Package ‘Reddy’

October 17, 2025

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 R-package Reddy provides functions to post-process, analyze and visualize eddy-covariance measurements.

Imports pracma, MASS, RcppRoll, gsignal, fields, omnibus

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

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]

Examples

```
ah2rh(0.005,273)
```

apply_quality_control *Apply quality control on high-frequency data (e.g. as post-processing for MRD or quadrant analysis)*

Description

Applies quality control and rotation to high-frequency data (and outputs the high-frequency data)

Usage

```
apply_quality_control(
  u,
  v,
  w,
  temp,
  h2o = NULL,
  co2 = NULL,
  ch4 = NULL,
  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(300, 500, 10, 4, 10),
  despike_ch4 = c(0, 12, 10, 2, 8),
  do_double_rotation = TRUE,
  do_planar_fit = FALSE,
  do_detrending = FALSE,
  do_flagging = 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)
`do_despiking`	logical, 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(230,300,10,2,8)`
`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_double_rotation`	logical, should the wind data be double rotated? default `TRUE`
`do_planar_fit`	logical, should the data be rotated with planar fit? default `FALSE` (either double rotation or planar fit can be `TRUE`)
`do_detrending`	logical, should the data be linearly detrended? default `FALSE`
`do_flagging`	logical, should the data be flagged? default `TRUE`, i.e. several flags are calculated, but no data is removed, can be used for quality analysis

Value

quality checked data in the same dimensions as the input variables

---

`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_2point_cor` *Two-point Correlation Function: Tool to study spatial characteristics of coherent structures*

Description

Two-point Correlation Function: Tool to study spatial characteristics of coherent structures

Usage

```
calc_2point_cor(ts1, ts2)
```

Arguments

`ts1`	timeseries at point 1 (high-resolution)
`ts2`	timeseries at point 2 (high-resolution)

Value

scalar giving the two-point correlation R(ts1,ts2)

Examples

```
set.seed(5)
ts1=rnorm(100)
ts2=rnorm(100)
cor_2point=calc_2point_cor(ts1,ts2)
```

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, plot = FALSE)
```

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)
plot	should the barycentric map be plotted? default `plot=FALSE`

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) #anisotropic
```

calc_blh *Boundary Layer Height*

Description

Calculates boundary layer height estimate following Nieuwstadt, 1981

Usage

```
calc_blh(L, ustar, f)
```

Arguments

`L`	Obukhov length [m]
`ustar`	friction velocity [m/s]
`f`	Coriolis parameter [1/s] (e.g. from `calc_coriolis`)

Value

Boundary layer height estimation [m]

Examples

```
calc_blh(-1,0.5,10^(-4))
```

`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 [-]

Examples

```
calc_br(50,20)
```

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_coriolis *Coriolis parameter*

Description

Calculates Coriolis parameter from latitude

Usage

```
calc_coriolis(phi)
```

Arguments

phi	latitude [deg]

Value

Coriolis parameter [1/s]

Examples

```
calc_coriolis(45)
```

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

Examples

```
calc_csi(273,230,70) #with relative humidity
```

`calc_decoupling_metric` *Decoupling metric (Omega)*

Description

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

Usage

```
calc_decoupling_metric(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) [-]

Examples

```
calc_decoupling_metric(1,1)
```

calc_distance *Calculates distance between two points on a sphere given in lon,lat*

Description

Calculates distance between to (lon, lat)-points on a spheroid

Usage

```
calc_distance(lon1, lat1, lon2, lat2)
```

Arguments

`lon1`	longitude location 1 [deg]
`lat1`	latitude location 1 [deg]
`lon2`	longitude location 2 [deg]
`lat2`	latitude location 2 [deg]

Value

distance between two points [m]

Examples

```
calc_distance(8,60,8,61)
```

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 [-]

Examples

```
calc_ef(50,20)
```

---

calc_ekman_layer_depth *Ekman layer thickness*

Description

Calculates Ekman layer thickness from eddy diffusivity and Coriolis parameter sqrt(2*Km/abs(f))

Usage

```
calc_ekman_layer_depth(Km, f)
```

Arguments

Km	eddy diffusivity [m^2/s]
f	Coriolis parameter [1/s] (e.g. from `calc_coriolis`)

