Package ‘Reddy’

October 13, 2024

Type Package

Title An open-source package for analyzing eddy-covariance measurements

Version 0.0.0.9000

Author Laura Mack

Maintainer Laura Mack <laura.mack@geo.uio.no>

Description The package Reddy provides functions from post-processing over analyzing to plotting turbulence data from eddy-covariance measurements.

Imports pracma, MASS, RcppRoll

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

NeedsCompilation no

R topics documented:

`ah2rh` *Converts absolute humidity to relative humidity*

Description

Calculates absolute humidity from relative humidity and temperature

Usage

```
ah2rh(ah, temp)
```

Arguments

`ah`	absolute humidity [kg/m^3]
`temp`	temperature [K]

Value

relative humidity [percent]

`averaging` *accumulating / averaging*

Description

averaging of a timeseries

Usage

```
averaging(var, tres1 = 0.05, tres2 = c(1, 10, 30) * 60)
```

Arguments

`var`	timeseries
`tres1`	time resolution [s] of the given timeseries `var`, default `tres1 = 0.05` (for 20 Hz)
`tres2`	desired time resolution(s) [s] of the averaged timeseries (scalar or vector), default `tres2 = c(1,10,30)*60` (for 1, 10 and 30 minutes)

Value

list containing mean and standard deviation of the timeseries for the desired time interval(s)

Examples

```
ts=rnorm(30*60*20) #30 minutes of 20 Hz measurements
averaging(ts)
```

`binning` *discrete binning*

Description

discrete binning of a variable `var1` based on another variable `var2` (e.g., the stability parameter `zeta`)

Usage

```
binning(var1, var2, bins)
```

Arguments

`var1`	vector, variable that should be binned
`var2`	vector, variable used for the binning
`bins`	vector, providing the intervals of the bins of `var2`

Value

matrix of dimension (`length(bins)-1,4`) with columns representing mean, median, q25, q75

Examples

```
zeta_bins=c(-10^(3:-3),10^(-3:3))
zeta_vals=rnorm(1000)
vals=runif(1000)
binned=binning(vals,zeta_vals,zeta_bins)
```

calc_anisotropy *Invariant analysis of Reynolds stress tensor*

Description

Invariant analysis of Reynolds stress tensor, calculation of Lumley and barycentric map coordinates and anisotropy

Usage

```
calc_anisotropy(a11, a12, a13, a22, a23, a33)
```

Arguments

`a11`	R11 element of Reynolds stress tensor: `u_sd^2` (scalar or vector)
`a12`	R12 element of Reynolds stress tensor: `cov(u,v)` (scalar or vector)
`a13`	R13 element of Reynolds stress tensor: `cov(u,w)` (scalar or vector)
`a22`	R22 element of Reynolds stress tensor: `v_sd^2` (scalar or vector)
`a23`	R23 element of Reynolds stress tensor: `cov(v,w)` (scalar or vector)
`a33`	R33 element of Reynolds stress tensor: `w_sd^2` (scalar or vector)

Value

list containing `xb`, `yb`, `eta`, `xi`, all eigenvalues and eigenvectors (`eta`, `xi` are the coordinates of the Lumley triangle and `xb`, `yb` the coordinates of the barycentric map)

Examples

```
calc_anisotropy(1,0,0,1,0,1) #isotropic
calc_anisotropy(1,0,1,1,0,1) #some anisotropy
```

calc_br *Bowen ratio BR*

Description

Calculates the Bowen ratio as ratio of sensible and latent heat flux, i.e., BR := SH/LH

Usage

```
calc_br(sh, lh)
```

Arguments

`sh`	sensible heat flux [W/m^2]
`lh`	latent heat flux [W/m^2]

Value

Bowen ratio [-]

`calc_circular_mean` *calculates circular mean*

Description

calculates circular mean

Usage

```
calc_circular_mean(x, na.rm = TRUE)
```

Arguments

`x`	input vector, e.g. wind directions [degree]
`na.rm`	should NA values be removed? default `TRUE`

Value

circular mean of x values

Examples

```
wd=c(280,90)
calc_circular_mean(wd)
```

calc_cov — *Calculates covariance of two timeseries using pair-wise complete observations*

Description

Calculates cov(x,y)

Usage

```
calc_cov(x, y)
```

Arguments

`x`	timeseries 1
`y`	timeseries 2

Value

cov(x,y)

Examples

```
set.seed(5)
x=rnorm(100)
y=rnorm(100)
y[1:10]=NA
cov_xy=calc_cov(x,y)
```

calc_csi — *Clear Sky Index (CSI)*

Description

Calculates clear sky index

Usage

```
calc_csi(temp, lw_in, rh = NULL, e = NULL)
```

Arguments

`temp`	temperature [K]
`lw_in`	longwave incoming radiation [W/m^2]
`rh`	relative humidity [percent]
`e`	vapor pressure [Pa] (either rh or e have to be given)

Value

CSI, clear sky index

---

calc_decoupling_param *Decoupling metric (Omega)*

Description

Calculates the decoupling metric (Omega) from Peltola et al., 2021 (without vegetation)

Usage

```
calc_decoupling_param(w_sd, N, z = 2)
```

Arguments

w_sd	standard deviation of vertical velocity [m/s]
N	Brunt-Vaisala frequency [1/s]
z	measurement height [m]

Value

decoupling metric (Omega) [-]

---

calc_dshear *Directional Shear*

Description

Calculates a measure for directional shear alpha_uw = arctan(cov(v,w)/cov(u,w))

