and effectively lead to pseudo-replicated altitude data, since altitudinal patterns smaller than the beam width cannot be resolved.

The default radar cross section (11 cm^2) corresponds to the average value found by Dokter et al. in a calibration campaign of a full migration autumn season in western Europe at C-band. It’s value may depend on radar wavelength. **rcs** will scale approximately $M^{2/3}$ with $M$ the bird’s mass.

Using default values of **range_min** and **range_max** is recommended. Ranges closer than 5 km tend to be contaminated by ground clutter, while range gates beyond 35 km become too wide to resolve the default altitude layer width of 200 meter (see **beam_width**).

For dealiasing, the torus mapping method by Haase et al. is used.

At S-band (radar wavelength ~ 10 cm), currently only **dual_pol** = TRUE mode is recommended.
On repeated calls of `calculate_vp`, the Docker container mount can be recycled from one call to the next if subsequent calls share the same `mount` argument. Re-mounting a Docker container takes time, therefore it is advised to choose a mountpoint that is a parent directory of all volume files to be processed, such that `calculate_vp` calls are as fast as possible.

If you have installed the vol2bird algorithm locally (not possible on Windows) you can call `vol2bird` through this local installation (bypassing the Docker container), which will be faster. Simply point `local_install` to the path of your local `vol2bird` executable. Your local `vol2bird` executable will be called through a bash login shell. LD_LIBRARY_PATH (Linux) or DYLD_LIBRARY_PATH (Mac) should be correctly specified in your `.bashrc` or `.bash_profile` file and contain all the required shared libraries by `vol2bird`. See `vol2bird` installation pages on Github for details.

**Value**

A vertical profile object of class `vp`. When defined, output files `vpfile` and `pvolfile_out` are saved to disk.

**References**

Dokter et al. (2011) is the main reference for the profiling algorithm (vol2bird) underlying this function. When using the `mistnet` option, please also cite Lin et al. 2019. When de-aliasing data, please also cite Haase et al. 2004.


**Examples**

```r
# locate example polar volume file:
pvolfile <- system.file("extdata", "volume.h5", package = "bioRad")

# copy to a home directory with read/write permissions:
file.copy(pvolfile, "~/volume.h5")

# calculate the profile:
## Not run:
profile <- calculate_vp("~/volume.h5")

# print some summary info:
profile

# convert profile to a data.frame:
```
check_docker

as.data.frame(profile)

## End(Not run)

# clean up:
file.remove("~/volume.h5")

check_docker

Check if Docker is running

Description

Checks that Docker daemon is running correctly on the local system, and that vol2bird Docker image is available.

Usage

check_docker(VERBOSE = TRUE)

Arguments

VERBOSE logical. When TRUE messages are printed to R console.

Value

0 upon success, otherwise an error code: 1 if Docker vol2bird image not available, 2 if Docker daemon not running, 3 if Docker daemon not found.

Examples

## Not run:
# check if Docker is running and vol2bird image is available:
check_docker()

## End(Not run)

check_night

Check if it is night at a given time and place

Description

Checks if it is night (TRUE/FALSE) for a combination of latitude, longitude, date and sun elevation. When used on a bioRad object (pvol, vp, vpts) this information is extracted from the bioRad object directly.
check_night

Usage

    check_night(x, ..., elev = -0.268)

    ## Default S3 method:
    check_night(x, lon, lat, ..., tz = "UTC", elev = -0.268)

    ## S3 method for class 'vp'
    check_night(x, ..., elev = -0.268)

    ## S3 method for class 'list'
    check_night(x, ..., elev = -0.268)

    ## S3 method for class 'vpts'
    check_night(x, ..., elev = -0.268)

    ## S3 method for class 'pvol'
    check_night(x, ..., elev = -0.268)

Arguments

    x    pvol, vp or vpts, or a date inheriting from class POSIXct or a string interpretable by as.POSIXct.
    ...
    optional lat,lon arguments.
    elev numeric. Sun elevation in degrees.
    lon numeric. Longitude in decimal degrees.
    lat numeric. Latitude in decimal degrees.
    tz character. Time zone. Ignored when date already has an associated time zone

Details

The angular diameter of the sun is about 0.536 degrees, therefore the moment of sunrise/sunset corresponds to half that elevation at -0.268 degrees.

    check_night() evaluates to FALSE when the sun has a higher elevation than parameter elev, otherwise TRUE.

    Approximate astronomical formula are used, therefore the day/night transition may be off by a few minutes.

Value

    TRUE when night, FALSE when day, NA if unknown (either datetime or geographic location missing).

    For vpts a vector of TRUE/FALSE values is returned.

Examples

    # check if it is night at UTC midnight in the Netherlands on January 1st:
    check_night("2016-01-01 00:00", 5, 53)
**composite_ppi**

Create a composite of multiple plan position indicators (ppi)

### Description

Combines multiple plan position indicators (ppi) into a single ppi. Can be used to make a composite of ppi’s from multiple radars.

### Usage

```r
composite_ppi(
  x,
  param = "DBZH",
  nx = 100,
  ny = 100,
  xlim,
  ylim,
  res,
  crs,
  raster = NA,
  method = "max",
  idp = 2,
  idw_max_distance = NA
)
```

### Arguments

- **x**: A list of ppi objects.
- **param**: Scan parameter to composite.
- **nx**: number of raster pixels in the x (longitude) dimension
- **ny**: number of raster pixels in the y (latitude) dimension
- **xlim**: x (longitude) range
- **ylim**: y (latitude) range
- **res**: numeric vector of length 1 or 2 to set the resolution of the raster (see res). If this argument is used, arguments nx and ny are ignored. Unit is identical to xlim and ylim.
- **crs**: character or object of class CRS. PROJ.4 type description of a Coordinate Reference System (map projection). When ‘NA’ (default), an azimuthal equidistant projection with origin at the radar location is used. To use a WSG84 (lat,lon) projection, use crs="+proj=longlat +datum=WGS84"
raster (optional) RasterLayer with a CRS. When specified this raster topology is used for the output, and nx, ny, res arguments are ignored.

method string. Compositing method, one of "mean", "min", "max" or "idw"

idp numeric. inverse distance weighting power

idw_max_distance numeric. Maximum distance from the radar to consider in inverse distance weighting. Measurements beyond this distance will have a weighting factor of zero.

Details

This function composites multiple ppi objects into a ppi object that combines all data. Either multiple ppi’s of different scan elevation of the same radar may be combined, or ppi’s of different radars can be composited.

Argument method determines how values of different ppi’s at the same geographic location are combined.

"mean" Compute the average value

"max" Compute the maximum value. If ppi’s are of the same radar and the same polar volume, this computes a max product, showing the maximum detected signal at that geographic location.

"min" Compute the minimum value

"idw" This option is useful primarily when compositing ppi’s of multiple radars. Performs an inverse distance weighting, where values are weighted according to 1/(distance from the radar)^idp

The coordinates system of the returned ppi is a WGS84 (lat,lon) datum.

This function is a prototype and under active development

Value

A ppi.

Examples

# locate example volume file:
pvolfile <- system.file("extdata", "volume.h5", package = "bioRad")

# load the file:
example_pvol <- read_pvolfile(pvolfile)

# calculate a ppi for each elevation scan
my_ppis <- lapply(example_pvol$scans, project_as_ppi)

# overlay the ppi's, calculating the maximum value observed
# across the available scans at each geographic location
my_composite <- composite_ppi(my_ppis, method="max")

## Not run:
# download basemap
bm <- download_basemap(my_composite)

# plot the calculated max product on the basemap
map(my_composite, bm)

## End(Not run)

dbz_to_eta

Convert reflectivity factor to reflectivity

Description

Convert reflectivity factor to reflectivity

Usage

dbz_to_eta(dbz, wavelength, K = 0.93)

Arguments

dbz: reflectivity factor in dBZ
wavelength: radar wavelength in cm
K: norm of the complex refractive index of water

Value

reflectivity in cm^2/km^3

Examples

# calculate eta for a 7 dBZ reflectivity factor at C-band:
dbz_to_eta(7, 5)

# calculate eta for a 7 dBZ reflectivity factor at S-band:
dbz_to_eta(7, 10)

# calculate animal density for a 5 dBZ reflectivity
# factor at C-band and S-band, assuming a
# 11 cm^2 radar cross section per animal:
dbz_to_eta(7, 5) / 11 # C-band
dbz_to_eta(7, 10) / 11 # S-band
download_basemap  Download a basemap for map(ppi)

Description

Downloads a Stamen Maps or Google Maps base layer map using get_map.

Usage

download_basemap(  
  x,  
  verbose = TRUE,  
  zoom,  
  alpha = 1,  
  source = "stamen",  
  maptype = "terrain",  
  ...  
)

Arguments

x  An object of class ppi.
verbose  Logical, whether to print information to console.
zoom  Zoom level (optional), see get_map. An integer from 3 (continent) to 21 (building). By default the zoom level matching the ppi extent is selected automatically.
alpha  Transparency of the basemap (0-1).
source  String identifying which map service should be used: "stamen" or "google".
...
Arguments to pass to get_map function.

Details

To use Google Maps as source, you will have to register with Google, enable billing and provide an API key to ggmap. See the ggmap README for details.

To use maptype, install the development version of ggmap (>3.0.0) with devtools::install_github("dkahle/ggmap").

Examples

# load an example scan:
data(example_scan)

# print summary info for the scan:
example_scan
# make ppi for the scan
ppi <- project_as_ppi(example_scan)
## Not run:
# grab a basemap that matches the extent of the ppi:
basemap <- download_basemap(ppi)

# map the reflectivity quantity of the ppi onto the basemap:
map(ppi, map = basemap, param = "DBZH")

# increase the transparency of the basemap:
basemap <- download_basemap(ppi, alpha = 0.3)
map(ppi, map = basemap, param = "DBZH")

# download a different type of basemap, e.g. a gray-scale image:
# see get_map() in ggmap library for full documentation of options
basemap <- download_basemap(ppi, maptype = "toner-lite")

# map the radial velocities onto the line image:
map(ppi, map = basemap, param = "VRADH")

## End(Not run)

---

**download_vpfies**  
*Download vertical profile (vp) files from the ENRAM data repository*

Description

Download and unzip a selection of vertical profile (vp) files from the ENRAM data repository, where these are stored as monthly zips per radar.

Usage

download_vpfies(
  date_min,
  date_max,
  radars,
  directory = ".",
  overwrite = FALSE
)

Arguments

date_min  character. YYYY-MM-DD start date of file selection. Days will be ignored.
date_max  character. YYYY-MM-DD end date of file selection. Days will be ignored.
radars  character (vector). 5-letter country/radar code(s) (e.g. "bejab") of radars to include in file selection.
directory character. Path to local directory where files should be downloaded and unzipped.

overwrite logical. TRUE for re-downloading and overwriting previously downloaded files of the same names.

