Package: foehnix (via r-universe)

May 10, 2026

Type Package

Title Objective Foehn Diagnosis

Version 0.1.6

Date 2023-06-02

Description A package providing tools to perform objective probabilistic foehn wind diagnosis and methods based on Gaussian mixture models with optional concomitants and methods to access the estimates.

License GPL-2

Encoding UTF-8

LazyData true

URL https://github.com/retostauffer/Rfoehnix

BugReports https://github.com/retostauffer/Rfoehnix/issues

Depends R (>= 3.5.0), zoo, stats, graphics, testthat

Suggests colorspace, glmnet, shiny

RoxygenNote 7.3.2

Config/pak/sysreqs cmake make libuv1-dev

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

Date/Publication 2025-05-15 18:44:07 UTC

RemoteUrl https://github.com/retostauffer/Rfoehnix

RemoteRef HEAD

RemoteSha d13d4e375bedc57ae52a7c4b08bb7197c89edc6a

Contents

`add_polygon` *Add Polygon to Plot*

Description

Helper function to plot a filled polygon based on a `zoo` time series object. Automatic handling of missing values.

Usage

```
add_polygon(x, col = "#ff0000", lower.limit = 0, ...)
```

Arguments

x	an univariate zoo time series object.
col	character, a hex color (no alpha channel), default "#ff0000" (red).
lower.limit	numeric, the lower limit used to plot the polygon. default is 0.
...	additional arguments forwarded to lines.

Details

Based on input x a filled polygon is added to the current plot. The color (col) is used to colorize the line, a lighter version of the color is used to fill the polygon by adding opacity.

If the input object x contains missing values (periods with explicit NA values) the polygon will be interrupted and gaps will be plotted.

Author(s)

Reto Stauffer

Examples

```
library("zoo")
# Create a time series object
set.seed(3)
a <- zoo(sin(1:100/80*pi) + 3 + rnorm(100, 0, 0.3), 201:300)

# Plot
par(mfrow = c(1,3))
plot(a, type = "n", main = "Demo Plot 1",
     ylim = c(-1, max(a)+1), xaxs = "i", yaxs = "i")
add_polygon(a)

# Specify lower.limit to -1 (lower limit of our ylim),
# add different color, change line style.
plot(a, type = "n", main = "Demo Plot 2",
     ylim = c(-1, max(a)+.2), xaxs = "i", yaxs = "i")
add_polygon(a, col = "#00CCBB", lower.limit = -1, lwd = 3)

# Using an "upper limit".
plot(a, type = "n", main = "Demo Plot 3",
     ylim = c(-1, max(a)+.2), xaxs = "i", yaxs = "i")
add_polygon(a, col = "#00BBFF", lower.limit = par()$usr[4L])

# Make a copy and add some missing values
b <- a
b[2:10]  <- NA
b[50:55] <- NA
b[70]    <- NA

# Plot
par(mfrow = c(1,1))
```

```
# Same as "Demo Plot 2" with the time series which
# contains missing values (b).
plot(b, type = "n", main = "Demo Plot 2 With Missing Values",
     ylim = c(-1, max(b, na.rm = TRUE)+.2), xaxs = "i", yaxs = "i")
add_polygon(b, col = "#00CCBB", lower.limit = -1, lwd = 3)
```

`center` *Extract Values used for Standardization (scaled:center)*

Description

Generic method to extract 'centering' values used for standardization (emprical first moment).

Usage

```
center(x, ...)

## S3 method for class 'standardized'
center(x, ...)
```

Arguments

`x`	object.
`...`	forwarded to generic methods.

Value

Returns 'scaled:center' used for standardization

`coef.foehnix` *Get Estimated Mixture Model Coefficients*

Description

Returns the estimated coefficients of a `foehnix` mixture model for both, the components and the concomitant model (if specified).

Usage

```
## S3 method for class 'foehnix'
coef(object, type = "parameter", ...)
```

Arguments

object	a foehnix object.
type	character, either "parameter" "coefficients", see 'Details' section.
...	additional arguments, ignored.

Details

Returns the coefficients of the mixture model. If type = "parameter" the parameters of the mixture model components are returned on the 'true' scale (parameter scale), else on the link scale (as used for optimization). For each component the location and scale parameters are returned. If a concomitant model has been specified, the coefficients of the (possibly regularized) logistic regression model are returned as well (concomitant model).

Value

Returns a coef.foehnix object with the estimated coefficients.

Author(s)

Reto Stauffer

See Also

foehnix, iwls_logit.