Value

Ekman layer thickness [m] (derived from boundary layer equations)

Examples

```
calc_ekman_layer_depth(0.1,10^(-4))
```

`calc_flux_footprint_climatology`
Calculate Flux Footprint Climatology

Description

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

Usage

```
calc_flux_footprint_climatology(
  zm,
  ws_mean = NA,
  wd_mean = NA,
  L,
  v_sd,
  ustar,
  z0 = NA,
  blh = NA,
  contours = seq(0.9, 0.1, -0.1),
  nres = 1000,
  method = "Kljun2015",
  plot = TRUE,
  ...
)
```

Arguments

zm	measurement height [m]
ws_mean	mean horizontal wind speed [m/s] (alternatively you can also use z0)
wd_mean	mean wind direction [deg] (used to rotate flux footprint, optional)
L	Obukhov length [m]
v_sd	standard deviation of crosswind [m/s]
ustar	friction velocity [m/s]
z0	roughness length [m] (either ws_mean or z0 have to be given)
blh	boundary-layer height [m]

contours	which contour lines should be calculated? default: `contours=seq(0.9,0.1,-0.1)`
nres	resolution (scalar) (default: `nres=1000`)
method	method used to calculate FFP: can be either `method="Kljun2015"` (default) or `method="KM2001"`
plot	logical, should the flux footprint be plotted? default `plot=TRUE`
...	paraemters passed to image.plot function

Value

list containing all relevant flux footprint information

Examples

```
nit=2
#ffp=calc_flux_footprint_climatology(zm=20,ws_mean=rep(2,nit),blh=rep(200,nit),L=rep(-1.5,nit),
#          v_sd=rep(0.6,nit),ustar=rep(0.4,nit),contours=0.8,wd_mean=c(50,240)) #todo
```

`calc_flux_footprint_Kljun2015`

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_Kljun2015(
  zm,
  ws_mean = NA,
  wd_mean = NA,
  L,
  v_sd,
  ustar,
  z0 = NA,
  blh,
  contours = seq(0.9, 0.1, -0.1),
  nres = 1000,
  plot = TRUE
)
```

Arguments

zm	measurement height [m]
ws_mean	mean horizontal wind speed [m/s] (alternatively you can also use z0)
wd_mean	mean wind direction [deg] (used to rotate flux footprint, optional)
L	Obukhov length [m]
v_sd	standard deviation of crosswind [m/s]
ustar	friction velocity [m/s]
z0	roughness length [m] (either ws_mean or z0 have to be given)
blh	boundary-layer height [m]
contours	which contour lines should be calculated? default: contours=seq(0.9,0.1,-0.1)
nres	resolution (default: nres=1000)
plot	logical, should the flux footprint be plotted? default plot=TRUE

Value

list containing all relevant flux footprint information

Examples

```
#unrotated (i.e. if wind direction not given):
ffp=calc_flux_footprint_Kljun2015(zm=20,ws_mean=2,blh=200,L=-1.5,
     v_sd=0.6,ustar=0.4,contours=0.8)
#rotated (i.e. given wind direction):
ffp=calc_flux_footprint_Kljun2015(zm=20,ws_mean=2,wd_mean=80,blh=200,L=-1.5,
     v_sd=0.6,ustar=0.4,contours=0.8)
```

---

calc_flux_footprint_KM2001

Flux-Footprint Calculation according to Korman and Meixner, 2001

Description

Calculates the Flux-Footprint Parametrization (FFP) according to Korman and Meixner, 2001

Usage

```
calc_flux_footprint_KM2001(
  zm,
  ws_mean = NA,
  wd_mean = NA,
  L,
  v_sd,
  ustar,
```

```
  z0,
  contours = seq(0.9, 0.1, -0.1),
  nres = 1000,
  dx = 1,
  plot = TRUE
)
```

Arguments

zm	measurement height [m]
ws_mean	mean horizontal wind speed [m/s] (alternatively you can also use z0)
wd_mean	mean wind direction [deg] (used to rotate flux footprint, optional)
L	Obukhov length [m]
v_sd	standard deviation of crosswind [m/s]
ustar	friction velocity [m/s]
z0	roughness length [m] (either ws_mean or z0 have to be given)
contours	which contour lines should be calculated? default: contours=seq(0.9,0.1,-0.1)
nres	domain size (default: nres=1000 to get a domain ranging from -500 to 500)
dx	resolution (default: dx=1)
plot	logical, should the flux footprint be plotted? default plot=TRUE

Value

list containing all relevant flux footprint information

Examples