Usage

```
calc_dshear(cov_uw, cov_vw)
```

Arguments

cov_uw	covariance cov(u,w)
cov_vw	covariance cov(v,w)

Value

angle that describes the impact of directional shear [deg]

Examples

```
calc_dshear(-0.5,0) #no shear
calc_dshear(-0.5,-0.1)
```

`calc_ef` *Evaporative fraction*

Description

Calculates the evaporative fraction EF := LH/(SH+LH)

Usage

```
calc_ef(sh, lh)
```

Arguments

`sh`	sensible heat flux [W/m^2]
`lh`	latent heat flux [W/m^2]

Value

evaporative fraction [-]

`calc_evapotranspiration` *Evapotranspiration*

Description

Calculates evapotranspiration from latent heat flux

Usage

```
calc_evapotranspiration(lh, temp = NULL)
```

Arguments

`lh`	latent heat flux [W/m^2]
`temp`	temperature [K] (optional), if provided, the latent heat of vaporization is calculated temperature-dependent

Value

evapotranspiration [kg/(s*m^2)]

`calc_flux_footprint` *Flux-Footprint Parametrization (FFP) according to Kljun et al., 2015*

Description

Calculates the Flux-Footprint Parametrization (FFP) according to Kljun et al., 2015

Usage

```
calc_flux_footprint(
  zm,
  u_mean = NA,
  h,
  L,
  v_sd,
  ustar,
  z0 = NA,
  contours = seq(0.9, 0.1, -0.1),
  nres = 1000
)
```

Arguments

`zm`	measurement height [m]
`u_mean`	mean horizontal wind speed [m/s] (alternatively you can also use `z0`)
`h`	boundary-layer height [m]
`L`	Obukhov length [m]
`v_sd`	standard deviation of crosswind [m/s]
`ustar`	friction velocity [m/s]
`z0`	roughness length [m] (either `u_mean` or `z0` have to be given)
`contours`	which contour lines should be calculated? default: `contours=seq(0.9,0.1,-0.1)`
`nres`	resolution (default: `nres=1000`)

Value

list containing all relevant flux footprint information

Examples

```
ffp=calc_flux_footprint(zm=20,u_mean=2,h=200,L=-1.5,v_sd=0.6,ustar=0.4,contours=0.8)
```

calc_gustfactor *Gust Factor*

Description

Calculates gust factor G := ws_max/ws_mean

Usage

```
calc_gustfactor(ws_max, ws_mean)
```

Arguments

`ws_max`	wind speed [m/s]
`ws_mean`	wind speed maximum [m/s]

Value

gust factor [-]

calc_iw *Vertical Turbulence Intensity Iw*

Description

Calculates vertical turbulence intensity `Iw = w_sd/ws_mean`

Usage

```
calc_iw(w_sd, ws_mean)
```

Arguments

`w_sd`	standard deviation of vertical wind (w-wind)
`ws_mean`	horizontal wind speed

Value

vertical turbulence intensity [-]

`calc_L` *Obukhov length*

Description

Calculates Obukhov length from friction velocity, mean temperature and cov(T,w)

Usage

```
calc_L(ustar, T_mean, cov_wT)
```

Arguments

`ustar`	friction velocity (e.g., from `calc_ustar`) [m/s]
`T_mean`	mean temperature [K]
`cov_wT`	covariance cov(w,T) [m/s K]

Value

Obukhov length [m]

`calc_mrd` *Multiresolution Decomposition (MRD) according to Vickers and Mahrt, 2003*

Description

Calculates multiresolution decomposition (MRD) according to Vickers and Mahrt, 2003

Usage

```
calc_mrd(var1, var2 = NULL, time_res = 0.05)
```

Arguments

`var1`	timeseries of a variable
`var2`	timeseries of another variable to calculate the cospectrum of `var1` and `var2`, optional (default is `NULL`)
`time_res`	time resolution of the given timeseries in seconds (e.g., `time_res = 0.05` for 20 Hz)

Value

MRD in form of a data frame containing the columns: index, scale, time, mean, median, q25, q75

Examples

```
series=c(1,3,2,5,1,2,1,3) #example used in Vickers and Mahrt, 2003
calc_mrd(series)
```

---

calc_phim — *Calculates Phi_m*

Description

calculates scaling function Phi_m (for momentum)

Usage

```
calc_phim(U1, U2, ustar, zm, dz)
```

Arguments

U1	wind speed at the lower level [m/s]
U2	wind speed at the upper level [m/s]
ustar	friction velocity [m/s]
zm	measurement/scaling height [m]
dz	height difference of the two measurements [m]

Value

Phi_m

---

calc_phit — *Calculates Phi_t*

Description

calculate scaling function Phi_t (for heat)

Usage

```
calc_phit(T1, T2, cov_wT, ustar, zm, dz)
```

Arguments

`T1`	temperature at the lower level [K]
`T2`	temperature at the upper level [K]
`cov_wT`	covariance cov(w,T) [K m/s]
`ustar`	friction velocity [m/s]
`zm`	measurement/scaling height [m]
`dz`	height difference of the two measurements [m]

Value

Phi_t

`calc_phix` *Calculates Phi_x (general flux-variance relation)*

Description

calculates Phi_x = sigma_x/xstar (for general flux-variance relation)

Usage

```
calc_phix(sigma_x, x, ustar)
```

Arguments

`sigma_x`	standard deviation of x
`x`	variable that should be scaled, e.g. vertical flux of x with x = T or x = q
`ustar`	friction velocity [m/s]