---

ddff2uv *Convert Wind Speed and Wind Direction to u/v Wind Components*

Description

Converts wind direction (dd) and wind speed (ff) information into u/v wind components (zonal and meridional wind component).

Usage

```
ddff2uv(dd, ff = NULL)
```

Arguments

dd	only necessary if ff is a numeric vector. Use NULL if input ff is containing both variables (ff, dd), else dd is a data vector containing the wind direction in degrees (0: north, 90: east, ...)
ff	numeric, matrix, data.frame, or zoo object (see 'Details' section).

Details

Converts data from wind speed and direction into the zonal and meridional wind components (u/v).

Different inputs are allowed:

- if ff is a matrix: requires to contains at least the two columns "ff" and "dd".
- if ff is a data.frame or zoo object: requires to contains at least the two variables "ff" and "dd".
- if ff is numeric: dd has to be provided in advance. ff and dd have to be of the same length or one has to be of length 1 (will be recycled).

Value

Returns a data.frame or zoo object (depending on input ff) containing the u and v components of the data. In addition, rad (mathematical representation of wind direction in radiant) is returned.

Author(s)

Reto Stauffer

See Also

uv2ddff

Examples

```
## Generate dd and ff variable
set.seed(0)
ff <- floor(abs(rnorm(20))*10)
dd <- sample(seq(0,359),20)
df <- data.frame('ff'=ff,'dd'=dd)

## Using with vectors
print(head(ddff2uv('dd' = dd, 'ff' = ff)))

## Using with data.frame
print(head(ddff2uv(df)))

## Using with matrix
print(head(ddff2uv(as.matrix(df))))

## Use with zoo
library("zoo")
set.seed(100)
Sys.setenv("TZ" = "UTC")
data <- zoo(data.frame(dd = runif(20, 0, 360), ff = rnorm(20, .2, 2)^2),
            as.POSIXct("2019-01-01 12:00") + 0:19 * 3600)
data <- round(data,2)
head(data)
class(data)
## Calculate dd/ff
```

```
uv <- ddff2uv(data)
head(uv)
class(uv)
```

`demodata` *foehnix Demo Data Set*

Description

The `foehnix` package comes with two sets of meteorological observations: one for Tyrol, Austria, and one for Southern California, USA. Both data sets come with observations from two stations, one valley station (or target station) and one station further upstream of the main foehn wind direction (crest station) used to filter the data (see `foehnix_filter`). For Tyrol, observations for station Ellbögen (valley) and station Sattelberg (crest) are included, the Californian data set consists of the crest station 'Lucky Five Ranch' and the valley station 'Viejas Casino and Resort'. More details are provided in the 'Details' section.

Usage

```
demodata(
  which = c("tyrol", "ellboegen", "sattelberg", "california", "viejas", "luckyfive")
)
```

Arguments

`which`	`which = "tyrol"` returns the combined Tyrolean data set (A) and `which = "california"` the combined Californian data set (USA), a time series object which contains measurements from both stations (valley/crest). Else, the data for a specific site will be returned (see 'Examples'/'Usage').

Details

The Tyrolean data set consists of station Elbögen located in the Wipp valley, an area well known for south foehn events. The second station called Sattelberg is located 18km (11.5mi) south of Ellbögen close to the main alpine ridge and is used as upstream crest station.

- Ellbögen: 11.42889 E/47.18694 N, 1080 meters above mean sea level
- Sattelberg: 11.47889 E/47.01083 N, 2107 meters above mean sea level

Hourly time series objects (class 'zoo'; UTC) including the following variables:

- `dd`: wind direction in degrees, meteorological format (0/360 correspond to 'wind from north', 90 to 'wind from east', 180 'wind from south', and 270 'wind from west')
- `ff`: wind speed in meters per second
- `p`: station pressure in hectopascal
- `rh`: relative humidity in percent
- `t`: dry air temperature in degrees Celsius

The Californian data sets consist of two stations located in the southern part of the state. Station 'Viejas' is installed next to the Viejas Casino and Resort located in Alpine, CA (valley station), the station 'Lucky Five Ranch' is located 22km (14mi) northwest of the Viejas Casino close to the main ridge of the Sierra Nevada mountain range and is used as the upstream crest station.