```
#unrotated (i.e. if wind direction not given):
ffp=calc_flux_footprint_KM2001(zm=20,ws_mean=2,L=-1.5,
     v_sd=0.6,ustar=0.4,z0=0.1,contours=0.8)
#rotated (i.e. given wind direction):
ffp=calc_flux_footprint_KM2001(zm=20,ws_mean=2,wd_mean=80,L=-1.5,
     v_sd=0.6,ustar=0.4,z0=0.1,contours=0.8)
```

---

calc_flux_intermittency

Flux intermittency

Description

Calculates flux intermittency FI = flux_sd/abs(flux) (flux_sd: sd of subsampled fluxes) following Mahrt, 1998 (similar to stationarity flag flag_stationarity)

Usage

```
calc_flux_intermittency(ts1, ts2 = NULL, nsub = 6000)
```

Arguments

`ts1`	timeseries 1
`ts2`	timeseries 2 (optional), if the flux should be calculated based on `ts1*ts2` (default `ts2=NULL`, i.e. `ts2` is not used)
`nsub`	number of elements used for subsampling, default `nsub=6000`, which corrosponds to 5 minutes of measurements from 20 Hz sampled half-hour (containing 30*60*20 = 36000 measurements)

Value

flux intermittency [-]

Examples

```
set.seed(5)
ts1=rnorm(30)
ts2=rnorm(30)
calc_flux_intermittency(ts1,ts2,nsub=6) #as product
calc_flux_intermittency(ts1*ts2,nsub=6) #the same from one variable
```

`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 [-]

Examples

```
calc_gustfactor(6,3)
```

calc_helmholtz_decomposition *Helmholtz-Hodge decomposition*

Description

Calculates Helmholtz-Hodge decomposition of horizontal wind using a spectral FFT-based method: decomposition of horizontal wind in rotational and divergent part: (u,v) = (u_rot,v_rot) + (u_div,v_div)

Usage

```
calc_helmholtz_decomposition(u, v, res = 1)
```

Arguments

`u`	zonal wind field with dimension (x,y)
`v`	meridional wind field with dimension (x,y)
`res`	spatial resolution (assuming equidistant grid)

Details

The implementation is based on the Python version from https://github.com/shixun22/helmholtz.

Value

list containing u_div, v_div, u_rot, v_rot

Examples

```
set.seed(5)
u=matrix(rnorm(100),ncol=10)
v=matrix(rnorm(100),ncol=10)
hd=calc_helmholtz_decomposition(u,v,100)
```

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 [-]

Examples

```
calc_iw(1,3) #unstable
```

`calc_k2d` *Calculates 2d (horizontal) wavenumber matrix from kx, ky*

Description

Calculates 2d (horizontal) wavenumber matrix from kx, ky

Usage

```
calc_k2d(kx, ky)
```

Arguments

`kx`	wavenumber in x-direction
`ky`	wavenumber in y-direction

Value

total spatial (horizontal) wavenumber k

Examples

```
kx=c(1:10)/10
ky=c(1:8)/8
k=calc_k2d(kx,ky)
```

calc_Kh *Calculates eddy conductivity K_h = -cov(w,T)/(dT/dz)*

Description

Calculates eddy conductivity K_h

Usage

```
calc_Kh(cov_wT, dT_dz)
```

Arguments

cov_wT	covariance cov(w,T) [K m/s]
dT_dz	vertical temperature gradient [K/m]

Value

eddy conductivity K_h [m^2/s]

Examples

```
calc_Kh(0.2,-1)
```

calc_Km *Calculates eddy viscosity K_m = -cov(u,w)/(du/dz)*

Description

Calculates eddy viscosity K_m

Usage