Value

Phi_x = sigma_x/xstar

`calc_quadrant_analysis` *Calculating Coherent Structures following Quadrant Analysis*

Description

Calculates occurrence fraction and strength of the four quadrants in the framework of quadrant analysis

Usage

```
calc_quadrant_analysis(
  xval,
  yval,
  do_normalization = TRUE,
  hole_sizes = seq(0, 10),
  orient = "+"
)
```

Arguments

`xval`	values of x variable (vector)
`yval`	values of y variable (vector)
`do_normalization`	should the values be normalized? i.e. `(x-mean(x))/sd(x)`, default: `do_normalization=TRUE`
`hole_sizes`	vector containing desired hole sizes (integers >= 0)
`orient`	only relevant for exuberance and organization ratio: if down-gradient flux corresponds to positive values, use `orient="+"` (for sensible and latent heat flux), if down-gradient flux corresponds to negative values, use `orient="-"` (for momentum flux and CO2 flux)

Value

list containing occurrence fraction and strength (calculated based on product and covariance) for all four quadrants (mathematical orientation) as well as the therefrom derived measures exuberance and organization ratio, i.e. the ratio of the strength (or occurrence frequency, respectively) of disorganized to organized structures

Examples

```
a=rnorm(100)
b=rnorm(100)
qa_ab=calc_quadrant_analysis(a,b)
```

`calc_ri` *Calculates Richardson number Ri*

Description

calculates Richardson number Ri

Usage

```
calc_ri(T1, T2, U1, U2, dz)
```

Arguments

`T1`	temperature at the lower level [K]
`T2`	temperature at the upper level [K]
`U1`	wind speed at the lower level [m/s]
`U2`	wind speed at the upper level [m/s]
`dz`	height difference of the two measurements [m]

Value

Ri

`calc_satvaporpressure` *Saturation vapor pressure over water*

Description

Calculates the saturation vapor pressure over water for given temperature and pressure

Usage

```
calc_satvaporpressure(temp)
```

Arguments

`temp`	temperature [K]

Value

E_s, saturation vapor pressure over water [Pa]

`calc_spectrum` *Spectrum of timeseries by wrapping rbase::spectrum()*

Description

Calculates and plots the averaged turbulence spectrum (as wrapper of rbase::spectrum)

Usage

```
calc_spectrum(ts, nbins = 100, plot = TRUE)
```

Arguments

`ts`	timeseries
`nbins`	number of bins used to average the spectrum, default `nbins=100`
`plot`	should the spectrum be plotted? default `plot=TRUE`

Value

binned spectrum

`calc_spectrum1D` *Spectrum of timeseries (1D)*

Description

Calculates and plots turbulence spectrum (in time) calculated using FFT (and optionally bins it)

Usage

```
calc_spectrum1D(ts, tres = 0.05, nbins = NULL, plot = TRUE, ...)
```

Arguments

`ts`	timeseries
`tres`	time resolution of the given timeseries, default `tres=0.05` for 20 Hz
`nbins`	number of bins used to average the spectrum, default `nbins=NULL`, i.e. no further binning is applied (means number of bins equals half of length of input timeseries)
`plot`	should the spectrum be plotted? default `plot=TRUE`
`...`	further arguments passed to plot function

Value

1D FFT spectrum

`calc_spectrum2D` *Spatial spectrum (2D)*

Description

Calculates and plots turbulence spectrum (in space) calculated using FFT or DCT (and optionally bins it)

Usage

```
calc_spectrum2D(
  field,
  xres = 1000,
  yres = NULL,
  nbins = NULL,
  method = "fft",
  plot = TRUE,
  ...
)
```

Arguments

`field`	two-dimensional input field
`xres`	spatial resolution in x-direction
`yres`	spatial resolution in y-direction, default `yres=NULL` meaning that the field is equidistant and `yres=xres` is used
`nbins`	number of bins used to average the spectrum, default `nbins=NULL`, i.e. no further binning is applied (means number of bins equals half of length of input timeseries)
`method`	method used to calculate the spectrum, can be either FFT (fast Fourier transform) or DCT (discrete cosine transform), default FFT
`plot`	should the spectrum be plotted? default `plot=TRUE`
`...`	further arguments passed to plot function

Value

2D spectrum from FFT or DCT

calc_theta   *Potential temperature*

Description

Calculates potential temperature for given temperature and pressure

Usage

```
calc_theta(temp, pres)
```

Arguments

`temp`	temperature [K]
`pres`	pressure [Pa]

Value

potential temperature [K]

calc_ti   *Horizontal Turbulence Intensity TI*

Description

Calculates horizontal turbulence intensity `TI = sqrt(u_sd^2+v_sd^2)/ws_mean`

Usage

```
calc_ti(u_sd, v_sd, ws_mean)
```

Arguments

`u_sd`	standard deviation of streamwise wind (u-wind)
`v_sd`	standard deviation of crosswise wind (v-wind)
`ws_mean`	horizontal wind speed

Value

horizontal turbulence intensity [-]

calc_tke *Turbulent Kinetic Energy TKE*

Description

Calculates turbulent kinetic energy (TKE) from `u_sd`, `v_sd` and `w_sd`

Usage

```
calc_tke(u_sd, v_sd, w_sd)
```

Arguments

`u_sd`	standard deviation of u-wind [m/s]
`v_sd`	standard deviation of v-wind [m/s]
`w_sd`	standard deviation of w-wind [m/s]