Examples

# Download data from radars "bejab" and "bewid", even if previously
# downloaded (overwrite = TRUE). Will successfully download 2016-10 files,
# but show 404 error for 2016-11 files (as these are not available).
## Not run:
dir.create("~/bioRad_tmp_files")
download_vpfiles(
  date_min = "2016-10-01",
  date_max = "2016-11-30",
  radar = c("bejab", "bewid"),
  directory = "~/bioRad_tmp_files",
  overwrite = TRUE
)
# clean up:
unlink("~/bioRad_tmp_files", recursive = T)
## End(Not run)

---

**eta_to_dbz**

Convert reflectivity to reflectivity factor

Description

Convert reflectivity to reflectivity factor

Usage

eta_to_dbz(eta, wavelength, K = 0.93)

Arguments

- **eta**: reflectivity in cm$^2$/km$^3$
- **wavelength**: radar wavelength in cm
- **K**: norm of the complex refractive index of water

Value

reflectivity factor in dBZ
Examples

# reflectivity factor (dBZ) at C-band for a reflectivity eta=10000 cm^2/km^3:
eta_to_dbz(10000, 5)

# reflectivity factor (dBZ) at S-band for a reflectivity eta=10000 cm^2/km^3:
eta_to_dbz(10000, 10)

# expected reflectivity factor (dBZ) for an
# animal density of 1000 individuals/km^3
# and a radar cross section of 11 cm^2 per individual:
# at C-band and S-band:
etta_to_dbz(1000 * 11, 5)  # C-band
netta_to_dbz(1000 * 11, 10)  # S-band

---

example_scan  Example object of class scan

Description

Example of a scan object with name example_scan.

Usage

elemente_example_scan

Format

An object of class scan of dimension 5 x 480 x 360.

Examples

# get summary of example scan:
supply(example_scan)

# example_scan was created with:
pvolfile <- system.file("extdata", "volume.h5", package = "bioRad")
pvol <- read_pvolfile(pvolfile)
extexample_scan <- pvol$scans[[1]]
## Not run:
# save(example_scan, file = "data/example_scan.rda")

## End(Not run)
### example_vp

*Example object of class vp*

#### Description

Example of a `vp` object with name `example_vp`. Can be created with `calculate_vp` or read from file with `read_vpfiles`.

#### Usage

```r
example_vp
```

#### Format

An object of class `vp` with 25 rows and 16 columns.

#### Examples

```r
# get summary of example vp:
summary(example_vp)

# example_vp was created with:
vpfile <- system.file("extdata", "profile.h5", package = "bioRad")
example_vp <- read_vpfiles(vpfile)
## Not run:
# save(example_vp, file = "data/example_vp.rda")
## End(Not run)
```

### example_vpts

*Example object of class vpts*

#### Description

Example of a `vpts` object (a time series of vertical profiles) with name `example_vpts`.

#### Usage

```r
example_vpts
```

#### Format

An object of class `vpts` of dimension 25 x 1934 x 15.
Examples

# get summary of example vpts:
summary(example_vpts)

# example_vpts was created with:
## Not run:
vptsfile <- system.file("extdata", "vpts.txt.zip", package = "bioRad")
unzip(vptsfile, exdir = (dirname(vptsfile)), junkpaths = T)
vptsfile <- substr(vptsfile, 1, nchar(vptsfile) - 4)
example_vpts <- read_vpts(vptsfile, radar = "KBGM", wavelength = "S")
rcs(example_vpts) <- 11
sd_vvp_threshold(example_vpts) <- 2
example_vpts$attributes$where$lat <- 42.2
example_vpts$attributes$where$lon <- -75.98
# save(example_vpts, file = "data/example_vpts.rda", compress = "xz")

## End(Not run)

---

filter_vpts

Time selection in a time series of vertical profiles ('vpts')

Description

Time selection in a time series of vertical profiles ('vpts')

Usage

filter_vpts(x, min, max, nearest)

Arguments

- **x**: A vpts object.
- **min**: Minimum datetime to be included. POSIXct value or character string convertible to POSIXct.
- **max**: Maximum datetime to be included. POSIXct value or character string convertible to POSIXct.
- **nearest**: If specified, min and max are ignored and the profile nearest to the specified datetime is returned. POSIXct value or character string convertible to POSIXct.

Details

returns profiles for which min <= timestamp profile < max.

Value

An object of class 'vpts', or an object of class 'vp' if argument nearest is specified.
get_elevation_angles

Examples

# load example vertical profile time series:
data(example_vpts)
ex_vpts

# select profiles later than 02-Sep-2016
filter_vpts(example_vpts, min = "2016-09-02")

# select the profile nearest to 2016-09-01 03:00 UTC:
filter_vpts(example_vpts, nearest = "2016-09-01 03:00")

# select profiles between than 1 and 3 UTC on 02-Sep-2016:
filter_vpts(example_vpts, min = "2016-09-02 01:00", max = "2016-09-02 03:00")

get_elevation_angles  Get elevation angles of a polar volume (pvol) or scan (scan)

Description

Gives the elevation angle of a single scan, or the elevation angles of all scans within a polar volume

Usage

get_elevation_angles(x)

## S3 method for class 'pvol'
get_elevation_angles(x)

## S3 method for class 'scan'
get_elevation_angles(x)

## S3 method for class 'param'
get_elevation_angles(x)

Arguments

x     A pvol or scan object.

Value

elevation in degrees

Methods (by class)

- pvol: Elevation angles of all scans in a polar volume.
- scan: Elevation angle of a scan.
- param: Elevation angle of a scan parameter.
Examples

# load a polar volume
pvolfile <- system.file("extdata", "volume.h5", package = "bioRad")
example_pvol <- read_pvolfile(pvolfile)

# elevations for the scans in the volume
get_elevation_angles(example_pvol)

# extract the first scan:
scan <- example_pvol$scans[[1]]

# elevation angle of the scan:
get_elevation_angles(scan)

get_odim_object_type

Check the ODIM data class of a polar volume file

Description

Checks which data class is contained in ODIM HDF5 file

Usage

get_odim_object_type(file)

Arguments

file A string containing a file name.

Value

character string pvol for polar volume, vp for vertical profile, otherwise NA

Examples

# locate a polar volume file:
pvolfile <- system.file("extdata", "volume.h5", package = "bioRad")

# check the data type:
get_odim_object_type(pvolfile) # > "P VOL"
get_param

*Get a scan parameter (param) from a scan (scan)*

**Description**

Get a scan parameter (param) from a scan (scan)

**Usage**

`get_param(x, param)`

**Arguments**

- `x` An object of class `scan`
- `param` a scan parameter

**Value**

An object of class `param`.

**Examples**

```r
# we will extract a scan parameter from the example scan object:
example_scan

# extract the VRADH scan parameter
my_param <- get_param(example_scan, "VRADH")
my_param
```

get_quantity

*Get a quantity of a vertical profile (vp) or time series of vertical profiles (vpts)*

**Description**

Get a quantity of a vertical profile (vp) or time series of vertical profiles (vpts)

**Usage**

`get_quantity(x, quantity)`

```r
## S3 method for class 'vp'
get_quantity(x, quantity = "dens")

## S3 method for class 'list'
get_quantity(x, quantity = "dens")
```
get_quantity

## S3 method for class 'vpts'
get_quantity(x, quantity = "dens")

Arguments

x A vp or vpts object.

quantity A profile quantity, one of:
  • "height"
  • "u"
  • "v"
  • "w"
  • "ff"
  • "dd"
  • "sd_vvp"
  • "gap"
  • "dbz"
  • "eta"
  • "dens"
  • "DBZH"
  • "n"
  • "n_all"
  • "n_dbz"
  • "n_dbz_all"

Details

This function grabs any of the data quantities stored in vp or vpts objects. See the documentation of the vertical profile vp class for a description of each of these quantities.

Value

class vp: a named vector for the requested quantity.
class list: a list of a named vectors for the requested quantity.
class vpts: a (height x time) matrix of the requested quantity.

Examples

# load example profile
data(example_vp)

# extract the animal density ("dens") quantity:
get_quantity(example_vp, "dens")
get_scan

Get a scan (scan) from a polar volume (pvol)

Description

Get a scan (scan) from a polar volume (pvol)

Usage

get_scan(x, elev)

Arguments

x An object of class pvol.

elev Elevation angle.

Details

The function returns the scan with elevation angle closest to elev.

Value

An object of class ‘scan’.

Examples

# locate example volume file:
pvolfile <- system.file("extdata", "volume.h5", package = "bioRad")

# load the file:
example_pvol <- read_pvolfile(pvolfile)

# extract the scan at 3 degree elevation:
scan <- get_scan(example_pvol, 3)

# print summary info for this scan:
scan
Vertically integrate profiles (vp or vpts) to an integrated profile (vpi)

Description

Performs a vertical integration of density, reflectivity and migration traffic rate, and a vertical averaging of ground speed and direction weighted by density.

Usage

```r
integrate_profile(x, alt_min, alt_max, alpha = NA, interval_max = Inf)
```

## S3 method for class 'vp'
```r
ingegrate_profile(
  x,
  alt_min = 0,
  alt_max = Inf,
  alpha = NA,
  interval_max = Inf
)
```

## S3 method for class 'list'
```r
integrate_profile(
  x,
  alt_min = 0,
  alt_max = Inf,
  alpha = NA,
  interval_max = Inf
)
```

## S3 method for class 'vpts'
```r
integrate_profile(
  x,
  alt_min = 0,
  alt_max = Inf,
  alpha = NA,
  interval_max = Inf
)
```

Arguments

- `x` A vp or vpts object.
- `alt_min` Minimum altitude in m.
- `alt_max` Maximum altitude in m.
- `alpha` Migratory direction in clockwise degrees from north.
**interval_max**  
Maximum time interval belonging to a single profile in seconds. Traffic rates are set to zero at times $t$ for which no profiles can be found within the period $t-\text{interval}\_\text{max}/2$ to $t+\text{interval}\_\text{max}/2$. Ignored for single profiles of class \_vp_.

**Details**

**Available quantities:** The function generates a specially classed data frame with the following quantities:

- **datetime**  
  POSIXct date of each profile in UTC

- **vid**  
  Vertically Integrated Density in individuals/km$^2$. \textit{vid} is a surface density, whereas \textit{dens} in \_vp objects is a volume density.

- **vir**  
  Vertically Integrated Reflectivity in cm$^2$/km$^2$

- **mtr**  
  Migration Traffic Rate in individuals/km/h

- **rtr**  
  Reflectivity Traffic Rate in cm$^2$/km/h

- **mt**  
  Migration Traffic in individuals/km, cumulated from the start of the time series up to \textit{datetime}

- **rt**  
  Reflectivity Traffic in cm$^2$/km, cumulated from the start of the time series up to \textit{datetime}

- **ff**  
  Horizontal ground speed in m/s

- **dd**  
  Horizontal ground speed direction in degrees

- **u**  
  Ground speed component west to east in m/s

- **v**  
  Ground speed component north to south in m/s

- **height**  
  Mean flight height (height weighted by \textit{eta}) in m above sea level

Vertically integrated density and reflectivity are related according to $\text{vid} = \frac{\text{vir}}{\text{rcs}(x)}$, with \textit{rcs} the assumed radar cross section per individual. Similarly, migration traffic rate and reflectivity traffic rate are related according to $\text{mtr} = \frac{\text{rtr}}{\text{rcs}(x)}$.