```
calc_Km(cov_uw, du_dz)
```

Arguments

cov_uw	covariance cov(u,w) [m^2/s^2]
du_dz	vertical wind speed gradient [1/s]

Value

eddy viscosity K_m [m^2/s]

Examples

```
calc_Km(-0.2,2)
```

---

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]

Examples

```
calc_L(0.2,273,0.1) #unstable
calc_L(0.2,273,-0.1) #stable
```

---

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, plot = TRUE, ...)
```

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)
`plot`	logical, should the MRD spectrum be plotted? default `plot=TRUE`
`...`	arguments passed to plot function

Value

MRD in form of data.frame with columns: index, m, 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_N2` *Brunt-Vaisala frequency squared*

Description

calculates Brunt-Vaisala frequency squared (N^2)

Usage

```
calc_N2(T1, T2, dz)
```

Arguments

`T1`	temperature at the lower level [K]
`T2`	temperature at the upper level [K]
`dz`	height difference of the two measurements [m]

Value

N2 [1/s^2]

calc_ogive — *Calculates Ogive (cumulative distribution function from cospectrum) based on MRD*

Description

Calculates Ogive from MRD spectram

Usage

```
calc_ogive(mrd, plot = TRUE, ...)
```

Arguments

`mrd`	an object returned from `calc_mrd`
`plot`	logical, should the ogive be plotted? default `plot=TRUE`
`...`	arguments passed to plot function

Examples

```
set.seed(5)
series=rnorm(2^10)
mrd_test=calc_mrd(c(series))
ogive_test=calc_ogive(mrd_test,plot=FALSE)
```

calc_ozmidov_scale — *Ozmidov scale (L_OZ)*

Description

Calculates the Ozmidov length scale L_OZ = sqrt(epsilon/N^3), with epsilon: TKE dissipation rate, and N: Brunt-Vaisala frequency

Usage

```
calc_ozmidov_scale(epsilon, N)
```

Arguments

`epsilon`	dissipation rate of TKE or w [m*s]
`N`	Brunt-Vaisala frequency [1/s]

Value

Ozmidov length scale [m]

Examples

```
calc_ozmidov_scale(-5/3,1*10^-4)
```

---

calc_phih *Calculates Phi_h*

Description

calculate scaling function Phi_h (for heat)

Usage

```
calc_phih(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_h

---

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_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_Pr` *Calculates turbulent Prandtl number Pr = K_m/K_h*

Description

Calculates turbulent Prandtl number Pr

Usage

```
calc_Pr(K_m, K_h)
```

Arguments

`K_m`	eddy viscosity [m^2/s]
`K_h`	eddy conductivity [m^2/s]

Value

Prandtl number [-]

Examples

```
calc_Pr(0.4,0.6)
```

`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 = "+",
  plot = 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)
`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)
`plot`	logical, should the quadrant analysis be plotted? default `plot=TRUE`
`...`	arguments passed to `plot_quadrant_analysis`

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 bulk 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_rif` *Calculates flux Richardson number Ri_f*

Description

calculates flux Richardson number Ri_f = g/T_mean*cov(w,T)/(cov(u,w)*du/dz)

Usage

```
calc_rif(cov_wT, cov_uw, U1, U2, dz, T_mean = NULL)
```

Arguments

`cov_wT`	covariance cov(w,T) [K m/s]
`cov_uw`	covariance cov(u,w) [m^2/s^2]
`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]
`T_mean`	mean temperature [K] (optional, used instead of T0=273.15)

Value

Ri_f [-]

`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]

Examples

```
calc_satvaporpressure(273)
```

`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, na.rm = 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`
`na.rm`	should NA values be removed from the timeseries? default `na.rm=TRUE`

Value

binned spectrum

Examples

```
set.seed(5)
ts=rnorm(1000)
calc_spectrum(ts,nbins=100,plot=FALSE)
```

`calc_spectrum1D` *Frequency spectrum (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,
  method = "fft",
  na.rm = TRUE,
  plot = TRUE,
  ...
)
```

Arguments

ts	timeseries of the variable for which the spectrum should be calculated
tres	time resolution [s] 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)
method	method used to calculate the spectrum, can be either FFT (fast Fourier transform) or DCT (discrete cosine transform), default FFT
na.rm	should NA values be removed from the timeseries? default `na.rm=TRUE`
plot	should the spectrum be plotted? default `plot=TRUE`
...	further arguments passed to plot function