Value

turbulent kinetic energy TKE [m^2/s^2]

calc_Tv *Virtual temperature*

Description

Calculates virtual temperature for given temperature and specific humidity (mixing ratio)

Usage

```
calc_Tv(temp, q)
```

Arguments

`temp`	temperature [K]
`q`	specific humidity [kg/kg]

Value

virtual temperature [K]

calc_ustar *Friction Velocity*

Description

Calculates friction velocity from the covariances cov(u,w) and cov(v,w)

Usage

```
calc_ustar(cov_uw, cov_vw)
```

Arguments

`cov_uw`	covariance cov(u,w) [m^2/s^2]
`cov_vw`	covariance cov(v,w) [m^2/s^2]

Value

friction velocity [m/s]

calc_var *Velocity Aspect Ratio (VAR)*

Description

Calculates the velocity aspect ratio: `VAR = sqrt(2)*w_sd/sqrt(u_sd^2+v_sd^2)`

Usage

```
calc_var(u_sd, v_sd, w_sd)
```

Arguments

`u_sd`	standard deviation of streamwise wind (u-wind)
`v_sd`	standard deviation of crosswise wind (v-wind)
`w_sd`	standard deviation of vertical wind (w-wind)

Value

velocity aspect ratio [-]

calc_vpd — *Vapor pressure deficit (VPD)*

Description

Calculates vapor pressure deficit (VPD) from temperature and relative humidity using Arrhenius formula

Usage

```
calc_vpd(temp, rh)
```

Arguments

`temp`	temperature [K]
`rh`	relative humidity [percent]

Value

VPD, vapor pressure deficit [Pa]

calc_vtke — *Turbulent Kinetic Energy Velocity Scale*

Description

Calculates the velocity scale of turbulent kinetic energy (TKE): `Vtke = sqrt(TKE)`

Usage

```
calc_vtke(u_sd, v_sd, w_sd)
```

Arguments

`u_sd`	standard deviation of u-wind [m/s]
`v_sd`	standard deviation of v-wind [m/s]
`w_sd`	standard deviation of w-wind [m/s]

Value

turbulent kinetic energy velocity scale [m/s]

`calc_windDirection` *Wind Direction*

Description

Calculates (horizontal) wind direction

Usage

```
calc_windDirection(u, v)
```

Arguments

`u`	u-wind [m/s]
`v`	v-wind [m/s]

Value

wind direction [deg]

`calc_windSpeed2D` *Horizontal Wind Speed*

Description

Calculates horizontal wind speed

Usage

```
calc_windSpeed2D(u, v)
```

Arguments

`u`	u-wind [m/s]
`v`	v-wind [m/s]

Value

wind speed [m/s]

calc_windSpeed3D *Wind Speed (3D)*

Description

Calculates wind speed (3D)

Usage

```
calc_windSpeed3D(u, v, w)
```

Arguments

u	u-wind [m/s]
v	v-wind [m/s]
w	w-wind [m/s]

Value

wind speed (3D) [m/s]

calc_xstar *Calculates xstar (denominator for general flux-variance relation)*

Description

calculates xstar = x/ustar (for general flux-variance relation)

Usage

```
calc_xstar(x, ustar)
```

Arguments

x	variable that should be scaled
ustar	friction velocity [m/s]

Value

xstar = x/ustar

calc_zeta *Stability Parameter*

Description

Calculates dimensionless stability parameter from Obukhov length and measurement height, i.e. zeta = z/L

Usage

```
calc_zeta(z, L)
```

Arguments

z	measurement height [m]
L	Obukhov length [m] (e.g., from calc_L)

Value

stability parameter [-]

cov2cf *Converts cov(co2,w) to CO2 flux*

Description

Converts cov(co2,w) to CO2 flux

Usage

```
cov2cf(cov_co2w, rho = NULL)
```

Arguments

cov_co2w	covariance cov(co2,w) [m/s]
rho	density of air [kg/m^3] (optional)

Value

CO2 flux [kg/(m^2*s)]

cov2lh — *Converts cov(w,q) to latent heat flux LH*

Description

Converts cov(w,q) to latent heat flux LH

Usage

```
cov2lh(cov_wq, rho = NULL)
```

Arguments

`cov_wq`	covariance cov(w,q) [m/s]
`rho`	density of air [kg/m^3] (optional)

Value

latent heat flux [W/m^2]

cov2sh — *Converts cov(w,T) to sensible heat flux SH*

Description

Converts cov(T,w) to sensible heat flux SH

Usage

```
cov2sh(cov_wT, rho = NULL)
```

Arguments

`cov_wT`	covariance cov(w,T) [K m/s]
`rho`	density of air [kg/m^3] (optional)

Value

sensible heat flux [W/m^2]

deaccumulate1h *deaccumulation*

Description

hourly deaccumulation, e.g. for fluxes from model output

Usage

```
deaccumulate1h(dat, factor = -1/3600)
```

Arguments

`dat`	vector (with dimension time) or array (with dimension x, y, time)
`factor`	factor for unit and sign conversion, default: `factor = -1/3600` for converting hour to seconds and adapting the sign convention

Value

vector or array hourly deaccumulated (same dimension as input)

despiking *Despiking*

Description

Applies (up to) three despiking methods based on (1) pre-defined thresholds, (2) median deviation (MAD) test and (3) skewness and kurtosis

Usage

```
despiking(
  series,
  thresholds = c(NA, NA),
  mad_factor = 10,
  threshold_skewness = 2,
  threshold_kurtosis = 8
)
```