**Ground speed (\textit{ff}) and ground speed components (\textit{u},\textit{v}):** The height-averaged ground speed is defined as:

$$ff = \frac{\sum_{i} \text{dens}_i \cdot ff_i}{\sum_{i} \text{dens}_i}$$

with the sum running over all altitude layers between \textit{alt}\_\text{min} and \textit{alt}\_\text{max}, \textit{dens}_i the bird density, $ff_i$ the ground speed at altitude layer $i$.

the height-averaged \textit{u} component (west to east) is defined as:

$$u = \frac{\sum_{i} \text{dens}_i \cdot u_i}{\sum_{i} \text{dens}_i}$$

the height-averaged \textit{v} component (south to north) is defined as:

$$v = \frac{\sum_{i} \text{dens}_i \cdot v_i}{\sum_{i} \text{dens}_i}$$

Note that $ff_i = \sqrt{(u_i^2 + v_i^2)}$, but the same does not hold for the height-integrated speeds, i.e. $ff! = \sqrt{(u^2 + v^2)}$ as soon as the ground speed directions vary with altitude.
Migration traffic rate (mtr) and reflectivity traffic rate (rtr): Migration traffic rate (mtr) for an altitude layer is a flux measure, defined as the number of targets crossing a unit of transect per hour.

Column mtr of the output dataframe gives migration traffic rates in individuals/km/hour. The transect direction is set by the angle alpha. When alpha=NA, the transect runs perpendicular to the measured migratory direction. mtr then equals the number of crossing targets per km transect per hour, for a transect kept perpendicular to the measured migratory movement at all times and altitudes. In this case mtr is always a positive quantity, defined as:

\[ mtr = 3.6 \sum_i dens_i ff_i \Delta h \]

with the sum running over all altitude layers between alt_min and alt_max, dens_i the bird density, ff_i the ground speed at altitude layer i, and \( \Delta h \) the altitude layer width. The factor 3.6 refers to a unit conversion of speeds \( ff_i \) from m/s to km/h.

If alpha is given a numeric value, the transect is taken perpendicular to the direction alpha, and the number of crossing targets per hour per km transect is calculated as:

\[ mtr = 3.6 \sum_i dens_i ff_i \cos(((dd_i - alpha) \pi / 180)) \Delta h \]

with \( dd_i \) the migratory direction at altitude i.

Note that this equation evaluates to the previous equation when alpha equals \( dd_i \). Also note we can rewrite this equation using trigonemetry as:

\[ mtr = 3.6 \sum_i dens_i (u_i \sin(alphapi/180) + v_i \cos(alphapi/180)) \Delta h \]

with \( u_i \) and \( v_i \) the u and v ground speed components at altitude i. In this definition mtr is a traditional flux into a direction of interest. Targets moving into the direction alpha contribute positively to mtr, while targets moving in the opposite direction contribute negatively to mtr. Therefore mtr can be both positive or negative, depending on the definition of alpha.

Note that mtr for a given value of alpha can also be calculated from the vertically integrated density vid and the height-integrated velocity components u and v as follows:

\[ mtr = 3.6(u \sin(alphapi/180) + v \cos(alphapi/180)) vid \]

Formula for reflectivity traffic rate rtr are found by replacing dens with eta and vid with vir in the formula for mtr. Reflectivity traffic rate gives the cross-sectional area passing the radar per km transect perpendicular to the migratory direction per hour. mtr values are conditional on settings of res, while rtr values are not.

Migration traffic (mt) and reflectivity traffic (rt): Migration traffic is calculated by time-integration of migration traffic rates. Migration traffic gives the number of individuals that have passed per km perpendicular to the migratory direction at the position of the radar for the full period of the time series within the specified altitude band.

Reflectivity traffic is calculated by time-integration of reflectivity traffic rates. Reflectivity traffic gives the total cross-sectional area that has passed per km perpendicular to the migratory direction at the position of the radar for the full period of the time series within the specified altitude band.
Values are conditional on settings of `rcs`, while `rt` values are not. Columns `mt` and `rt` in the output dataframe provides migration traffic as a numeric value equal to migration traffic and reflectivity traffic from the start of the time series up till the moment of the time stamp of the respective row.

**Value**

an object of class `vpi`, a data frame with vertically integrated profile quantities

**Methods (by class)**

- `vp`: Vertically integrate a vertical profile.
- `list`: Vertically integrate a list of vertical profiles.
- `vpts`: Vertically integrate a time series of vertical profiles.

**Examples**

```r
# MTR for a single vertical profile
t = integrate_profile(example_vp)

# MTRs for a list of vertical profiles
t = integrate_profile(c(example_vp, example_vp))

# MTRs for a time series of vertical profiles
# load example data:
data(example_vpts)
ex = example_vpts
# print migration traffic rates
t = integrate_profile(ex)
# plot migration traffic rates for the full air column
plot(ex)
# plot migration traffic rates for altitudes > 1 km above sea level
plot(integrate_profile(ex, alt_min = 1000))
# plot the (cumulative) migration traffic
plot(integrate_profile(ex), quantity = "mt")
```

### integrate_to_ppi

`integrate_to_ppi` is a function that calculates a plan position indicator (PPI) of vertically integrated density adjusted for range effects. It is designed to estimate a spatial image (PPI object) of vertically integrated density (VID) based on all elevation scans of the radar, while accounting for the changing overlap between the radar beams as a function of range. The resulting PPI is a vertical integration over the layer of biological scatterers based on all available elevation scans, corrected for range effects due to partial beam overlap with the layer of biological echoes (overshooting) at larger distances from the radar. The methodology is described in detail in Kranstauber et al. (2020).
integrate_to_ppi

Usage

```r
integrate_to_ppi(
  pvol,
  vp,
  nx = 100,
  ny = 100,
  xlim,
  ylim,
  zlim = c(0, 4000),
  res,
  quantity = "eta",
  param = "DBZH",
  raster = NA,
  lat,
  lon,
  antenna,
  beam_angle = 1,
  crs,
  param_ppi = c("VIR", "VID", "R", "overlap", "eta_sum", "eta_sum_expected"),
  k = 4/3,
  re = 6378,
  rp = 6357,
)
```

Arguments

- **pvol**: a polar volume of class pvol
- **vp**: a vertical profile of class vp
- **nx**: number of raster pixels in the x (longitude) dimension
- **ny**: number of raster pixels in the y (latitude) dimension
- **xlim**: x (longitude) range
- **ylim**: y (latitude) range
- **zlim**: altitude range in meter, given as a numeric vector of length two.
- **res**: numeric vector of length 1 or 2 to set the resolution of the raster (see `res`). If this argument is used, arguments `nx` and `ny` are ignored. Unit is identical to `xlim` and `ylim`.
- **quantity**: profile quantity on which to base range corrections, 'eta' or 'dens'.
- **param**: reflectivity factor scan parameter on which to base range corrections. Typically the same parameter from which animal densities are estimated for object `vp`. One of 'DBZH','DBZV','DBZ','TH','TV'.
- **raster**: (optional) RasterLayer with a CRS. When specified this raster topology is used for the output, and `nx`, `ny`, `res` arguments are ignored.
- **lat**: Geodetic latitude of the radar in degrees. If missing taken from `pvol`.
- **lon**: Geodetic latitude of the radar in degrees. If missing taken from `pvol`. 
antenna  radar antenna height. If missing taken from vp
beam_angle numeric. Beam opening angle in degrees, typically the angle between the half-power (-3 dB) points of the main lobe
crs  character or object of class CRS. PROJ.4 type description of a Coordinate Reference System (map projection). When 'NA' (default), an azimuthal equidistant projection with origin at the radar location is used. To use a WGS84 (lat,lon) projection, use crs="+proj=longlat +datum=WGS84"
param_ppi one or multiple of 'VIR', 'VID', 'R', 'overlap', 'eta_sum', 'eta_sum_expected'
k  Standard refraction coefficient.
re  Earth equatorial radius in km.
rp  Earth polar radius in km.

Details

The function requires

- a polar volume, containing one or multiple scans (pvol)
- a vertical profile (of birds) calculated for that same polar volume (vp)
- a grid defined on the earth’s surface, on which we will calculate the range corrected image (defined by raster, or a combination of nx,ny,res arguments).

The pixel locations on the ground are easily translated into a corresponding azimuth and range of the various scans (see function beam_range).

For each scan within the polar volume, the function calculates:

1. the vertical radiation profile for each ground surface pixel for that particular scan, using beam_profile.
2. the reflectivity expected for each ground surface pixel ($\eta_{expected}$), given the vertical profile (of biological scatterers) and the part of the profile radiated by the beam. This $\eta_{expected}$ is simply the average of (linear) $\eta$ in the profile, weighted by the vertical radiation profile.
3. the observed $\eta$ at each pixel $\eta_{observed}$, which is converted from $DBZH$ using function dbz_to_eta, with $DBZH$ the reflectivity factor measured at the pixel’s distance from the radar.

For each pixel on the ground, we thus retrieve a set of $\eta_{expected}$ and a set of $\eta_{observed}$. From those we can calculate a spatial adjustment factor $R$ as:

$$ R = \frac{\sum \eta_{observed}}{\sum \eta_{expected}} $$

, with the sum running over scans.

To arrive at the final PPI image, the function calculates

- the vertically integrated density ($vid$) and vertically integrated reflectivity ($vir$) for the profile, using the function integrate_profile.
- the spatial range-corrected PPI for VID, defined as the adjustment factor image ($R$), multiplied by the vid calculated for the profile.
The spatial range-corrected PPI for VIR, defined as the adjustment factor $R$, multiplied by the vir calculated for the profile.

If one of lat or lon is missing, the extent of the PPI is taken equal to the extent of the data in the first scan of the polar volume.

As an additional parameter, overlap between vertical profile and vertical radiation profile is calculated using `beam_profile` and stored as quantity `overlap`.

Scans at 90 degree beam elevation (birdbath scans) are ignored.

**Value**

An object of class `ppi`.