Value

binned frequency spectrum from 1D FFT

Examples

```
set.seed(5)
ts=rnorm(1000)
calc_spectrum1D(ts) #no binning
calc_spectrum1D(ts,nbins=100) #binning
```

---

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

binned wavenumber spectrum from 2D FFT or DCT

Examples

```
set.seed(5)
field=matrix(rnorm(10000),nrow=100)
calc_spectrum2D(field,xres=100) #equidistant grid, no binning, fft
calc_spectrum2D(field,xres=100,yres=200) #non-equidistant grid, no binning, fft
calc_spectrum2D(field,xres=100,nbins=1000) #equidistant grid, binning, fft
calc_spectrum2D(field,xres=100,nbins=1000,method="dct") #equidistant grid, binning, dct
```

`calc_structure_function` *Structure functions*

Description

Calculates the structure function of given timeseries and given order, S := < (ts[t+dt], ts[t])^order >

Usage

```
calc_structure_function(ts, order = 2)
```

Arguments

`ts`	timeseries
`order`	order of the structure function, typically d = 2, 3, 4

Value

structure function

Examples

```
ts=rnorm(100)
S2_ts=calc_structure_function(ts,2)
```

`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]

Examples

```
calc_theta(273,70000)
```

`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 [-]

Examples

```
calc_ti(1,1,3)
```

---

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]

Examples

```
calc_tke(1,1,1)
```

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]

Examples

```
calc_Tv(273,0) #no difference
calc_Tv(273,0.1)
```

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 = 0)
```

Arguments

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

Value

friction velocity [m/s]

Examples

```
calc_ustar(-0.3,0.02)
```

---

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 [-]

Examples

```
calc_var(1,1,1) #"isotropic"
calc_var(1,1,2) #not isotropic
```

---

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]

Examples

```
calc_vpd(273,70)
```

---

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]

Examples

```
-
calc_vtke(1,1,1)
```

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]

Examples

```
calc_windDirection(3,3)
```

calc_windprofile *Wind profile from Monin-Obukhov similarity theory*

Description

Calculates vertical profile of horizontal wind speed following Monin-Obukhov similarity theory

Usage

```
calc_windprofile(zs, ustar, z0 = 0, d = 0, zeta = 0, method = "ecmwf")
```

Arguments

zs	scalar or vector, heights [m] at which the horizontal wind speed should be calculate
ustar	friction velocity [m/s]
z0	surface roughness length [m], default `z0=0` (note: it could be an option to calculate z0 from ustar with `ustar2z0()`)
d	displacement height [m], optional, default `d=0` (i.e. no displacement)
zeta	stability parameter [-] to correct for stability effects, default `zeta=0` (i.e. no stability correction, resulting in classical logarithmic wind profile)
method	"method" for calculating stability correction function (only relevant if zeta is non-zero), default `method="ecmwf"` for using Phi_m from ECMWF-IFS

Value

data frame containing the requested heights `zs` and the calculated wind speed [m/s] there

Examples

```
zs=seq(1,100)
ustar=0.2
u_neutral=calc_windprofile(zs,ustar)
u_unstable=calc_windprofile(zs,ustar,zeta=-0.2)
u_stable=calc_windprofile(zs,ustar,zeta=0.2)
```

`calc_windspeed` *Wind Speed*

Description

Calculates wind speed (2D or 3D)

Usage

```
calc_windspeed(u, v, w = NULL)
```

Arguments

`u`	u-wind [m/s]
`v`	v-wind [m/s]
`w`	w-wind [m/s] (optional), default `w=NULL` for horizontal wind speed

Value

wind speed [m/s]

Examples

```
calc_windspeed(3,3,0.1)
```

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 [-]

Examples

```
calc_zeta(2,-1) #unstable
calc_zeta(2,1) #stable
```

count_spikes *Count spikes*

Description

Counts spikes in timeseries

Usage