Arguments

`series` timeseries that shall be despiked

`thresholds` vector with two elements representing lower and upper bounds for despiking (pre-defined thresholds), `NA` means that the respective bound is not used

`mad_factor` factor for the MAD test, default `mad_factor` = 10

`threshold_skewness` threshold for skewness test, default `threshold_skewness` = 2

`threshold_kurtosis` threshold for kurtosis test, default `threshold_kurtosis` = 8

Value

despiked timeseries

Examples

```
set.seed(5)
ts1=rnorm(100)
despiking(ts1,thresholds=c(-1,1))

ts2=rexp(1000)
despiking(ts2)
```

`ECprocessing` *Eddy-covariance post-processing*

Description

An eddy-covariance post-processing routine utilizing the functions from ec_processing.R

Usage

```
ECprocessing(
  u,
  v,
  w,
  temp,
  h2o = NULL,
  co2 = NULL,
  ch4 = NULL,
  time_resolution = 0.05,
  time_averaging = 30,
  measurement_height = 1,
  do_despiking = TRUE,
  despike_u = c(-15, 15, 10, 2, 8),
```

```
    despike_v = c(-15, 15, 10, 2, 8),
    despike_w = c(-4, 4, 10, 2, 8),
    despike_temp = c(230, 300, 10, 2, 8),
    despike_h2o = c(0, 12, 10, 2, 8),
    despike_co2 = c(0, 12, 10, 2, 8),
    despike_ch4 = c(0, 12, 10, 2, 8),
    do_detrending = FALSE,
    do_double_rotation = TRUE,
    do_planar_fit = FALSE,
    do_flagging = TRUE,
    dir_blocked = c(0, 0),
    do_SNDcorrection = TRUE,
    A = 7/8,
    B = 7/8,
    store = TRUE,
    format_out = "txt",
    filename = NULL,
    meta = TRUE
  )
```

Arguments

`u` u-wind [m/s] (sonic)

`v` v-wind [m/s] (sonic)

`w` w-wind [m/s] (sonic)

`temp` temperature [K] (sonic)

`h2o` H2O mixing ratio (gas analyzer, optional)

`co2` CO2 mixing ratio (gas analyzer, optional)

`ch4` CH4 mixing ratio (gas analyzer, optional)

`time_resolution`
time resolution of the measurements [s], default 20 Hz = 0.05 s

`time_averaging`
desired time averaging for flux calculations [min], default 30 minutes

`measurement_height`
measurement height [m], only used for calculation of the stability parameter `zeta`

`do_despiking` locigal, should the data be despiked? default `TRUE`

`despike_u` vector containing 5 elements: lower and upper bound, MAD factor, threshold skewness, threshold kurtosis. Details see `?despiking`. Default `despike_u=c(-15,15,10,2,8)`

`despike_v` vector containing 5 elements: lower and upper bound, MAD factor, threshold skewness, threshold kurtosis. Details see `?despiking`. Default `despike_v=c(-15,15,10,2,8)`

`despike_w` vector containing 5 elements: lower and upper bound, MAD factor, threshold skewness, threshold kurtosis. Details see `?despiking`. Default `despike_w=c(-4,4,10,2,8)`

`despike_temp` vector containing 5 elements: lower and upper bound, MAD factor, threshold skewness, threshold kurtosis. Details see `?despiking`. Default `despike_temp=c(230,300,10,` `despike_h2o` vector containing 5 elements: lower and upper bound, MAD factor, threshold skewness, threshold kurtosis. Details see `?despiking`. Default `despike_h2o=c(0,12,10,2,8)`

`despike_co2` vector containing 5 elements: lower and upper bound, MAD factor, threshold skewness, threshold kurtosis. Details see `?despiking`. Default `despike_co2=c(0,12,10,2,8)`

`despike_ch4` vector containing 5 elements: lower and upper bound, MAD factor, threshold skewness, threshold kurtosis. Details see `?despiking`. Default `despike_ch4=c(0,12,10,2,8)`

`do_detrending` logical, should the data be linearly detrended? default `FALSE`

`do_double_rotation` locigal, should the wind data be double rotated? default `TRUE`

`do_planar_fit` locigal, should the data be rotated with planar fit? default `FALSE` (either double rotation or planar fit can be `TRUE`)

`do_flagging` locigal, should the data be flagged? default `TRUE`, i.e. several flags are calculated, but no data is removed, can be used for quality analysis

`dir_blocked` vector containing 2 elements: wind directions blocked through mast or tower, used in flow distortion flag only

`do_SNDcorrection` locigal, should SND correction be applied to the buoyancy flux? default `TRUE`

`A` constant used in SND correction, default `A=7/8` for CSAT3 sonic

`B` constant used in SND correction, default `A=7/8` for CSAT3 sonic

`store` logical, should the output be stored? default `TRUE`

`format_out` file format of the output, can be either `txt` or `rds` (for netcdf, see separate function), only used if `store=TRUE`

`filename` desired output filename, default `NULL`, the date and runtime will be used to create a filename, only used if `store=TRUE`

`meta` locical, should meta data be stored? default `TRUE`

Value

data frame of post-processed eddy-covariance data (that is also stored in the output file by default)

`flag_distortion` *Flow Distortion Flag and Wind Constancy Ratio*

Description

Flow Distortion Flag according to Mauder et al., 2013: Wind coming from (pre-defined) directions blocked by the measurement device is flaged with 2 (for wind speeds greater than 0.1 assuming that during calm wind the wind direction is not well-defined). The wind constancy ratio is calculated to quantify the variability of horizontal wind direction according to Mahrt, 1999.

