

# Package ‘Reddy’

September 16, 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

## R topics documented:

averaging . . . . .	1
binning . . . . .	2
calc_anisotropy . . . . .	3
calc_br . . . . .	3
calc_csi . . . . .	4
calc_dshear . . . . .	4
calc_ef . . . . .	5
calc_flux_footprint . . . . .	5
calc_gustfactor . . . . .	6
calc_iw . . . . .	7
calc_L . . . . .	7
calc_mrd . . . . .	8
calc_phim . . . . .	8
calc_phit . . . . .	9
calc_phix . . . . .	10
calc_pottemp . . . . .	10

calc_quadrant_analysis . . . . .	11
calc_ri . . . . .	12
calc_satvaporpressure . . . . .	12
calc_spectrum . . . . .	13
calc_ti . . . . .	13
calc_tke . . . . .	14
calc_ustar . . . . .	14
calc_var . . . . .	15
calc_vtke . . . . .	15
calc_windDirection . . . . .	16
calc_windSpeed2D . . . . .	16
calc_windSpeed3D . . . . .	17
calc_xstar . . . . .	17
calc_zeta . . . . .	18
cov2cf . . . . .	18
cov2lh . . . . .	19
cov2sh . . . . .	19
deaccumulate1h . . . . .	20
despiking . . . . .	20
flag_distortion . . . . .	21
flag_most . . . . .	22
flag_stationarity . . . . .	22
flag_w . . . . .	23
gapfilling . . . . .	24
plot_barycentric_map . . . . .	24
plot_flux_footprint . . . . .	25
plot_mrd . . . . .	26
plot_quadrant_analysis . . . . .	26
plot_seb . . . . .	27
ppt2rho . . . . .	28
pres2height . . . . .	29
rh2q . . . . .	29
rotate_double . . . . .	30
rotate_planar . . . . .	30
scale_phih . . . . .	31
scale_phim . . . . .	31
scale_phiT . . . . .	32
scale_phiu . . . . .	32
scale_phiw . . . . .	33
shift2maxccf . . . . .	33
SNDcorrection . . . . .	34
sos2Ts . . . . .	35
WPLcorrection . . . . .	35

---

averaging	<i>accumulating / averaging</i>
-----------	---------------------------------

---

**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	<i>discrete binning</i>
---------	-------------------------

---

**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  $(\text{length}(\text{bins}) - 1, 4)$  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	<i>Invariant analysis of Reynolds stress tensor</i>
-----------------	---

---

**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: $\text{cov}(u, v)$ (scalar or vector)
a13	R13 element of Reynolds stress tensor: $\text{cov}(u, w)$ (scalar or vector)
a22	R22 element of Reynolds stress tensor: $v_{sd}^2$ (scalar or vector)
a23	R23 element of Reynolds stress tensor: $\text{cov}(v, w)$ (scalar or vector)
a33	R33 element of Reynolds stress tensor: $w_{sd}^2$ (scalar or vector)

**Value**

list containing  $x_b, y_b, \eta, \xi$ , all eigenvalues and eigenvectors ( $\eta, \xi$  are the coordinates of the Lumley triangle and  $x_b, y_b$  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 <sup>2</sup> ]
lh	latent heat flux [W/m <sup>2</sup> ]

**Value**

Bowen ratio [-]

---

`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 <sup>2</sup> ]
rh	relative humidity [percent]
e	vapor pressure [Pa] (either rh or e have to be given)

**Value**

CSI, clear sky index

---

calc_dshear	<i>Directional Shear</i>
-------------	--------------------------

---

**Description**

Calculates a measure for directional shear  $\alpha_{uw} = \arctan(\text{cov}(v,w)/\text{cov}(u,w))$

**Usage**

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

**Arguments**

cov_uw	covariance $\text{cov}(u,w)$
cov_vw	covariance $\text{cov}(v,w)$

**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	<i>Evaporative fraction</i>
---------	-----------------------------

---

**Description**

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

**Usage**

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

**Arguments**

sh	sensible heat flux [W/m <sup>2</sup> ]
lh	latent heat flux [W/m <sup>2</sup> ]

**Value**

evaporative fraction [-]

---

`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	<i>Gust Factor</i>
-----------------	--------------------

---

**Description**

Calculates gust factor  $G := w_{s\_max}/w_{s\_mean}$

**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	<i>Vertical Turbulence Intensity I<sub>w</sub></i>
---------	--

---

**Description**

Calculates vertical turbulence intensity  $I_w = w_{sd}/w_{s\_mean}$

**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	<i>Obukhov length</i>
--------	-----------------------

---

**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	<i>Multiresolution Decomposition (MRD) according to Vickers and Mahrt, 2003</i>
----------	---

---

**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	<i>Calculates Phi_m</i>
-----------	-------------------------

---

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	<i>Calculates Phi_t</i>
-----------	-------------------------

---

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	<i>Calculates Phi_x (general flux-variance relation)</i>
-----------	--

---

Description

calculates  $\Phi_x = \sigma_x / x_{star}$  (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

$\Phi_x = \sigma_x / x_{star}$

---

calc_pottemp	<i>Potential temperature</i>
--------------	------------------------------

---

**Description**

Calculates potential temperature for given temperature and pressure

**Usage**

```
calc_pottemp(temp, pres)
```

**Arguments**

temp	temperature [K]
pres	pressure [Pa]

**Value**

potential temperature [K]

---

calc_quadrant_analysis	<i>Calculating Coherent Structures following Quadrant Analysis</i>
------------------------	--

---

**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. $(x - \text{mean}(x)) / \text{sd}(x)$ , default: do_normalization=TRUE
hole_sizes	vector containing desired hole sizes (integers $\geq 0$ )
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