**References**


**Examples**

```r
# locate example polar volume file:
pvolfile <- system.file("extdata", "volume.h5", package = "bioRad")

# load polar volume
eexample_pvol <- read_pvolfile(pvolfile)

data(example_vp)

# calculate the range-corrected ppi on a 50x50 pixel raster
my_ppi <- integrate_to_ppi(example_pvol, example_vp, nx = 50, ny = 50)

# plot the vertically integrated reflectivity (VIR) using a 0-2000 cm^2/km^2 color scale:
plot(my_ppi, zlim = c(0, 2000))

## Not run:
# calculate the range-corrected ppi on finer 2000m x 2000m pixel raster:
my_ppi <- integrate_to_ppi(example_pvol, example_vp, res = 2000)

# plot the vertically integrated density (VID) using a 0-200 birds/km^2 color scale:
plot(my_ppi, param = "VID", zlim = c(0, 200))

# to overlay ppi objects on a background map, first
# download a basemap, and then map the ppi:

bm <- download_basemap(my_ppi)
map(my_ppi, bm)
```
# the ppi can also be projected on a user-defined raster, as follows:
# first define the raster:
template_raster <- raster::raster(raster::extent(12, 13, 56, 57), crs = sp::CRS("+proj=longlat"))

# project the ppi on the defined raster:
my_ppi <- integrate_to_ppi(example_pvol, example_vp, raster = template_raster)

# extract the raster data from the ppi object:
raster::brick(my_ppi$data)

# calculate the range-corrected ppi on an even finer 500m x 500m pixel raster,
# cropping the area up to 50000 meter from the radar.
my_ppi <- integrate_to_ppi(example_pvol, example_vp, 
  res = 500,
  xlim = c(-50000, 50000), ylim = c(-50000, 50000)
)
plot(my_ppi, param = "VID", zlim = c(0, 200))

## End(Not run)

---

**is.pvolfile**

*Check if a local file is a polar volume (pvol)*

**Description**

Checker whether a file is a polar volume in ODIM hdf5 format that can be read with package *bioRad*

**Usage**

```r
is.pvolfile(file, filename = NULL)
```

**Arguments**

- `file`: A string containing a file name.
- `filename`: Deprecated argument, use `file` instead.

**Details**

The function checks whether a hdf5 file provided as input is a polar volume in ODIM hdf5 format. The function currently evaluates to FALSE for NEXRAD and IRIS RAW polar volume files.

**Value**

TRUE when `file` is a polar volume in readable format, otherwise FALSE
Examples

# locate example file:
pvolfile <- system.file("extdata", "volume.h5", package = "bioRad")

# check that the file is an ODIM hdf5 polar volume:
is.pvolfile(pvolfile) # > TRUE

---

is.vpfile  

*Check if a local file is a vertical profile (vp)*

Description

Checker whether a file is a vertical profile that can be read with package *bioRad*

Usage

`is.vpfile(file, filename = NULL)`

Arguments

- `file`  
  A string containing a filename
- `filename`  
  Deprecated argument, use file instead.

Value

TRUE when `filename` is a vertical profile, otherwise FALSE

Examples

```r
profile <- system.file("extdata", "profile.h5", package = "bioRad")
is.vpfile(profile) # > TRUE
```

---

map  

*Map a plan position indicator (ppi)*

Description

Plot a ppi on a Stamen Maps, OpenStreetMap, Google Maps or Naver Map base layer map using `ggmap`.
Usage

map(x, ...)  

## S3 method for class 'ppi'
map(
  x,
  map,
  param,
  alpha = 0.7,
  xlim,
  ylim,
  zlim = c(-20, 20),
  ratio,
  radar.size = 3,
  radar.color = "red",
  n_color = 1000,
  radar.size = 3,
  radar.color = "red",
  n.color = 1000,
  palette = NA,
  ...
)

Arguments

x An object of class ppi.

... Arguments passed to low level ggmap function.

map The basemap to use, result of a call to download_basemap.

param The scan parameter to plot.

alpha Transparency of the data, value between 0 and 1.

xlim Range of x values to plot (degrees longitude), as atomic vector of length 2.

ylim Range of y values to plot (degrees latitude), as an atomic vector of length 2.

zlim The range of values to plot.

ratio Aspect ratio between x and y scale, by default 1/cos(latituderadar * pi/180).

radar.size Size of the symbol indicating the radar position.

radar.color Color of the symbol indicating the radar position.

n_color The number of colors (>=1) to be in the palette.

radar.size Deprecated argument, use radar_size instead.

radar.color Deprecated argument, use radar_color instead.

n.color Deprecated argument, use n_color instead.

palette (Optional) character vector of hexadecimal color values defining the plot color scale, e.g. output from viridis
Details

Available scan parameters for mapping can be printed to screen by `summary(x)`. Commonly available parameters are:

"DBZH", "DBZ" (Logged) reflectivity factor [dBZ]
"TH", "T" (Logged) uncorrected reflectivity factor [dBZ]
"VRADH", "VRAD" Radial velocity [m/s]. Radial velocities towards the radar are negative, while radial velocities away from the radar are positive
"RHOHV" Correlation coefficient [unitless]. Correlation between vertically polarized and horizontally polarized reflectivity factor
"PHIDP" Differential phase [degrees]
"ZDR" (Logged) differential reflectivity [dB]

The scan parameters are named according to the OPERA data information model (ODIM), see Table 16 in the ODIM specification.

Value

A ggmmap object (a classed raster object with a bounding box attribute).

Methods (by class)

- `ppi`: plot a 'ppi' object on a map

Examples

```r
# load an example scan:
data(example_scan)
# make ppi's for all scan parameters in the scan
ppi <- project_as_ppi(example_scan)
## Not run:
# grab a basemap that matches the extent of the ppi:
# using a gray-scale basemap:
basemap <- download_basemap(ppi, maptype = "toner-lite")

# map the radial velocity scan parameter onto the basemap:
map(ppi, map = basemap, param = "VRADH")

# extend the plotting range of velocities, from -50 to 50 m/s:
map(ppi, map = basemap, param = "VRADH", zlim = c(-50, 50))

# map the reflectivity on a terrain basemap:
basemap <- download_basemap(ppi, maptype = "terrain")
map(ppi, map = basemap, param = "DBZH")

# change the color palette, e.g. Viridis colors:
map(ppi, map = basemap, param = "DBZH", palette = viridis::viridis(100), zlim=c(-10,10))

# give the data more transparency:
map(ppi, map = basemap, param = "DBZH", alpha = 0.3)
```
# change the appearance of the symbol indicating the radar location:
map(ppi, map = basemap, radar_size = 5, radar_color = "blue")

# crop the map:
map(ppi, map = basemap, xlim = c(12.4, 13.2), ylim = c(56, 56.5))

## End(Not run)

---

### nexrad_to_odim

**Convert a NEXRAD polar volume file to an ODIM polar volume file**

#### Description

Convert a NEXRAD polar volume file to an ODIM polar volume file

#### Usage

```r
nexrad_to_odim(
  pvolfile_nexrad,
  pvolfile_odim,
  verbose = FALSE,
  mount = dirname(pvolfile_nexrad)
)
```

#### Arguments

- `pvolfile_nexrad`: Polar volume input file in RSL format.
- `pvolfile_odim`: Filename for the polar volume in ODIM hdf5 format to be generated.
- `verbose`: logical. When TRUE, pipe Docker stdout to R console. On Windows always TRUE.
- `mount`: character. String with the mount point (a directory path) for the Docker container.

#### Value

TRUE on success

#### Examples

```r
## Not run:
# download a NEXRAD file, save as KBGM_example
download.file(paste("https://noaa-nexrad-level2.s3.amazonaws.com/",
  "2019/10/01/KBGM/KBGM20191001_000542_V06",
  sep = "")
), "~/KBGM_example")
```
nyquist_velocity

Nyquist velocity for a given pulse repetition frequency (PRF)

Description

Calculates the Nyquist velocity given a radar’s pulse repetition frequency (PRF) and wavelength. When specifying two PRFs, the extended Nyquist velocity is given for a radar using the dual-PRF technique.

Usage

nyquist_velocity(wavelength, prf1, prf2)

Arguments

- wavelength: radar wavelength in cm
- prf1: radar pulse repetition frequency in Hz
- prf2: alternate radar pulse repetition frequency in Hz (for a radar operating in dual-PRF mode)

Value

Nyquist velocity in m/s.

Examples

# at C-band (5.3 cm wavelength) and a PRF of 2000 Hz
nyquist_velocity(5.3, 2000)

# extended Nyquist velocity in a dual-PRF scheme
# using 2000 Hz and 1500 Hz PRFs:
nyquist_velocity(5.3, 2000, 1500)
Description

Plot a plan position indicator (PPI) generated with ppi using ggplot

Usage

```r
## S3 method for class 'ppi'
plot(
x, param, xlim, ylim, zlim = c(-20, 20), ratio = 1, na.value = "transparent",
...)
```

Arguments

- `x`: An object of class `ppi`.
- `param`: The scan parameter to plot, see details below.
- `xlim`: Range of x values to plot.
- `ylim`: Range of y values to plot.
- `zlim`: The range of parameter values to plot.
- `ratio`: Aspect ratio between x and y scale.
- `na.value`: ggplot argument setting the plot color of NA values.
- `...`: Arguments passed to low level `ggplot` function.

Details

Available scan parameters for plotting can be printed to screen by `summary(x)`. Commonly available parameters are:

- "DBZH", "DBZ" (Logged) reflectivity factor [dBZ]
- "TH", "T" (Logged) uncorrected reflectivity factor [dBZ]
- "VRADH", "VRAD" Radial velocity [m/s]. Radial velocities towards the radar are negative, while radial velocities away from the radar are positive.
- "RHOHV" Correlation coefficient [unitless]. Correlation between vertically polarized and horizontally polarized reflectivity factor.
- "PHIDP" Differential phase [degrees]
"ZDR" (Logged) differential reflectivity [dB]

The scan parameters are named according to the OPERA data information model (ODIM), see Table 16 in the ODIM specification.

Examples

```r
# load an example scan:
data(example_scan)

# print to screen the available scan parameters:
summary(example_scan)

# make ppi for the scan
ppi <- project_as_ppi(example_scan)

# plot the first scan parameter, which in this case is "VRADH":
plot(ppi)

# plot the reflectivity parameter:
plot(ppi, param = "DBZH")

# change the range of reflectivities to plot, from -10 to 10 dBZ:
plot(ppi, param = "DBZH", zlim = c(-10, 10))

# change the scale name and colour scheme, using viridis colors:
plot(ppi, param = "DBZH", zlim = c(-10, 10)) + viridis::scale_fill_viridis(name = "dBZ")
```

---

**plot.scan**  
Plot a scan (scan) in polar coordinates

**Description**

Plots a scan in polar coordinates. For plots in Cartesian coordinates, see `ppi`

**Usage**

```r
## S3 method for class 'scan'
plot(
  x,  
  param,  
  xlim = c(0, 1e+05),  
  ylim = c(0, 360),  
  zlim = c(-20, 20),  
  na.value = "transparent",  
  ...  
)
```
Arguments

- **x**: An object of class `scan`.
- **param**: The scan parameter to plot, see details below.
- **xlim**: Range of x (range, distance from radar) values to plot.
- **ylim**: Range of y (azimuth) values to plot.
- **zlim**: The range of parameter values to plot.
- **na.value**: `ggplot` argument setting the plot color of NA values
- **...**: Arguments passed to low level `ggplot` function.

Details

Available scan parameters for plotting can be printed to screen by `summary(x)`. Commonly available parameters are:

- "DBZH", "DBZ" (Logged) reflectivity factor [dBZ]
- "TH", "T" (Logged) uncorrected reflectivity factor [dBZ]
- "VRADH", "VRAD" Radial velocity [m/s]. Radial velocities towards the radar are negative, while radial velocities away from the radar are positive
- "RH0HV" Correlation coefficient [unitless]. Correlation between vertically polarized and horizontally polarized reflectivity factor
- "PHIDP" Differential phase [degrees]
- "ZDR" (Logged) differential reflectivity [dB]

The scan parameters are named according to the OPERA data information model (ODIM), see Table 16 in the ODIM specification.