Usage

```
flag_distortion(u, v, dir_blocked = c(30, 60), threshold_cr = 0.9)
```

Arguments

`u`	u-wind (levelled sonic)
`v`	v-wind (levelled sonic)
`dir_blocked`	vector containing the lower and upper bound of the blocked wind sector in degrees (e.g., `dir_blocked = c(30,60)`)
`threshold_cr`	threshold for constancy ratio (default `threshold_cr = 0.9`, may be adapted to used data set)

Value

distortion flags (0: in full agreement with the criterion ... 2: does not fulfill the criterion)

`flag_most` *Integral Turbulence Characteristics Flag*

Description

Integral Turbulence Characteristics Flag: Tests the consistency with Monin-Obukhov similarity theory using the scaling functions from Panofsky and Dutton, 1984.

Usage

```
flag_most(w_sd, ustar, zeta)
```

Arguments

`w_sd`	standard deviation of vertical velocity
`ustar`	friction velocity
`zeta`	stability parameter `zeta = z/L`

Value

integral turbulence characteristics flags (0: in full agreement with the criterion ... 2: does not fulfill the criterion)

Examples

```
itc_flag=flag_most(0.2,0.4,-0.3)
```

flag_stationarity *Stationarity Flag*

Description

Stationarity Flag according to Foken and Wichura, 1996 based on the assumption that the covariance of two variables (`var1` and `var2`, one usually representing vertical velocity) calculated for blocks (of length `nsub`) does not differ to much from the total covariance

Usage

```
flag_stationarity(var1, var2, nsub = 3000)
```

Arguments

`var1`	variable 1
`var2`	variable 2 (same length as `var1`, usually either `var1` or `var2` represent vertical velocity)
`nsub`	number of elements used for subsampling (`nsub < length(var1)`)

Value

stationarity flags (0: in full agreement with the criterion ... 2: does not fulfill the criterion)

Examples

```
set.seed(5)
ts1=rnorm(30)
ts2=rnorm(30)
flag_stationarity(ts1,ts2,nsub=6)
```

flag_w *Vertical Velocity Flag*

Description

Vertical Velocity Flag according to Mauder et al., 2013: After rotation the vertical velocity should vanish, this flag flags high remaining vertical velocities.

Usage

```
flag_w(w)
```

Arguments

`w`	vertical velocity

Value

vertical velocity flags (0: in full agreement with the criterion ... 2: does not fulfill the criterion)

`gapfilling` *gap-filling*

Description

gap-filling of a timeseries based on linear or constant interpolation

Usage

```
gapfilling(var, nmissing = 4, method = "linear")
```

Arguments

`var`	timeseries, where NA indicates missing values that should be filled
`nmissing`	number of allowed missing values, default `nmissing = 4`
`method`	interpolation method, can be either `method = "linear"` for linear interpolation (default) or `method = "constant"` for constant interpolation

Value

gap-filled timeseries

Examples

```
ts1=c(1,2,NA,0)
gapfilling(ts1) #1,2,1,0
gapfilling(ts1,method="constant") #1,2,2,0
gapfilling(ts1,nmissing=0) #too many missing values
```

plot_barycentric_map *Plot in barycentric map*

Description

Plots `(xb,yb)` from invariant analysis of Reynolds stress tensor (`calc_anisotropy`) in barycentric map

Usage

```
plot_barycentric_map(xb, yb, contours = c(5, 10, 20))
```

Arguments

`xb`	xb coordinate (e.g., from `calc_anisotropy`)
`yb`	yb coordinate (e.g., from `calc_anisotropy`)
`contours`	vector containing levels of contour lines for 2d kernel density estimation, default: `contours=c(5,10,20)`

Value

plots (xb, yb) in barycentric map with 2d kernel density estimation (no return)

Examples

```
nm=100
example1=calc_anisotropy(rep(1,nm),rep(0,nm),runif(nm,0,1),
rep(1,nm),rep(0,nm),runif(nm,1,1.5))
plot_barycentric_map(example1$xb,example1$yb)
```

plot_flux_footprint *Plot Flux-Footprint*

Description

Plots Flux-Footprint Parametrization (FFP) according to Kljun et al., 2015

Usage

```
plot_flux_footprint(ffp, levels = c(0, 10^seq(-6, -3, 0.1)))
```

Arguments

`ffp`	an object returned from `calc_flux_footprint`
`levels`	levels used for filled contour plot of footprint, default `levels=c(0,10^seq(-6,-3,0.1))`

Value

no return

Examples

```
ffp=calc_flux_footprint(zm=5,u_mean=5,h=700,L=-1.3,v_sd=1.2,ustar=0.35)
plot_flux_footprint(ffp)
```

`plot_mrd` *Plotting Multiresolution Decomposition*

Description

Plots multiresolution decomposition (MRD)

Usage

```
plot_mrd(mrd_out, ...)
```

Arguments

`mrd_out`	an object returned from `calc_mrd`
`...`	parameters passed to plot function

Value

creates a plot of MRD with logarithmic time scale (no return)

Examples

```
set.seed(5)
series=rnorm(2^10)
mrd_test=calc_mrd(c(series))
plot_mrd(mrd_test)
```

`plot_quadrant_analysis` *Plotting Quadrant Analysis*

Description

Plots two vectors in the framework of quadrant analysis with 2d kernel density estimation

Usage

```
plot_quadrant_analysis(
  xval,
  yval,
  do_normalization = TRUE,
  hole_sizes = c(1, 2),
  contours = 10^(-3:3),
  print_fit = TRUE,
  ...
)
```