- Viejas Casino and Resort: -116.70437 E/32.84559 N, 715 meters above mean sea level
- Lucky Five Ranch: -116.528 E/32.9331 N, 1445 meters above mean sea level

Hourly time series objects (class 'zoo'; UTC) including the following variables:

- `air_temp`: dry air temperature in degrees Celsius
- `relative_humidity`: relative humidity in percent
- `wind_speed`: wind speed in meters per second
- `wind_direction`: wind direction in degrees, meteorological format (0/360 correspond to 'wind from north', 90 to 'wind from east', 180 'wind from south', and 270 'wind from west')
- `wind_gust`: wind gust speed in meters per second

Author(s)

Reto Stauffer, Deborah Detka

Reto Stauffer

Source

Station Ellögen operated by the Department of Atmospheric and Cryospheric Sciences (http://acinn.uibk.ac.at) of the University of Innsbruck. The data is available under the CC BY-SA 4.0 license.

Station Sattelberg operated by the Department of Atmospheric and Cryospheric Sciences (http://acinn.uibk.ac.at) of the University of Innsbruck. The data is available under the CC BY-SA 4.0 license.

Station Viejas Casino and Resort operated by San Diego Gas and Electric and made freely publicly available through Synoptic PBC at https://synopticdata.com/.

Station Lucky Five Ranch operated by San Diego Gas and Electric and made freely publicly available through Synoptic PBC at https://synopticdata.com/.

Examples

```
# Loading the combined demo data set for "Tyrol (A)".
# Stations: Ellboegen (valley station) and Sattelberg (crest station).
# Variables starting with "crest_" are observations from station Sattelberg,
x <- demodata("tyrol") # Default
print(head(x))

# Loading the combined demo data set for "California (USA)".
# Stations: Viejas (valley station) and 'Lucky Five Ranch' (crest station).
# Variables starting with "crest_" are observations from station 'Lucky Five Ranch'.
x <- demodata("california")
print(head(x))
```

```
# Sattelberg only
x <- demodata("sattelberg")
print(head(x))

# Solely Ellboegen
x <- demodata("ellboegen")
print(head(x))

# Viejas
x <- demodata("viejas")
print(head(x))

# Lucky Five Ranch
x <- demodata("luckyfive")
print(head(x))
```

`destandardize` *Destandardize a Standardized Matrix*

Description

Destandardize a Standardized Matrix

Usage

```
destandardize(x, ...)
```

Arguments

`x`	standardized matrix of class `standardized`.
`...`	forwarded, unused.

Details

Reverse function of `standardize`.

Author(s)

Reto Stauffer

See Also

`standardize`

Examples

```
# See example on R documentation for \code{?standardize}.
```

edf *Returns Effective Degrees of Freedom*

Description

Function which returns the effective degrees of freedom.

Usage

```
edf(object, ...)
```

Arguments

object	the object from which the effective degrees of freedom should be returned.
...	forwarded to class specific edf methods.

fitted.foehnix *Returns Fitted Values of foehnix Mixture Models*

Description

Extracts fitted probabilities and/or flags from a foehnix mixture model.

Usage

```
## S3 method for class 'foehnix'
fitted(object, which = "probability", ...)
```

Arguments

object	a foehnix model object.
which	what to get as return. Can a character string, one of "probability" or "flag", or "both". Alternatively integers can be used (1, 2, or c(1,2)).
...	additional arguments, ignored.

Value

Returns a univariate or multivariate zoo object.

Author(s)

Reto Stauffer

foehnix *Foehn Classification Based on a Two-Component Mixture Model*

Description

This is the main method of the foehnix package to estimate two-component mixture models for automated foehn classification.

Usage

```
foehnix(
  formula,
  data,
  switch = FALSE,
  filter = NULL,
  family = "gaussian",
  control = foehnix.control(family, switch, ...),
  ...
)

## S3 method for class 'foehnix'
logLik(object, ...)

## S3 method for class 'foehnix'
nobs(object, ...)

## S3 method for class 'foehnix'
AIC(object, ...)

## S3 method for class 'foehnix'
BIC(object, ...)

## S3 method for class 'foehnix'
IGN(object, ...)

## S3 method for class 'foehnix'
edf(object, ...)

## S3 method for class 'foehnix'
print(x, ...)

## S3 method for class 'foehnix'
formula(x, ...)

## S3 method for class 'foehnix'
summary(object, eps = 1e-04, detailed = FALSE, ...)
```

Arguments

formula	an object of class formula (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details'.
data	a regular (not necessarily strictly regular) time series object of class zoo containing the variables for the two-part mixture model.
switch	logical. If set to TRUE the two estimated components will be switched. This is important if the covariate for the components (left hand side of input formula is smaller for cases with foehn than for cases without foehn, e.g., when using the temperature difference as main covariate.
filter	a named list can be provided to apply a custom (simple) filter to the observations on data. Can be used to e.g., prespecify a specific wind sector for foehn winds. Please read the manual of the foehnix_filter method for more details and examples.
family	character (at the moment "gaussian" or "logistic") or an object of class foehnix.family.
control	additional control arguments, see foehnix.control.
...	forwarded to foehnix.control
object	a foehnix object (input for S3 methods)
x	a foehnix object (input for S3 methods)
eps	threshold for posterior probabilities used in summary. numeric value, default 1e-4.
detailed	boolean, default FALSE. If TRUE, additional information will be returned.