Examples

```r
# load an example scan:
data(example_scan)

# print to screen the available scan parameters
summary(example_scan)

# make ppi for the scan
# plot the reflectivity param:
plot(example_scan, param = "DBZH")
## Not run:
# change the range of reflectivities to plot, from -10 to 10 dBZ:
plot(example_scan, param = "DBZH", zlim = c(-10, 10))

# change the scale name and colour scheme, using viridis colors:
plot(example_scan, param = "DBZH", zlim = c(-10, 10)) + viridis::scale_fill_viridis(name = "dBZ")

## End(Not run)
```
plot.vp

Description
Plot a vertical profile (vp)

Usage
## S3 method for class 'vp'
plot(
  x,
  quantity = "dens",
  xlab = expression("volume density [#/km]^3 * "])
  ylab = "height [km]",
  line_col = "red",
  line_lwd = 1,
  line.col = "red",
  line.lwd = 1,
  ...
)

Arguments
x A vp class object.
quantity Character string with the quantity to plot. See vp for list of available quantities.
  • Aerial density related: 'dens', 'eta', 'dbz', 'DBZH' for density, reflectivity, reflectivity factor and total reflectivity factor, respectively.
  • Ground speed related: 'ff', 'dd', for ground speed and direction, respectively.
xlab A title for the x axis.
ylab A title for the y axis.
line_col Color of the plotted curve.
line_lwd Line width of the plotted curve.
line.col Deprecated argument, use line_col instead.
line.lwd Deprecated argument, use line_lwd instead.
...
  Additional arguments to be passed to the low level plot plotting function.

Examples
# load example vp object:
data(example_vp)

# plot the animal density:
plot(example_vp, quantity = "dens")
# change the line color:
plot(example_vp, line_col = "blue")

# plot the ground speed:
plot(example_vp, quantity = "ff")

# plot the reflectivity factor of
# all scatterers (including precipitation):
plot(example_vp, quantity = "DBZH")

---

plot.vpi  
Plot an integrated profile (vpi)

Description

Plot an object of class vpi.

Usage

```r
## S3 method for class 'vpi'
plot(
  x, 
  quantity = "mtr", 
  xlab = "time", 
  ylab = "migration traffic rate [#/km/h]", 
  main = "MTR", 
  night_shade = TRUE, 
  elev = -0.268, 
  lat = NULL, 
  lon = NULL, 
  ylim = NULL, 
  nightshade = TRUE, 
  ...
)
```

Arguments

- **x**: 1 class object inheriting from class vpi, typically a call to `integrate_profile`.
- **quantity**: Character string with the quantity to plot, one of 'vid' (vertically integrated density), 'vir' (vertically integrated reflectivity), 'mtr' (migration traffic rate), 'rtr' (reflectivity traffic rate), 'mt' ((cumulative) migration traffic), 'rt' ((cumulative) reflectivity traffic), 'ff', (height-averaged speed) 'dd', (height-averaged direction) 'u', (height-averaged u-component of speed), 'v', (height-averaged v-component of speed).
- **xlab**: A title for the x-axis.
- **ylab**: A title for the y-axis.
main A title for the plot.
night_shade Logical, whether to plot night time shading.
elev Numeric, sun elevation to use for day/night transition, see sunrise.
lat (optional) Latitude in decimal degrees. Overrides the lat attribute of x.
lon (optional) Longitude in decimal degrees. Overrides the lon attribute of x.
ylim y-axis plot range, numeric atomic vector of length 2.
nightshade Deprecated argument, use night_shade instead.
... Additional arguments to be passed to the low level plot plotting function.

Details

The integrated profiles can be visualized in various related quantities, as specified by argument quantity:

"vid" Vertically Integrated Density, i.e. the aerial surface density of individuals. This quantity is dependent on the assumed radar cross section per individual (RCS)

"vir" Vertically Integrated Reflectivity. This quantity is independent of the value of individual’s radar cross section

"mtr" Migration Traffic Rate. This quantity is dependent on the assumed radar cross section (RCS)

"rtr" Reflectivity Traffic Rate. This quantity is independent on the assumed radar cross section (RCS)

"mt" Migration Traffic. This quantity is dependent on the assumed radar cross section (RCS)

"rt" Reflectivity Traffic. This quantity is independent on the assumed radar cross section (RCS)

ff Horizontal ground speed in m/s
dd Horizontal ground speed direction in degrees
u Ground speed component west to east in m/s
v Ground speed component north to south in m/s
height Mean flight height (height weighted by reflectivity eta) in m above sea level

The height-averaged speed quantities (ff,dd,u,v) and height are weighted averages by reflectivity eta.

Examples

# vertically integrate a vpts object:
vpi <- integrate_profile(example_vpts)
# plot the migration traffic rates
plot(vpi)
# plot the vertically integrated densities, without night shading:
plot(vpi, quantity = "vid", night_shade = FALSE)
plot.vpts  

Plot a time series of vertical profiles (vpts)

Description

Plot a time series of vertical profiles of class vpts.

Usage

```r
## S3 method for class 'vpts'
plot(
  x,  
  xlab = "time",  
  ylab = "height [m]",  
  quantity = "dens",  
  log = TRUE,  
  barbs = TRUE,  
  barbs_height = 10,  
  barbs_time = 20,  
  barbs_dens_min = 5,  
  zlim,  
  legend_ticks,  
  legend_ticks,  
  main,  
  barbs.h = 10,  
  barbs.t = 20,  
  barbs.dens = 5,  
  ...  
)
```

Arguments

- **x**: A vp class object inheriting from class vpts.
- **xlab**: A title for the x-axis.
- **ylab**: A title for the y-axis.
- **quantity**: Character string with the quantity to plot, one of 'dens','eta','dbz','DBZH' for density, reflectivity, reflectivity factor and total reflectivity factor, respectively.
- **log**: Logical, whether to display quantity data on a logarithmic scale.
- **barbs**: Logical, whether to overlay speed barbs.
- **barbs_height**: Integer, number of barbs to plot in altitudinal dimension.
- **barbs_time**: Integer, number of barbs to plot in temporal dimension.
- **barbs_dens_min**: Numeric, lower threshold in aerial density of individuals for plotting speed barbs in individuals/km^3.
- **zlim**: Optional numerical atomic vector of length 2, specifying the range of quantity values to plot.
legend_ticks  Numeric atomic vector specifying the ticks on the color bar.
legend_ticks  Deprecated argument, use legend_ticks instead.
main  A title for the plot.
barbs.h  Deprecated argument, use barbs_height instead.
barbs.t  Deprecated argument, use barbs_time instead.
barbs.dens  Deprecated argument, use barbs_dens_min instead.
...  Additional arguments to be passed to the low level image plotting function.

Details
Profile can be visualized in three related quantities, as specified by argument quantity:

"dens"  the aerial density of individuals. This quantity is dependent on the assumed radar cross section (RCS) in the x$attributes$how$rcs_bird attribute
"eta"  reflectivity. This quantity is independent of the value of the rcs_bird attribute
"dbz"  reflectivity factor. This quantity is independent of the value of the rcs_bird attribute, and corresponds to the dBZ scale commonly used in weather radar meteorology. Bioscatter by birds tends to occur at much higher reflectivity factors at S-band than at C-band
"DBZH"  total reflectivity factor. This quantity equals the reflectivity factor of all scatterers (biological and meteorological scattering combined)

In the speed barbs, each half flag represents 2.5 m/s, each full flag 5 m/s, each pennant (triangle) 25 m/s

Examples
```r
# locate example file:
ts <- example_vpts
# plot density of individuals for the first 500 time steps, in the altitude
# layer 0-3000 m.
plot(ts[1:500], ylim = c(0, 3000))
# plot total reflectivity factor (rain, birds, insects together):
plot(ts[1:500], ylim = c(0, 3000), quantity = "DBZH")
```

---

project_as_ppi  Project a scan (scan) or parameter (param) to a plan position indicator (ppi)

Description
Make a plan position indicator (ppi)
Usage

project_as_ppi(
  x,
  grid_size = 500,
  range_max = 50000,
  project = TRUE,
  ylim = NULL,
  xlim = NULL,
  raster = NA,
  k = 4/3,
  re = 6378,
  rp = 6357
)

## S3 method for class 'param'
project_as_ppi(
  x,
  grid_size = 500,
  range_max = 50000,
  project = TRUE,
  ylim = NULL,
  xlim = NULL,
  raster = NA,
  k = 4/3,
  re = 6378,
  rp = 6357
)

## S3 method for class 'scan'
project_as_ppi(
  x,
  grid_size = 500,
  range_max = 50000,
  project = TRUE,
  ylim = NULL,
  xlim = NULL,
  raster = NA,
  k = 4/3,
  re = 6378,
  rp = 6357
)

Arguments

x An object of class param or scan.
grid_size Cartesian grid size in m.
range_max Maximum range in m.
project_as_ppi

project Whether to vertically project onto earth’s surface.
ylim The range of latitudes to include.
xlim The range of longitudes to include.
raster (optional) RasterLayer with a CRS. When specified this raster topology is used for the output, and grid_size, range_max, xlim, ylim are ignored.
k Standard refraction coefficient.
re Earth equatorial radius in km.
rp Earth polar radius in km.

Details

The returned PPI is in Azimuthal Equidistant Projection.

Value

An object of class 'ppi'.

Methods (by class)

- param: Project as ppi for a single scan parameter.
- scan: Project multiple ppi’s for all scan parameters in a scan

Examples

```r
# load a polar scan example object:
data(example_scan)
example_scan

# plot the scan:
plot(example_scan)

# make PPIs for all scan parameters in the scan:
ppi <- project_as_ppi(example_scan)

# print summary info for the ppi:
ppi

# plot the ppi:
plot(ppi)

# extract the DBZH scan parameter of the volume to a new
# object 'param':
param <- get_param(example_scan, "VRADH")

# make a ppi for the new 'param' object:
ppi <- project_as_ppi(param)

# print summary info for this ppi:
ppi
```
# plot the ppi:
plot(ppi)

---

### rcs

*Get radar cross section*

#### Description

Gives the currently assumed radar cross section in cm$^2$.

#### Usage

```r
rcs(x)
```

- **## S3 method for class 'vp'
  
  `rcs(x)`

- **## S3 method for class 'list'
  
  `rcs(x)`

- **## S3 method for class 'vpts'
  
  `rcs(x)`

- **## S3 method for class 'vpi'
  
  `rcs(x)`

#### Arguments

- **x**
  
  A vp, list of vp or vpts object.

#### Details

See also `rcs<-` for changing or setting the radar cross section of an object.