Arguments

`xval`	values of x variable (vector)
`yval`	values of y variable (vector)
`do_normalization`	should the values be normalized? i.e. `(x-mean(x))/sd(x)`, default: `do_normalization=TRUE`
`hole_sizes`	vector containing desired hole sizes (integers >= 0), default: `hole_sizes=c(1,2)`
`contours`	vector containing levels of contour lines for 2d kernel density estimation, default: `contours=10^(-3:3)`
`print_fit`	should the fit summary from the linear regression be printed? default: `print_fit=TRUE`
`...`	arguments passed to plot function

Value

no return

Examples

```
a=rnorm(100)
b=rnorm(100)
plot_quadrant_analysis(a,b)
```

plot_seb — *Plotting of surface energy balance and calculation of surface energy balance unclosure*

Description

Plotting of surface energy balance and calculation of surface energy balance unclosure as residual flux and closure ratio

Usage

```
plot_seb(
  sw_in,
  sw_out,
  lw_in,
  lw_out,
  sh = NULL,
  lh = NULL,
  gh = NULL,
  time_vector = NULL,
  print_fit = TRUE,
  ...
)
```

Arguments

`sw_in`	incoming shortwave radiation [W/m^2] (as vector of time)
`sw_out`	outgoing shortwave radiation [W/m^2] (as vector of time)
`lw_in`	incoming longwave radiation [W/m^2] (as vector of time)
`lw_out`	outgoing longwave radiation [W/m^2] (as vector of time)
`sh`	sensible heat flux [W/m^2] (as vector of time) – if measured
`lh`	latent heat flux [W/m^2] (as vector of time) – if measured
`gh`	ground heat flux [W/m^2] (as vector of time) – if measured
`time_vector`	times used as x-axis labels (optional)
`print_fit`	should the fit summary be printed? default: `print_fit=TRUE`
`...`	optional plot parameters

Value

no return

`ppt2rho` *Unit conversion of "parts-per" to density (for closed-path gas analyzer)*

Description

Unit conversion of "parts-per" to density (for closed-path gas analyzer)

Usage

```
ppt2rho(ppt, T_mean = 288.15, pres = 101325, e = 0, gas = "H2O")
```

Arguments

`ppt`	measurement in parts per thousand [ppt]
`T_mean`	temperature [K]
`pres`	pressure [Pa]
`e`	water vapor pressure [Pa]
`gas`	which gas? can be either `H2O`, `CO2`, `CH4` (if `CO2`/`CH4` is selected, make sure that it's still in ppt and not ppm as usual)

Value

density of the gas [kg/m^3]

`pres2height` *Converts pressure to height (using barometric formula)*

Description

Calculates height from pressure

Usage

```
pres2height(pres, pres0 = 101315, temp0 = 288.15)
```

Arguments

`pres`	pressure [Pa]
`pres0`	reference pressure, scalar [Pa], default `pres0=101315`
`temp0`	reference temperature, scalar [K], default `temp0=288.15`

Value

height [m]

`Reddy package` *Introduction*

Description

EC postprocessing and analysis

Details

to be detailed

References

- Mack, L., Berntsen, T.K., Vercauteren, N., Pirk, N. (2024). Transfer Efficiency and Organization in Turbulent Transport over Alpine Tundra. Boundary-Layer Meteorology 190, 38. doi: https://doi.org/10.1007/s10546-024-00879-5

`rh2ah` *Converts relative humidity to absolute humidity*

Description

Calculates absolute humidity from relative humidity and temperature

Usage

```
rh2ah(rh, temp)
```

Arguments

`rh`	relative humidity [percent]
`temp`	temperature [K]

Value

absolute humidity [kg/m^3]

rh2q *Converts relative humidity to specific humidity*

Description

Calculates specific humidity from relative humidity, temperature and pressure

Usage

```
rh2q(rh, temp, pres)
```

Arguments

`rh`	relative humidity [percent]
`temp`	temperature [K]
`pres`	pressure [Pa]

Value

specific humidity [kg/kg]

rotate_double *Double rotation*

Description

Double rotation (i.e., sonic coordinate system will be aligned with streamlines)

Usage

```
rotate_double(u, v, w)
```

Arguments

`u`	u-wind (levelled sonic)
`v`	v-wind (levelled sonic)
`w`	w-wind (levelled sonic)

Value

list containing the wind in a natural coordinate system (streamwise, crosswise, vertical) and the two rotation angles theta and phi

Examples

```
wind_rotated=rotate_double(4,3,1) #double rotation can be applied instantenously
```

`rotate_planar` *Planar fit rotation*

Description

Planar fit rotation (i.e., sonic coordinate system will be aligned with the mean streamlines resulting in vanishing of w_mean)

Usage

```
rotate_planar(u, v, w, bias = c(0, 0, 0))
```

Arguments

`u`	u-wind (levelled sonic)
`v`	v-wind (levelled sonic)
`w`	w-wind (levelled sonic)
`bias`	a three-dimensional correction vector containing the offsets of u-, v-, w-wind

Value

list containing u, v, w after planar fit rotation as well as the rotation angles alpha, beta and gamma and the fitted offset c3

Examples

```
u=rnorm(1000)
v=rnorm(1000)
w=rnorm(1000)
wind_rotated=rotate_planar(u,v,w) #for planar fit a timeseries is required
```

`RTcorrection` *Response-time correction factor (spectral correction)*

Description

Calculates the response-time correction factor from cospectrum, e.g. Peltola et al., 2021