```
count_spikes(ts, thresholds = c(NA, NA))
```

Arguments

`ts`	time series
`thresholds`	vector with lower and upper threshold, e.g. c(0,10)

Value

number of spikes in timeseries (i.e. values lower than lower threshold and higher than upper threshold)

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)

density2mixingratio *Conversion of density to mixing ratio*

Description

Conversion of density to mixing ratio

Usage

```
density2mixingratio(rho)
```

Arguments

`rho`	density [kg/m^3]

Value

mixing ratio of the gas [kg/kg]

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

Arguments

`ts`	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)
```

df_dx *df_dx*

Description

Calculates x-derivative for equidistant grid

Usage

```
df_dx(fld, xres = 1)
```

Arguments

fld	input field with dimension (x,y)
xres	resolution in x-direction

Value

x-derivative of fld (same dimensions)

Examples

```
set.seed(5)
field=matrix(rnorm(16),ncol=4)
df_dx(field,10)
```

df_dy *df_dy*

Description

Calculates y-derivative for equidistant grid

Usage

```
df_dy(fld, yres = 1)
```

Arguments

fld	input field with dimension (x,y)
yres	resolution in y-direction

Value

y-derivative of fld (same dimensions)

Examples

```
set.seed(5)
field=matrix(rnorm(16),ncol=4)
df_dy(field,10)
```

---

dt2dx_taylor — *Transform time (difference) to space (difference) using Taylor hypothesis*

Description

Transform time difference to space difference using Taylor hypothesis

Usage

```
dt2dx_taylor(dt, ws)
```

Arguments

dt	time (difference) [s]
ws	wind speed [m/s]

Value

space (difference) dx [m]

Examples

```
dx=dt2dx_taylor(0.1,3)
```

EC_processing_realtime — *Eddy-covariance post-processing for near-real-time analysis*

Description

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

Usage

```
EC_processing_realtime(
  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 = NULL,
  despike_co2 = NULL,
  despike_ch4 = NULL,
  do_detrending = FALSE,
  do_double_rotation = TRUE,
  do_flagging = TRUE,
  dir_blocked = c(0, 0),
  do_SNDcorrection = TRUE,
  A = 7/8,
  B = 7/8,
  do_WPLcorrection = FALSE,
  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
: logical, 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(230,300,10,2,8)

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
: logical, should the wind data be double rotated? default TRUE

do_flagging
: logical, 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
: logical, 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

do_WPLcorrection
: logical, should WPL correction be applied to gas fluxes? default FALSE

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 time of the run will be used to create a filename, only used if store=TRUE

meta
: logical, 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)

`fftfreq` *FFT frequency*

Description

Returns FFT sampling frequencies (R version of the function fft.fftfreq in numpy)

Usage

```
fftfreq(n, res = 1)
```

Arguments

`n`	length
`res`	spatial resolution (default: `res=1`)

Value

vector of length n containing FFT frequencies

Examples

```
fftfreq(10)
```

`find_closest_grid_point` *Find closest grid point*

Description

Finds the closest grid point from a given point (lon_loc, lat_loc) to a given grid (lons,lats)

Usage

```
find_closest_grid_point(lons, lats, lon_loc, lat_loc)
```

Arguments

`lons`	longitudes of a grid (vector or matrix) [deg]
`lats`	latitudes of a grid (vector or matrix) [deg]
`lon_loc`	longitude of the location [deg]
`lat_loc`	latitude of the location [deg]

Value

array index of lons, lats that represents the closest grid point to lon_loc, lat_loc

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, 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)

Examples

```
flag_distortion(1,1,dir_blocked=c(30,60))
flag_distortion(1,1,dir_blocked=c(180,360))
```

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, thresholds_most = c(0.3, 0.8))
```

Arguments

`w_sd`	standard deviation of vertical velocity
`ustar`	friction velocity
`zeta`	stability parameter `zeta = z/L`
`thresholds_most`	vector containing 2 elements to distinguish between flag=0 and flag=1, as well as flag=1 and flag=2, default: `c(0.3,0.8)`

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, thresholds_stationarity = c(0.3, 1))
```

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)`)
`thresholds_stationarity`	vector containing 2 elements to distinguish between flag=0 and flag=1, as well as flag=1 and flag=2, default: `c(0.3,1)`

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, thresholds_w = c(0.1, 0.15))
```