#### Value

A radar cross section in cm$^2$

#### Methods (by class)

- **vp**: radar cross section of a vertical profile
- **list**: radar cross sections for a list of vertical profiles
- **vpts**: radar cross section of a time series of vertical profile
- **vpi**: radar cross section of a time series of vertically integrated vertical profile(s)
Examples

# load example data:
data(example_vp)
data(example_vpts)

# retrieve RCS for a single vertical profile:
rcs(example_vp)

# retrieve RCS for a vertical profile time series:
rcs(example_vpts)

# change or set RCS for a single vertical profile:
rcs(example_vp) <- 11

# change or set RCS for a vertical profile time series:
rcs(example_vpts) <- 11

rcs<- \textit{Set radar cross section}

Description

Sets the assumed radar cross section in cm\(^2\). This method also updates the migration densities in \(x\$data\$dens\).

Usage

rcs(x) <- value

## S3 replacement method for class 'vp'
rcs(x) <- value

## S3 replacement method for class 'list'
rcs(x) <- value

## S3 replacement method for class 'vpts'
rcs(x) <- value

## S3 replacement method for class 'vpi'
rcs(x) <- value

Arguments

x \hspace{1cm} \text{a vp, list of vp or vpts object}
value \hspace{1cm} \text{the cross section value to assign}

Details

See also \texttt{rcs} for retrieving the radar cross section of an object.
Examples

# load example data
data(example_vp)
data(example_vpts)

# change or set RCS for a single vertical profile:
rcs(example_vp) <- 11

# change or set RCS for a vertical profile time series:
rcs(example_vpts) <- 11

# retrieve RCS for a single vertical profile:
rcs(example_vp)

# retrieve RCS for a vertical profile time series:
rcs(example_vpts)

---

read_cajun

Read a vertical profile (vp) from UMASS Cajun text file

Description

Read a vertical profile (vp) from UMASS Cajun text file

Usage

read_cajun(file, rcs = 11, wavelength = "S")

Arguments

- **file**: A text file containing the standard output (stdout) generated by UMASS Cajun pipeline
- **rcs**: numeric. Radar cross section per bird in cm\(^2\).
- **wavelength**: Radar wavelength in cm, or one of 'C' or 'S' for C-band and S-band radar, respectively, in which case C-band wavelength is assumed to be 5.3 cm and S-band wavelength 10.6 cm

Value

An object inheriting from class vp, see vp for details.
read_pvolfile

Description

Read a polar volume \((pvol)\) from file

Usage

\[
\text{read_pvolfile(}
\text{file,}
\text{param = c("DBZH", "DBZ", "VRADH", "VRAD", "TH", "T", "RHQHV", "ZDR", "PHIDP", "CELL",}
\text{ "BIOLOGY", "WEATHER", "BACKGROUND"),}
\text{sort = TRUE,}
\text{lat,}
\text{lon,}
\text{height,}
\text{elev_min = 0,}
\text{elev_max = 90,}
\text{verbose = TRUE,}
\text{mount = dirname(file)}
\text{)}
\]

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>A string containing the path to a polar volume file</td>
</tr>
</tbody>
</table>
| param    | An atomic vector of character strings, containing the names of scan parameters to read. To read all scan parameters use ‘all’.
| sort     | A logical value, when TRUE sort scans ascending by elevation. |
| lat      | Latitude in decimal degrees of the radar position. If not specified, value stored in file is used. If specified, value stored in file is overwritten. |
| lon      | Longitude in decimal degrees of the radar position. If not specified, value stored in file is used. If specified, value stored in file is overwritten. |
| height   | Height of the center of the antenna in meters above sea level. If not specified, value stored in file is used. If specified, value stored in file is overwritten. |
| elev_min | Minimum scan elevation to read in degrees. |
| elev_max | Maximum scan elevation to read in degrees. |
| verbose  | A logical value, whether to print messages (TRUE) to console. |
| mount    | A character string with the mount point (a directory path) for the Docker container. |
Details
Scan parameters are named according to the OPERA data information model (ODIM), see Table 16 in the ODIM specification. Commonly available parameters are:

"DBZH", "DBZ" (Logged) reflectivity factor [dBZ]
"TH", "T" (Logged) uncorrected reflectivity factor [dBZ]
"VRADH", "VRAD" Radial velocity [m/s]. Radial velocities towards the radar are negative, while radial velocities away from the radar are positive
"RHOHV" Correlation coefficient [unitless]. Correlation between vertically polarized and horizontally polarized reflectivity factor
"PHIDP" Differential phase [degrees]
"ZDR" (Logged) differential reflectivity [dB]

Value
An object of class *pvol*, which is a list containing polar scans, i.e. objects of class *scan*

Examples

```r
# locate example volume file:
pvolfile <- system.file("extdata", "volume.h5", package = "bioRad")

# print the local path of the volume file:
pvolfile

# load the file:
example_pvol <- read_pvolfile(pvolfile)

# print summary info for the loaded polar volume:
exmple_pvol

# print summary info for the scans in the polar volume:
exmple_pvol$scans

# copy the first scan to a new object 'scan'
scan <- example_pvol$scans[[1]]

# print summary info for the new object:
scan
```

**read_vpfiles**

*Read a vertical profile (vp) or a list of vertical profiles (vp) from files*

Description

Read a vertical profile (vp) or a list of vertical profiles (vp) from files
read_vpts

**Usage**

```r
read_vpfiles(files)
```

**Arguments**

- `files` A character vector containing the file names of vertical profiles in ODIM HDF5 format generated by `calculate_vp`.

**Value**

A single `vp` object or a list of `vp` objects.

**Examples**

```r
# locate example profile file:
vpfile <- system.file("extdata", "profile.h5", package = "bioRad")

# print the local path of the profile file:
vpfile

# load the file:
read_vpfiles(vpfile)

# load multiple files at once:
## Not run:
# read_vpfiles(c("my/path/profile1.h5", "my/path/profile2.h5", ...))

## End(Not run)
```

---

**read_vpts**  
Read a time series of vertical profiles (vpts) from file

**Description**

Read a time series of vertical profiles (vpts) from file

**Usage**

```r
read_vpts(file, radar, lat, lon, height, wavelength = "C")
```

**Arguments**

- `file` A text file containing the standard output (stdout) generated by vol2bird (or the package function `calculate_vp`).
- `radar` A string containing a radar identifier.
- `lat` numeric. Latitude of the radar in decimal degrees.
- `lon` numeric. Longitude of the radar in decimal degrees.
height numeric. Height above sea level of the radar antenna in meters.

wavelength Radar wavelength in cm, or one of 'C' or 'S' for C-band and S-band radar, respectively, in which case C-band wavelength is assumed to be 5.3 cm and S-band wavelength 10.6 cm

Value
An object inheriting from class vpts, see vpts for details.

Examples

# locate example file:
vptsfile <- system.file("extdata", "example_vpts.txt", package = "bioRad")
# load time series:
ts <- read_vpts(vptsfile, radar = "KBGM", wavelength = "S")
ts

regularize_vpts Regularize a time series of vertical profiles (vpts) on a regular time grid

Description
Projects objects of class vpts on a regular time grid

Usage

regularize_vpts(
  ts,
  interval = "auto",
  date_min = ts$daterange[1],
  date_max = ts$daterange[2],
  units = "secs",
  fill = FALSE,
  verbose = TRUE
)

Arguments

  ts An object inheriting from class vpts, see vpts for details.
  interval Time interval grid to project on. When 'auto' the median interval in the time series is used.
  date_min Start time of the projected time series, as a POSIXct object. Taken from ts when 'auto'.
  date_max End time of the projected time series, as a POSIXct object. Taken from ts when 'auto'.
units | Optional units of interval, one of 'secs', 'mins', 'hours', 'days', 'weeks'. Defaults to 'mins'.
fill | Logical, whether to fill missing timesteps with the values of the closest neighboring profile.
verbose | Logical, when TRUE prints text to console.

Details

Irregular time series of profiles are typically aligned on a regular time grid with the expected time interval at which a radar provides data. Empty profiles with only missing data values will be inserted at time stamps of the regular time grid that have no matching profile in the irregular time series.

In plots of regular time series (see `plot.vpts`) temporal gaps of missing profiles (e.g. due to radar down time) become visible. In irregular time series data points in the plot are carried through until the time series continues, and temporal data gaps are filled up visually.

Value

An object of class vpts with regular time steps.

Examples

```r
# start form example vpts object:
data(example_vpts)
ts <- example_vpts

# regularize the time series on a 5 minute interval grid
tsRegular <- regularize_vpts(ts, interval = 300)
```

scan_to_raster | convert a polar scan into a raster

Description

convert an object of class `scan` into a raster of class `RasterBrick`

Usage

```r
scan_to_raster(
  scan,
  nx = 100,
  ny = 100,
  xlim,
  ylim,
  res = NA,
  param,
  raster = NA,
  lat,
```
lon,  
crs = NA,  
k = 4/3,  
re = 6378,  
rp = 6357
)

Arguments

scan             a scan (sweep) of class scan
nx               number of raster pixels in the x (longitude) dimension
ny               number of raster pixels in the y (latitude) dimension
xlim             x (longitude) range
ylim             y (latitude) range
res              numeric vector of length 1 or 2 to set the resolution of the raster (see res). If this
                 argument is used, arguments nx and ny are ignored. Unit is identical to xlim
                 and ylim.
param            scan parameters to include. If NA include all scan parameters. Reducing the
                 number of scan parameters speeds up evaluation.
raster           (optional) RasterLayer with a CRS. When specified this raster topology is used
                 for the output, and nx, ny, res arguments are ignored.
lat              Geodetic latitude of the radar in degrees. If missing taken from scan.
lon              Geodetic longitude of the radar in degrees. If missing taken from scan.
crs              character or object of class CRS. PROJ.4 type description of a Coordinate Ref-  
                 erence System (map projection). When 'NA' (default), an azimuthal equidistant
                 projection with origin at the radar location is used. To use a WSG84 (lat,lon)
                 projection, use crs="+proj=longlat +datum=WGS84"
k                Standard refraction coefficient.
re               Earth equatorial radius in km.
rp               Earth polar radius in km.

Details

uses scan_to_spatial to georeference the scan’s pixels. If multiple scan pixels fall within the same
raster pixel, the last added pixel is given (see rasterize for details).

Value

a RasterBrick

Examples

# default projects full extent on 100x100 pixel raster:
scan_to_raster(example_scan)

# crop the scan and project at a resolution of 0.1 degree:
scan_to_spatial

scan_to_raster(example_scan, ylim = c(55, 57), xlim = c(12, 13), res = .1)

# using a template raster
template_raster <- raster::raster(raster::extent(12, 13, 56, 58), crs = sp::CRS("+proj=longlat"))
scan_to_raster(example_scan, raster = template_raster)

---

scan_to_spatial convert a polar scan into a spatial object.

Description

Georeferences the pixels of a scan into a SpatialPointsDataFrame object.

Usage

scan_to_spatial(scan, lat, lon, k = 4/3, re = 6378, rp = 6357)

Arguments

- **scan**: a scan (sweep) of class scan
- **lat**: Geodetic latitude of the radar in degrees. If missing taken from scan.
- **lon**: Geodetic longitude of the radar in degrees. If missing taken from scan.
- **k**: Standard refraction coefficient.
- **re**: Earth equatorial radius in km.
- **rp**: Earth polar radius in km.