Details

The two-component mixture model can be specified via formula object where the left hand side of the formula contains the 'main' variable explaining the two components (only one variable), the right hand side of the formula specifies the concomitant variables (multiple variables allowed). As an example: let's assume that our zoo object data contains the following columns:

- ff: observed wind speed at target site
- rh: observed relative humidity at target site
- diff_t: (potential) temperature difference between target site and a station upstream of the foehn wind direction

The specification for formula could e.g. look as follows:

- ff ~ 1: the two components of the mixture model will be based on the observed wind speed (ff), no concomitant model (right hand side is simply 1).
- ff ~ rh: similar to the specification above, but using observed relative humidity (rh) as concomitant.
- ff ~ rh + diff_t: as above but with an additional second concomitant variable (observed temperature difference, diff_t).

- `diff_t ~ ff + rh`: using temperature difference as the main variable for the two components while `ff` and `rh` are used as concomitants. Note: in this case it will be required to set `switch = TRUE` as lower values of `diff_t` indicate less stratified conditions where the occurrence of foehn is more likely. If `switch = FALSE` the two components (foehn and no foehn) may be interchanged and the model will return "probabilities of not observing foehn".

Note that these are just examples and have to be adjusted given data availability, location, structure/names of the variables in the `data` object.

The optional input `filter` allows to specify simple or complex filters (see `foehnix_filter` for details). This allows to add additional constraints, e.g., adding a filter on the observed wind direction to ensure that only events within a specific wind sector (the main foehn wind direction) are classified as "foehn".

Value

Returns an object of class `foehnix`.

Log-likelihood sum (`numeric`).

Number of observations (`integer`) used to train the model.

Returns the Akaike information criterion (`numeric`).

Bayesian information criterion (`numeric`).

Ignorance (mean negative log-likelihood; `numeric`).

Effective degrees of freedom.

Returns the formula of the model (`formula`).

Object of class `summary.foehnix`.

Author(s)

Reto Stauffer

References

Plavcan D, Mayr GJ, Zeileis A (2014). Automatic and Probabilistic Foehn Diagnosis with a Statistical Mixture Model. *Journal of Applied Meteorology and Climatology*. **53**(3), 652–659. doi:10.1175/JAMCD130267.1

Gr\"un B, Leisch F (2007). Fitting Finite Mixtures of Generalized Linear Regressions in *R*. *Computational Statistics \& Data Analysis*. **51**(11), 5247–5252. doi:10.1016/j.csda.2006.08.014

Gr\"un B, Leisch F (2008). FlexMix Version 2: Finite Mixtures with Concomitant Variables and Varying and Constant Parameters. *Journal of Statistical Software, Articles*. **28**(4), 1–35. doi:10.18637/jss.v028.i04

Fraley C, Raftery AE (2002). Model-Based Clustering, Discriminant Analysis, and Density Estimation. *Journal of the American Statistical Association*. **97**(458), 611–631. doi:10.1198/016214502760047131

See Also

See foehnix_filter for more information about the filter option. See also: tsplot, windrose.

Foehnix family objects: foehnix.family.

S3 methods for foehnix objects: plot.foehnix, predict.foehnix, fitted.foehnix, print.foehnix, summary.foehnix, windrose.foehnix, coef.foehnix, nobs.foehnix, edf.foehnix, AIC.foehnix, BIC.foehnix, IGN.foehnix, logLik.foehnix, ...

---

foehnix.control *foehnix Two-Component Mixture-Model Control Object*

Description

Used to control the foehnix mixture models.