Arguments

`w`	vertical velocity
`thresholds_w`	vector containing 2 elements to distinguish between flag=0 and flag=1, as well as flag=1 and flag=2, default: `c(0.1,0.15)`

Value

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

Examples

```
flag_w(0.01)
```

`gapfilling` *Very basic constant/linear 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
```

---

get_amplitude_resolution
Amplitude resolution

Description

Gives amplitude resolution of time series (i.e. number of different values in time series)

Usage

```
get_amplitude_resolution(ts)
```

Arguments

ts	time series

Value

number of different values in time series

get_contours_from_f2d *Get contours from 2D flux footprint matrix*

Description

Calculates contours for given flux footprint

Usage

```
get_contours_from_f2d(x, y, fmat, contours = seq(0, 0.9, 0.1))
```

Arguments

`x`	x-coordinate (vector)
`y`	y-coordinate (vector)
`fmat`	2D flux footprint (matrix/array with the dimensions matching nx x ny)
`contours`	which contours? default `contours=seq(0,0.9,0.1)`

Value

list with x- and y-coordinates of the contours

lh2et *Evapotranspiration*

Description

Converts latent heat flux to evaporation

Usage

```
lh2et(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)]

Examples

```
lh2et(20)
lh2et(20,273)
```

`locate_flux_footprint` *Transform flux footprint from (x,y)-coordinates to (lon,lat)-coordinates through given station location*

Description

Transform flux footprint from (x,y)-coordinates to (lon,lat)-coordinates through given station location

Usage

```
locate_flux_footprint(ffp, lon_station, lat_station)
```

Arguments

`ffp`	ffp object returned by `calc_flux_footprint_[method]`
`lon_station`	lon of station location
`lat_station`	lat of station location

Value

ffp object which contains ffp also in lon-lat coordinates (with extension _earth)

Examples

```
lon1=7.527061462
lat1=60.59384155
ffp=calc_flux_footprint_Kljun2015(zm=20,ws_mean=2,blh=200,L=-1.5,v_sd=0.6,ustar=0.4,contours=0.8)
ffp=locate_flux_footprint(ffp,lon1,lat1)
```

`molarconcentration2density`
Conversion of molar concentration to density

Description

Conversion of molar concentration to density

Usage

```
molarconcentration2density(c, gas = "H2O")
```

Arguments

`c`	molar concentration in mol/m^3
`gas`	which gas? can be either `H2O`, `CO2`, `CH4`

Value

density of the gas [kg/m^3]

`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)`
`...`	arguments passed to plot function

Value

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

Examples

```
set.seed(5)
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)

Usage

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

Arguments

`ffp`	an object returned from `calc_flux_footprint_[method]`
`levels`	levels used for filled contour plot of footprint, default `levels=c(0,10^seq(-6,-3,0.1))`
`mode`	can be either `mode="distance"` for plotting footprint relative to station location in cartesian coordinates or `mode="lonlat"` for plotting in (lon,lat)-ccordinates
`...`	paraemters passed to image.plot function

Value

no return

Examples

```
ffp=calc_flux_footprint_Kljun2015(zm=5,ws_mean=5,blh=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`
`...`	arguments 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_ogive` *Plotting Ogive*

Description

Plots ogive

Usage

```
plot_ogive(ogive, ...)
```

Arguments

`ogive`	an object returned from `calc_ogive`
`...`	arguments passed to plot function

Value

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

Examples

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

plot_quadrant_analysis *Plotting Quadrant Analysis*

Description

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

Usage

```
plot_quadrant_analysis(
  xval,
  yval,
  do_normalization = TRUE,
  hole_sizes = c(1, 2, 3),
  plot_kde2d = TRUE,
  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)`
`plot_kde2d`	should the contour lines of the 2d kernel density estimation be plotted? default `plot_kde2d = TRUE`
`contours`	vector containing levels of contour lines for 2d kernel density estimation, only used if `plot_kde2d = TRUE`, 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" (molar mixing ratio) 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]

Examples

```
pres2height(60000) #using default surface values
pres2height(60000,95000,265) #adapted surface values
```

`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]

Examples

```
rh2ah(70,273)
```

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]

Examples