Details

Beam altitude accounts for the curvature of the earth, using `beam_height`. Distance from the radar over the earth’s surface is calculated using `beam_distance`.

Value

- a SpatialPointsDataFrame

Examples

# load example scan:
data(example_scan)

# convert to a SpatialPointsDataFrame:
scan_to_spatial(example_scan)
**sd_vvp_threshold**

Get threshold of the VVP-retrieved radial velocity standard deviation

---

**Description**

Gives the current threshold in VVP-retrieved radial velocity standard deviation in m/s.

**Usage**

```r
sd_vvp_threshold(x)
```

```r
def sd_vvp_threshold(x)
  ## S3 method for class 'vp'
  sd_vvp_threshold(x)

  ## S3 method for class 'list'
  sd_vvp_threshold(x)

  ## S3 method for class 'vpts'
  sd_vvp_threshold(x)
```

**Arguments**

- `x` A `vp`, list of `vp` or `vpts` object.

**Details**

See also `sd_vvp_threshold<-` for setting an objects radial velocity standard deviation.

**Value**

threshold for `sd_vvp` in m/s.

**Methods (by class)**

- `vp`: threshold in VVP-retrieved radial velocity standard deviation of a vertical profile
- `list`: threshold in VVP-retrieved radial velocity standard deviation of a list of vertical profiles
- `vpts`: threshold in VVP-retrieved radial velocity standard deviation of a time series of vertical profiles

**Examples**

```r
# load example data:
data(example_vp)
data(example_vpts)

# retrieve threshold for a single vertical profile:
sd_vvp_threshold(example_vp)
```
Set threshold of the VVP-retrieved radial velocity standard deviation

Description

Sets the threshold for sd_vvp. Altitude layers with sd_vvp below this threshold are assumed to have an aerial density of zero individuals. This method updates the migration densities in x$data$dens

Usage

sd_vvp_threshold(x) <- value

## S3 replacement method for class 'vp'
sd_vvp_threshold(x) <- value

## S3 replacement method for class 'list'
sd_vvp_threshold(x) <- value

## S3 replacement method for class 'vpts'
sd_vvp_threshold(x) <- value

Arguments

x a vp, list of vp or vpts object

value the value to assign

Details

See also sd_vvp_threshold for retrieving an objects radial velocity standard deviation.

Examples

# load example data:
data(example_vp)
data(example_vpts)

# change or set the threshold for a single vertical profile:
sd_vvp_threshold(example_vp) <- 2
# change or set the threshold for a vertical profile time series:
sd_vvp_threshold(example_vp) <- 2

# retrieve threshold for a single vertical profile:
sd_vvp_threshold(example_vp)

# retrieve threshold for a vertical profile time series:
sd_vvp_threshold(example_vpts)

---

select_vpfiles

Select vertical profile (vp) files from computer

Description

Create a list of vertical profile (vp) files from a local directory that match a specific date and radar range. Files are selected based on their file name (not directory structure), which should be of format radar_vp_yyyymmdd*.*, such as bewid_vp_20171123T1900Z_0x5.h5.

Usage

```r
select_vpfiles(
  date_min = NULL,
  date_max = NULL,
  radars = NULL,
  directory = "."
)
```

Arguments

- `date_min` character. YYYY-MM-DD start date of file selection.
- `date_max` character. YYYY-MM-DD end date of file selection.
- `radars` character (vector). 5-letter country/radar code(s) (e.g. `bejab`) of radars to include in file selection.
- `directory` character. Path to local directory where files should be looked for.

Value

Character vector of file paths that comply to the given date and radar range.

Examples

```r
select_vpfiles(
  date_min = "2016-10-03",
  date_max = "2016-10-05",
  radars = "bejab",
  directory = "my_data"
)```
Class `param` for a parameter of a scan of a polar volume, and its associated R base functions.

## Usage

```r
## S3 method for class 'param'
summary(object, ...)

is.param(x)
```

### Arguments

- `object` Object of class `param`.
- `...` Additional arguments affecting the summary produced.
- `x` Object of class `param`.

### Details

An object of class `scan` is a simple matrix, with the following specific attributes:

- `radar` character string with the radar identifier
- `datetime` nominal time of the volume to which this scan belongs [UTC]
- `lat` latitude of the radar [decimal degrees]
- `lon` longitude of the radar [decimal degrees]
- `height` height of the radar antenna [meters above sea level]
- `get_elevation_angles` radar beam elevation [degrees]
- `param` string with the name of the polar scan parameter

Scan parameters are named according to the OPERA data information model (ODIM), see Table 16 in the ODIM specification. Commonly available parameters are:

- "DBZH", "DBZ" (Logged) reflectivity factor [dBZ]
- "VRADH", "VRAD" Radial velocity [m/s]. Radial velocities towards the radar are negative, while radial velocities away from the radar are positive
- "RHOHV" Correlation coefficient [unitless]. Correlation between vertically polarized and horizontally polarized reflectivity factor
- "PHIDP" Differential phase [degrees]
- "ZDR" (Logged) differential reflectivity [dB]
Value

for is.scan: TRUE if its argument is of class "param"

Examples

# load example scan object
data(example_scan)

# extract the DBZH scan parameter:
param <- get_param(example_scan, "DBZH")

# verify this is an object of class param:
is.param(param)

# print summary info for this scan parameter:
param

summary.ppi

Class ppi: a plan position indicator

Description

Class ppi for a plan position indicator, and its associated R base functions.

Usage

## S3 method for class 'ppi'
summary(object, ...)

is.ppi(x)

## S3 method for class 'ppi'
dim(x)

Arguments

object Object of class ppi.
...
Additional arguments affecting the summary produced.
x Object of class ppi.

Details

ppi objects are generated from elevation scans (scan objects) with project_as_ppi or from polar volumes with integrate_to_ppi, producing projections of the radar data onto the earth’s surface.

An object of class ppi is a list containing:

data an object of class SpatialGridDataFrame containing the georeferenced data. Commonly available parameters are:
"DBZH", "DBZ" (Logged) reflectivity factor [dBZ]
"TH", "T" (Logged) uncorrected reflectivity factor [dBZ]
"VRADH", "VRAD" Radial velocity [m/s]. Radial velocities towards the radar are negative, while radial velocities away from the radar are positive
"RHOHV" Correlation coefficient [unitless]. Correlation between vertically polarized and horizontally polarized reflectivity factor
"PHIDP" Differential phase [degrees]
"ZDR" (Logged) differential reflectivity [dB]

geo geographic data, a list with:
  lat  latitude of the radar [decimal degrees]
  lon  longitude of the radar [decimal degrees]
  height height of the radar antenna [meters above sea level]
  elangle  radar beam elevation [degrees]
  rscale  range bin size [m]
  ascale  azimuth bin size [deg]

The geo element of a 'scan' object is a copy of the geo element of its parent scan or scan parameter.

Value

For is.ppi: TRUE if its argument is of class ppi.

For dim.ppi: dimensions of the ppi.

Examples

# load example scan object
data(example_scan)

# calculate ppi object
e.example_ppi <- project_as_ppi(example_scan)

# print summary info:
e.example_ppi

# verify example_ppi is a ppi object:
is.ppi(example_ppi)

# ppi object dimensions:
dim(example_ppi)
**Summary.pvol**

*Class pvol: a polar volume*

**Description**

Class `pvol` for a polar volume, and its associated R base functions.

**Usage**

```r
## S3 method for class 'pvol'
summary(object, ...)

is.pvol(x)
```

**Arguments**

- `object` Object of class `pvol`.
- `...` Additional arguments affecting the summary produced.
- `x` Object of class `pvol`.

**Details**

An object of class `pvol` is a list containing:

- `radar` character string with the radar identifier
- `datetime` nominal time of the volume [UTC]
- `scans` a list with scan objects of class `scan`
- `attributes` list with the volume’s `what`, `where` and `how` attributes
- `geo` geographic data, a list with:
  - `lat` latitude of the radar [decimal degrees]
  - `lon` longitude of the radar [decimal degrees]
  - `height` height of the radar antenna [meters above sea level]

**Value**

For `is.pvol`: `TRUE` if its argument is of class `pvol`

**Examples**

```r
# locate example volume file:
pvolfile <- system.file("extdata", "volume.h5", package = "bioRad")

# print the local path of the volume file:
pvolfile

# load the file:
```
example_pvol <- read_pvolfile(pvolfile)

# print summary info for the loaded polar volume:
example_pvol

# verify that this is a pvol object:
is.pvol(example_pvol)

# print summary info for the scans in the polar volume:
example_pvol$scans

# copy the first scan to a new object 'scan':
scan <- example_pvol$scans[[1]]

# print summary info for the scan:
scan
is.pvol("this is not a polar volume but a string") # > FALSE

---

### summary.scan

**Class:** a scan of a polar volume

**Description**

Class scan for a scan of a polar volume, and its associated R base functions.

**Usage**

```r
## S3 method for class 'scan'
summary(object, ...)  
is.scan(x)
```

**Arguments**

- `object` Object of class `scan`
- `...` Additional arguments affecting the summary produced.
- `x` Object of class `scan`

**Details**

A object of class `scan` is a list containing:

- `radar` character string with the radar identifier
- `datetime` nominal time of the volume to which this scan belongs [UTC]
- `params` a list with scan parameters
attributes list with the scan’s \texttt{what, where and how} attributes

geo geographic data, a list with:
\begin{itemize}
  \item \texttt{lat} latitude of the radar [decimal degrees]
  \item \texttt{lon} longitude of the radar [decimal degrees]
  \item \texttt{height} height of the radar antenna [meters above sea level]
  \item \texttt{elangle} radar beam elevation [degrees]
  \item \texttt{rscale} range bin size [m]
  \item \texttt{ascale} azimuth bin size [deg]
\end{itemize}

The geo element of a scan object is a copy of the geo element of its parent polar volume of class pvol.

\section*{Value}

For \texttt{is.scan}: \texttt{TRUE} if its argument is of class \texttt{scan}.

For \texttt{dim.scan}: dimensions of the scan.

\section*{Examples}

\begin{verbatim}
# load example scan object
data(example_scan)

# verify this object is of class scan:
is.scan(example_scan)

# print the scan parameters contained in the scan:
example_scan$params

# extract the VRADH scan parameter:
param <- get_param(example_scan, "VRADH")

# print summary info for this scan parameter:
param
is.scan("this is not a polar scan but a string") # > FALSE
\end{verbatim}
Usage