*Turbulence Spectrum of Timeseries*


---

### 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_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**

<code>u_sd</code>	standard deviation of streamwise wind (u-wind)
<code>v_sd</code>	standard deviation of crosswise wind (v-wind)
<code>ws_mean</code>	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**

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

**Value**

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

---

calc_ustar	<i>Friction Velocity</i>
------------	--------------------------

---

**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	<i>Velocity Aspect Ratio (VAR)</i>
----------	------------------------------------

---

**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_vtke`*Turbulent Kinetic Energy Velocity Scale*

---

**Description**

Calculates the velocity scale of turbulent kinetic energy (TKE):  $V_{tke} = \sqrt{TKE}$

**Usage**

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

**Arguments**

<code>u_sd</code>	standard deviation of u-wind [m/s]
<code>v_sd</code>	standard deviation of v-wind [m/s]
<code>w_sd</code>	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**

<code>u</code>	u-wind [m/s]
<code>v</code>	v-wind [m/s]

**Value**

wind direction [deg]

---

calc_windSpeed2D	<i>Horizontal Wind Speed</i>
------------------	------------------------------

---

**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	<i>Wind Speed (3D)</i>
------------------	------------------------

---

**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	<i>Calculates xstar (denominator for general flux-variance relation)</i>
------------	--

---

**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	<i>Stability Parameter</i>
-----------	----------------------------

---

**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	<i>Converts cov(co2,w) to CO2 flux</i>
--------	--

---

**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**

latent heat flux [W/m^2]

---

cov2lh	<i>Converts cov(w,q) to latent heat flux LH</i>
--------	---

---

**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	<i>Converts cov(w,T) to sensible heat flux SH</i>
--------	---

---

**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]

---

deaccumulatelh	<i>deaccumulation</i>
----------------	-----------------------

---

**Description**

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

**Usage**

```
deaccumulatelh(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)
```

---

flag_distortion	<i>Flow Distortion Flag and Wind Constancy Ratio</i>
-----------------	--

---

### Description

Flow Distortion Flag according to Mauder et al., 2013: Wind coming from (pre-defined) directions blocked by the measurement device is flagged 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., <code>dir_blocked = c(30, 60)</code> )
threshold_cr	threshold for constancy ratio (default <code>threshold_cr = 0.9</code> , 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	<i>Integral Turbulence Characteristics Flag</i>
-----------	---

---

### 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(sigma_w, ustar, zeta)
```

### Arguments

sigma_w	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	<i>Stationarity Flag</i>
-------------------	--------------------------

---

**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**

<code>var1</code>	variable 1
<code>var2</code>	variable 2 (same length as <code>var1</code> , usually either <code>var1</code> or <code>var2</code> represent vertical velocity)
<code>nsub</code>	number of elements used for subsampling ( <code>nsub &lt; length(var1)</code> )

**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. $(x - \text{mean}(x)) / \text{sd}(x)$ , default: do_normalization=TRUE
hole_sizes	vector containing desired hole sizes (integers $\geq 0$ ), 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 <sup>2</sup> ] (as vector of time)
sw_out	outgoing shortwave radiation [W/m <sup>2</sup> ] (as vector of time)
lw_in	incoming longwave radiation [W/m <sup>2</sup> ] (as vector of time)
lw_out	outgoing longwave radiation [W/m <sup>2</sup> ] (as vector of time)
sh	sensible heat flux [W/m <sup>2</sup> ] (as vector of time) – if measured
lh	latent heat flux [W/m <sup>2</sup> ] (as vector of time) – if measured
gh	ground heat flux [W/m <sup>2</sup> ] (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	<i>Unit conversion of "parts-per" to density (for closed-path gas analyzer)</i>
---------	---

---

**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<sup>3</sup>]

---

pres2height	<i>Converts pressure to height (using barometric formula)</i>
-------------	---

---

**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]

---

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	<i>Planar fit rotation</i>
---------------	----------------------------

---

**Description**

Planar fit rotation (i.e., sonic coordinate system will be aligned with the mean streamlines resulting in vanishing of  $w_{\text{mean}}$ )

**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
```

---

scale_phih	<i>Scaling function for heat Phi_h</i>
------------	--

---

**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	<i>Scaling function for momentum Phi_m</i>
------------	--

---

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	<i>Scaling function for temperature Phi_T</i>
------------	---

---

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	<i>Scaling function for horizontal windspeed Phi_u</i>
------------	--

---

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	<i>Scaling function for vertical windspeed Phi_w</i>
------------	--

---

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 <code>method="PD1984"</code> for Panofsky and Dutton, 1984

**Value**

Phi\_w

---

shift2maxccf	<i>Shifting two timeseries to match maximum cross-correlation</i>
--------------	---

---

**Description**

Shifts two timeseries to match their maximum cross-correlation

**Usage**

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

**Arguments**

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

**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	<i>SND and cross-wind correction of sensible heat flux</i>
---------------	--

---

### Description

SND and cross-wind correction of sensible heat flux: converts the buoyancy flux  $\text{cov}(w, T_s)$  (based on sonic temperature  $T_s$ ) to sensible heat flux

### Usage

```
SNDcorrection(u, v, w, Ts, q = NULL, A = 7/8, B = 7/8)
```

### Arguments

u	u-wind [m/s] (levelled sonic)
v	v-wind [m/s] (levelled sonic)
w	w-wind [m/s] (levelled sonic)
Ts	temperature [K] (sonic temperature or corrected temperature)
q	specific humidity [kg/kg] (if measured by the sonic, default NULL)
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	<i>Converts speed of sound (sos) to sonic temperature</i>
--------	---

---

### 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]

---

WPLcorrection	<i>WPL correction</i>
---------------	-----------------------

---

**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