```
rh2q(70,273,101300)
```

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 = "ecmwf")
```

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 options: `method="B1971"` for Businger et al., 1971, and `DH1970` for Dyer and Hicks, 1970

Value

Phi_h

Examples

```
scale_phih(-1)
scale_phih(1,method="B1971")
```

`scale_phim` *Scaling function for momentum Phi_m*

Description

scaling function Phi_m

Usage

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

Arguments

`zeta`	stability parameter [-]
`method`	defining from which paper the scaling function should be used, default `method="ecmwf"` for the ones used in ECMWF-IFS, other options: `method="B1971"` for Businger et al., 1971, and `DH1970` for Dyer and Hicks, 1970

Value

Phi_m

Examples

```
scale_phim(-1)
scale_phim(1,method="B1971")
```

`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 should be 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 should be used, default `method="PD1984"` for Panofsky and Dutton, 1984

Value

Phi_w

shade_between — *Shades area between two functions*

Description

Shades area between two functions given in form of x1, x2, f1, f2

Usage

```
shade_between(x1, x2, f1, f2, ...)
```

Arguments

x1	x-coordintes of f1
x2	x-coordintes of f2
f1	y-coordintes of f1
f2	y-coordintes of f2
...	arguments passed to plot function

Value

no return

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)
```

`sigma2height` *Converts hybrid (terrain-following) sigma levels to physical heights*

Description

Converts hybrid (terrain-following) sigma levels to physical heights

Usage

```
sigma2height(hybrid, Tv = 273.15)
```

Arguments

`hybrid`	scalar or vector, hybrid sigma levels
`Tv`	virtual temperature

Value

hybrid levels converted to physical height [m]

Examples

```
sigma2height(0.1)
sigma2height(0.1,288)
```

`smaller_than_machine_epsilon` *Set everything smaller than machine epsilon to zero*

Description

Calculates machine epsilon (machine-dependent) and sets everything smaller to exactly zero

Usage

```
smaller_than_machine_epsilon(vec)
```

Arguments

`vec`	vector/time series

Value

vector of same length, just all values smaller than machine epsilon are set to exactly zero

Examples

```
ts=c(1,0.1,1e-15,1e-16,1e-17,1e-18,1e-19)
ts=smaller_than_machine_epsilon(ts)
```

`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,
  sos = csound()
)
```

Arguments

Ts_mean	sonic 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
sos	speed of sound [m/s], default `sos = csound()` corresponding to 343 m/s

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]

Ts2T	*Ts2T*

Description

Converts sonic temperature Ts to temperature T

Usage

```
Ts2T(Ts, q)
```

Arguments

Ts	sonic temperature [K] (similar as virtual temperature)
q	specific humidity [kg/kg]

Value

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]

Examples

```
ustar2z0(0.2)
```

WPLcorrection *WPL correction*

Description

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

Usage

```
WPLcorrection(
  Ts_mean,
  q_mean,
  cov_wTs,
  rhow_mean,
  cov_wrhow,
  rhoc_mean = NULL,
  cov_wrhoc = NULL
)
```

Arguments

`Ts_mean`	temperature [K] (sonic temperature or corrected temperature)
`q_mean`	specific humidity [kg/kg] (if measured, default `NULL`)
`cov_wTs`	covariance cov(w,Ts) [m/s*K]
`rhow_mean`	measured water vapor density [kg/m^3]
`cov_wrhow`	covariance cov (w,rhow) [m/s*kg/m^3]
`rhoc_mean`	measured trace gas density [kg/m^3] (only if WPL-correction should be applied to another flux, e.g. CO2 flux, default `NULL`)
`cov_wrhoc`	covariance cov (w,rhoc) [m/s*kg/m^3] (only if WPL-correction should be applied to another flux, e.g. CO2 flux, default `NULL`)

Value

WPL correction of respective flux

zeta2Ri *Converts stability parameter zeta to Richardson Ri using Businger-Dyer relations*

Description

converts zeta to Ri using Businger-Dyer relations

Usage

```
zeta2Ri(zeta)
```

Arguments

zeta	stability parameter [-]

Value

Richardson number [-]

Examples

```
Ri_transformed=zeta2Ri(0.1)
```