Usage

```
RTcorrection(cospectrum, freq, tau = 1)
```

Arguments

`cospectrum`	cospectrum
`freq`	frequency [Hz], corresponding to the cospectrum, i.e. same length
`tau`	response time of the instrument [s] (has to be determined first by comparison with another instrument, that samples faster)

Value

response-time correction factor (which then can be used to correct the covariance and fluxes by multiplication)

`scale_phih` *Scaling function for heat Phi_h*

Description

scaling function Phi_h

Usage

```
scale_phih(zeta, method = "BD")
```

Arguments

`zeta`	stability parameter [-]
`method`	defining from which paper the scaling function should be used, default `method="ecmwf"` for for the ones used in ECMWF-IFS

Value

Phi_h

`scale_phim` *Scaling function for momentum Phi_m*

Description

scaling function Phi_m

Usage

```
scale_phim(zeta, method = "BD")
```

Arguments

`zeta` stability parameter [-]

`method` defining from which paper the scaling function should be used, default `method="ecmwf"` for for the ones used in ECMWF-IFS, other option `method="BD"` for the lienar Businger-Dyer relations

Value

Phi_m

`scale_phiT` *Scaling function for temperature Phi_T*

Description

scaling function Phi_T

Usage

```
scale_phiT(zeta, method = "K1994")
```

Arguments

`zeta` stability parameter [-]

`method` defining from which paper the scaling function should be used, default `method="K1994"` for Katul, 1994, other option `method="SC2018"` for Stiperski and Calaf, 2018

Value

Phi_T

`scale_phiu` *Scaling function for horizontal windspeed Phi_u*

Description

scaling function Phi_u

Usage

```
scale_phiu(zeta, method = "PD1984")
```

Arguments

`zeta`	stability parameter [-]
`method`	defining from which paper the scaling function is used, default `method="PD1984"` for Panofsky and Dutton, 1984

Value

Phi_u

`scale_phiw` *Scaling function for vertical windspeed Phi_w*

Description

scaling function Phi_w

Usage

```
scale_phiw(zeta, method = "PD1984")
```

Arguments

`zeta`	stability parameter [-]
`method`	defining from which paper the scaling function is used, default `method="PD1984"` for Panofsky and Dutton, 1984

Value

Phi_w

`shift2maxccf` *Shifting two timeseries to match maximum cross-correlation*

Description

Shifts two timeseries to match their maximum cross-correlation (can be used e.g. for lag-time correction)

Usage

```
shift2maxccf(var1, var2, plot = TRUE)
```

Arguments

`var1`	vector, first timeseries
`var2`	vector, second timeseries
`plot`	logical, should the cross-correlation be plotted? default `plot = TRUE`

Value

a matrix cotaining timeseries `var1` and `var2` as columns after shifting to the maximum cross-correlation

Examples

```
ts1=runif(10)
ts2=c(1,1,ts1)
shifted=shift2maxccf(ts1,ts2)
```

`SNDcorrection` *SND and cross-wind correction of sensible heat flux*

Description

SND and cross-wind correction of sensible heat flux: converts the buoyancy flux cov(w,Ts) (based on sonic temperature Ts) to sensible heat flux

Usage

```
SNDcorrection(
  Ts_mean,
  u_mean,
  v_mean,
  cov_uw,
  cov_vw,
  cov_wTs,
  cov_qw = NULL,
  A = 7/8,
  B = 7/8
)
```

Arguments

`Ts_mean`	temperature [K] (averaged)
`u_mean`	u-wind [m/s] (averaged)
`v_mean`	v-wind [m/s] (averaged)
`cov_uw`	cov(u,w) [m^2/s^2]

`cov_vw`	cov(v,w) [m^2/s^2]
`cov_wTs`	cov(Ts,w) [K*m/s] (buoyancy flux)
`cov_qw`	cov(q,w) [kg/kg*m/s] (optional)
`A`	constant used in cross-wind correction, default `A = 7/8` for CSAT3
`B`	constant used in cross-wind correction, default `B = 7/8` for CSAT3

Value

SND correction of sensible heat flux

`sos2Ts` *Converts speed of sound (sos) to sonic temperature*

Description

Converts speed of sound (sos) to sonic temperature

Usage

```
sos2Ts(sos)
```

Arguments

`sos`	speed of sound [m/s]

Value

sonic temperature (virtual temperature) [K]

`ustar2z0` *Calculates surface roughness length z0 from friction velocity using the simple estimate from Charnock, 1955*

Description

Calculates surface roughness z0 from friction velocity using the simple estimate from Charnock, 1955: z0 = alpha*ustar^2/g with alpha=0.016 and g=9.81 m/s^2

Usage

```
ustar2z0(ustar)
```

Arguments

`ustar`	friction velocity [m/s]

Value

surface roughness length [m]

`WPLcorrection` *WPL correction*

Description

WPL correction: density correction for trace gas fluxes (i.e., converts volume- to mass-related quantity)

Usage

```
WPLcorrection(rho_w, rho_c = NULL, w, Ts, q)
```

Arguments

`rho_w`	measured water vapor density [kg/m^3]
`rho_c`	measured trace gas density [kg/m^3] (only if WPL-correction should be applied to another flux, e.g. CO2 flux, default `NULL`)
`w`	w-wind [m/s] (levelled sonic)
`Ts`	temperature [K] (sonic temperature or corrected temperature)
`q`	specific humidity [kg/kg] (if measured, default `NULL`)

Value

WPL correction of respective flux