```r
## S3 method for class 'vp'
summary(object, ...)
```

```r
is.vp(x)
```

```r
## S3 method for class 'vp'
dim(x)
```

Arguments

- `object`: An object of class `vp`.
- `...`: Additional arguments affecting the summary produced.
- `x`: An object of class `vp`.

Details

An object of class `vp` contains a vertical profile. A vertical profile contains a collection of quantities, with each quantity having values at different altitude layers above the earth’s surface, typically equally spaced altitudinal layers.

Data contained in this class object should be accessed with the `get_quantity` function. Information stored under `attributes` (see below) can be accessed directly.

A `vp` object is a list containing:

- `radar`: the radar identifier
- `datetime`: the nominal time of the profile
- `data`: the profile data, a list containing:
  - `height`: height above mean sea level [m]. Alt. bin from height to height+interval)
  - `u`: speed component west to east [m/s]
  - `v`: speed component north to south [m/s]
  - `w`: vertical speed (unreliable!) [m/s]
  - `ff`: horizontal speed [m/s]
  - `dd`: direction [degrees, clockwise from north]
  - `sd_vvp`: VVP radial velocity standard deviation [m/s]
  - `gap`: Angular data gap detected [T/F]
  - `dbz`: Animal reflectivity factor [dBZ]
  - `eta`: Animal reflectivity [cm^2/km^3]
  - `dens`: Animal density [animals/km^3]
  - `DBZH`: Total reflectivity factor (bio+meteo scattering) [dBZ]
  - `n`: number of points VVP velocity analysis (u,v,w,ff,dd)
  - `n_all`: number of points VVP st.dev. estimate (sd_vvp)
  - `n_dbz`: number of points density estimate (dbz,eta,dens)
  - `n_dbz_all`: number of points total reflectivity estimate (DBZH)
- `attributes`: list with the profile’s `\what`, `\where` and `\how` attributes
Conventions:
• NA Maps to 'nodata' in the ODIM convention: value to denote areas void of data (never radiated)
• NaN Maps to 'undetect' in the ODIM convention: denote areas below the measurement detection threshold (radiated but nothing detected). The value is also used when there are too few datapoints to calculate a quantity.
• 0 Maps to 0 in the ODIM convention: denote areas where the quantity has a measured value of zero (radiated and value zero detected or inferred).

It depends on a radar’s detection threshold or signal to noise ratio whether it safe to assume an 'undetect' is equivalent to zero. When dealing with close range data only (within 35 km), it is typically safe to assume aerial densities (dens) and reflectivities (eta) are in fact zero in case of undetects.

Value
For is.vp: TRUE if its argument is of class vp.
For dim.vp: dimensions of the profile data.

Examples
# load example vp object
data(example_vp)
example_vp

# check that the object is a vp object:
is.vp(example_vp)

# dimensions of the vp object:
dim(example_vp)

summary.vpts  Class vpts: a time series of vertical profiles

Description
Class vpts for a time series of vertical profiles, and its associated R base functions.

Usage
## S3 method for class 'vpts'
summary(object, ...)

is.vpts(x)

## S3 method for class 'vpts'
dim(x)
Arguments

object  An object of class vpts.
...  Additional arguments affecting the summary produced.
x  An object of class vpts.

Details

An object of class vpts contains time-ordered profiles of a single radar station.
The time series can be regular or irregular, indicated by the regular field.
In a regular vpts object the profiles are equally spaced in time. In an irregular vpts object the time
steps between profiles are of unequal length.
Irregular time series can be projected onto a regular time grid using the regularize_vpts function.
By contrast, vp objects can be concatenated in a list to combine profiles without time ordering, and
profiles of multiple radars.
Data contained in this class object should be accessed with the get_quantity function. Information
stored under attributes (see below) can be accessed directly.
An object of class vpts is a list containing
radar  string containing the radar identifier
datetime the N nominal times of the profiles (named dates in bioRad versions < 0.4.0)
height the M heights of the layers in the profile
daterange the minimum and maximum nominal time of the profiles in the list
timesteps time differences between the profiles. Element i gives the time difference between
profile i and i+1
data  list of N by M matrices containing the vertical profiles for each quantity. For a description of
available quantities, see the data element of the vp class in read_vpfiles
attributes  profile attributes, copied from the first profile contained in x
regular  logical indicating whether the time series is regular or not

Value

For is.vpts: TRUE if its argument is of class vpts.
For dim.vpts: dimensions of the time series.

Examples

# load example vertical profile time series:
data(example_vpts)
example_vpts

# verify this is a vpts object
is.vpts(example_vpts)

# dimensions of the vpts object
dim(example_vpts)
sunrise_sunset

Calculate sunrise or sunset for a time and place

Description

Calculate sunrise or sunset for a time and place

Usage

```
sunrise(date, lon, lat, elev = -0.268, tz = "UTC")

sunset(date, lon, lat, elev = -0.268, tz = "UTC")
```

Arguments

- `date`: Date inheriting from class POSIXt or a string interpretable by `as.Date`.
- `lon`: Longitude in decimal degrees.
- `lat`: Latitude in decimal degrees.
- `elev`: Sun elevation in degrees.
- `tz`: Output time zone. Ignored if `date` has an associated time zone already.

Details

The angular diameter of the sun is about 0.536 degrees, therefore the moment of sunrise/sunset corresponds to half that elevation at -0.268 degrees. This is a convenience function mapping to `crepuscule`. Approximate astronomical formula are used, therefore the moment of sunrise / sunset may be off by a few minutes.

Value

The moment of sunrise or sunset in UTC time.

Examples

```
# sunrise in the Netherlands
sunrise("2016-01-01", 5, 53)

# sunset in the Netherlands
sunset("2016-01-01", 5, 53)

# civil twilight in Ithaca, NY, today
sunrise(Sys.time(), -76.5, 42.4, elev = -6)
```
**update_docker**

*Update Docker image from Docker hub*

**Description**
Pulls and installs the latest Docker image used by bioRad from Docker hub

**Usage**

```
update_docker(mistnet = FALSE)
```

**Arguments**

- **mistnet**
  logical. When True, installs MistNet segmentation model, downloading an additional 1GB Docker image (see *apply_mistnet* for details).

**Details**
This command pulls the latest vol2bird Docker image from Docker hub. Run this command to ensure all Docker functionality (e.g. the *calculate_vp* function) runs at the latest available version.

To install the MistNet segmentation model into bioRad, run `update_docker(mistnet = TRUE)`

**Value**
the POSIXct creation date of the installed Docker image

**Examples**

```r
## Not run:
# update the vol2bird docker image:
update_docker()

## End(Not run)
```

**vol2bird_version**

*Check version of the vol2bird algorithm used by bioRad*

**Description**
Checks that Docker daemon is running correctly on the local system and returns the version of the installed vol2bird algorithm in the Docker container.

**Usage**

```
vol2bird_version(local_install)
```
Arguments

local_install (optional) String with path to local vol2bird installation, see calculate_vp for details.

Details

when argument local_install is specified with a path to a local executable of vol2bird, the function will return the version of this local installation.

Value

an object of class numeric_version, NA if docker system command not available, NaN if Docker daemon not running, NULL if adokter/vol2bird docker image not available

Examples

```r
## Not run:
# check installed vol2bird version:
vol2bird_version()
## End(Not run)
```

---

`.ppi` *Subset a plan position indicator (ppi)*

Description

Select quantities by index from a ppi

Usage

```r
## S3 method for class 'ppi'
x[i]
```

Arguments

- `x` An object of class param or scan.
- `i` Indices specifying elements to extract.

Examples

```r
# make a ppi:
my_ppi <- project_as_ppi(example_scan)

# this ppi contains 5 quantities (VRADH DBZH ZDR RHOHV PHIDP):
my_ppi

# This ppi only contains the first quantity (VRADH):
```
my_ppi[1]

# This ppi contains the first three quantities (VRADH, DBZH, ZDR):
my_ppi[1:3]

Subset a time series of vertical profiles (vpts)

Description
Select a vertical profile (vp) or a time series of vertical profiles (vpts) by index from a vpts

Usage

## S3 method for class 'vpts'
x[i]

Arguments

x  Object of class vpts.
i  Indices specifying elements to extract.

Examples

# we start with the example vertical profile time series:
data(example_vpts)
example_vpts

# extract the 10th profile in the time series (returns a vp object)
example_vpts[10]

# extract the 20th to 100th profile from the time series (returns a vpts object)
example_vpts[20:100]
Index

*Topic datasets
  example_scan, 31
  example_vp, 32
  example_vpts, 32
  [, .ppi, 86
  [, .vpts, 87
  apply_mistnet, 3, 85
  as.data.frame.vp, 5
  as.data.frame.vpts, 7
  as.Date, 84
  as.POSIXct, 24
  beam_distance, 9, 71
  beam_height, 9, 10, 11, 15, 71
  beam_profile, 11, 12, 13, 44, 45
  beam_profile_overlap, 12
  beam_range, 14, 44
  beam_width, 11, 15, 21
  bind_into_vpts, 16
  c.vp, 17
  calculate_param, 18
  calculate_vp, 19, 32, 67, 85, 86
  check_docker, 23
  check_night, 23
  composite_ppi, 25
  crepuscule, 84
  dbz_to_eta, 27, 44
  dim.ppi (summary.ppi), 76
  dim.scan (summary.scan), 79
  dim.vp (summary.vp), 80
  dim.vpts (summary.vpts), 82
  download_basemap, 28, 48
  download_vpfiles, 29
  eta_to_dbz, 30
  example_scan, 31
  example_vp, 32
  example_vpts, 32
  filter_vpts, 33
  get_elevation_angles, 34
  get_map, 28
  get_odim_object_type, 35
  get_param, 36
  get_quantity, 36, 81, 83
  get_quantity.vp, 6, 8
  get_scan, 38
  ggmap, 47, 48
  ggplot, 52, 54
  image, 59
  integrate_profile, 39, 44, 56
  integrate_to_ppi, 13, 42, 76
  is.param (summary.param), 75
  is.ppi (summary.ppi), 76
  is.pvol (summary.pvol), 78
  is.pvolfile, 46
  is.scan (summary.scan), 79
  is.vp (summary.vp), 80
  is.vpfile, 47
  is.vpts (summary.vpts), 82
  map, 47
  nexrad_to_odim, 50
  numeric_version, 86
  nyquist_velocity, 51
  param, 36
  plot, 55, 57
  plot.ppi, 52
  plot.scan, 53
  plot.vp, 6, 8, 55
  plot.vpi, 56
  plot.vpts, 6, 8, 58, 69
  ppi, 26, 45, 52, 53, 61
  project_as_ppi, 59, 76
  pvol, 66

88
rasterize, 70
rcs, 40–42, 62, 63
rcs<-, 62, 63
read_cajun, 64
read_pvolfile, 65
read_vpfiles, 32, 66, 83
read_vpts, 67
regularize_vpts, 68, 83
res, 25, 43, 70

scan, 31, 38
scan_to_raster, 69
scan_to_spatial, 70, 71
sd_vvp_threshold, 6, 8, 72, 73
sd_vvp_threshold<-, 72, 73
select_vpfiles, 74
SpatialGridDataFrame, 76
summary.param, 75
summary.ppi, 76
summary.pvol, 78
summary.scan, 79
summary.vp, 80
summary.vpts, 82
sunrise, 6, 8, 57
sunrise(sunrise_sunset), 84
sunrise_sunset, 84
sunset, 6, 8
sunset(sunrise_sunset), 84

update_docker, 85

viridis, 48
vol2bird_version, 85
vp, 21, 22, 32, 33, 37, 55, 64, 83
vpts, 16, 32, 33, 37, 68