Usage

```
foehnix.control(
  family,
  switch,
  left = -Inf,
  right = Inf,
  truncated = FALSE,
  standardize = TRUE,
  maxit = 100L,
  tol = 1e-08,
  force.inflate = FALSE,
  glmnet.control = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

family	character specifying the distribution of the components in the mixture model. Allowed: "gaussian" and "logistic". For experts: custom foehnix.family objects can be provided as well.
switch	logical whether or not the two components should be switched. By default (switch = FALSE) the component which shows higher values of y is assumed to be the foehn cluster! Depending on what your covariate is you might need to switch the clusters (by setting switch = TRUE).
left	default is -Inf, left censoring or truncation point. See also input right and input truncated. Can be set to any finite numeric value.
right	default is Inf, right censoring or truncation point. See also input left and input truncated. Can be set to any finite numeric value.

truncated	logical. If set to TRUE truncation is used instead of censoring. This only affects the foehnix model estimate if input left and/or input right are specified.
standardize	logical flag, default is TRUE. Whether or not the model matrix for the concomitant model should be standardized for model estimation (recommended).
maxit	control argument for the iterative solvers. Default is 100L, the maximum number of iterations for the EM algorithm and the IWLS backfitting algorithm for the concomitant model. If a vector of length two is provided the first value is used for the EM algorithm, the second for the IWLS backfitting.
tol	similar as for maxit. Used to identify convergence of the iterative solvers. Default is 1e-8, if two values are provided the first will be used for the EM algorithm, the second one for the IWLS backfitting procedure. If set to -Inf maxit will be used as stopping criteria.
force.inflate	logical, default is FALSE. foehnix creates a strictly regular time series object by inflating the data set using the smallest time interval in the data set. If the inflation rate is larger than 2 the script will stop except the user forces inflation by specifying force.inflate = TRUE. See 'Details' section for more information.
glmnet.control	an object of class glmnet.control containing the arguments for the glmnet function (experimental).
verbose	logical, if set to FALSE output is suppressed.
...	currently sent to hell.

Details

Inflation: foehnix models are based on time series objects. For some methods (e.g., to create nice and easy to read time series plots and count statistics) foehnix inflates the time series object using the smallest time interval in the data set. This can, possibly, yield very large data sets. Thus, foehnix is pre-calculating the inflation rate, the fraction between the length of the inflated data set versus the length of the data set provided by the user. If this inflation rate exceeds 2 the script will raise an error!

In this case, the user is kindly asked to check if the time series object (input data). A possible scenario: a user is performing foehn diagnosis using 5 years of data from one station with 10 minute observations. This yields (neglecting leap years) 5 * 365 * 144 = 262.800 observations. Imagine that there is one incorrect observation reported one second after one of the regular 10 minute records. The smallest time increment would thus be 1 second. This would yield an inflated time series object with a total record length of 5 * 365 * 86.400 = 157.680.000. Even if only filled with missing values (NA) this will be extremely memory demanding. To avoid this, foehnix will stop in such situations. However, the user is allowed to overrule this condition by setting the force.inflate option to TRUE.

Author(s)

Reto Stauffer

See Also

foehnix, foehnix.family.

`foehnix.family` *foehnix Families for Two-Component Mixture Models*

Description

The foehnix mixture models are based on a set of families provided with this package. Currently, the package provides a two-component Gaussian and a two-component logistic mixture model and their truncated and censored versions. Custom `foehn.family` objects could also be plugged in (see 'Details' section).

Usage

```
## S3 method for class 'foehnix.family'
print(x, ...)

# Check if truncation is set
is.truncated(x, ...)

# Check if left censoring/truncation limit has been specified
has.left(x, ...)

# Check if right censoring/truncation limit has been specified
has.left(x, ...)

# Logistic mixture-model family
foehnix_logistic()

# Censored logistic mixture-model family
foehnix_clogistic(left = -Inf, right = Inf)

# Truncated logistic mixture-model family
foehnix_tlogistic(left = -Inf, right = Inf)

# Gaussian mixture-model family
foehnix_gaussian()

# Censored Gaussian mixture-model family
foehnix_cgaussian(left = -Inf, right = Inf)

# Truncated Gaussian mixture-model family
foehnix_tgaussian(left = -Inf, right = Inf)
```

Arguments

`x`	`foehnix.family` object.
`...`	additional arguments, ignored.

left	left censoring or truncation point (if the family object provides this option).
right	right censoring or truncation point (if the family object provides this option).

Details

The method foehnix allows to specify a family input argument which has to be either "gaussian" (the default) or "logistic" for now. If finite arguments for left and/or right are set as well, a censored Gaussian/logistic mixture model will be estimated (or truncated, if truncated = TRUE; see foehnix and foehnix.control).

However, feel free to develop custom family objects if needed: if a foehnix.family object is provided on family when calling foehnix this custom object will be used. For example:

- fam <- foehnix:::foehnix_cgaussian(left = 0)
- mod <- foehnix(dt ~ ff + dd + rh, data = data, family = fam)

Each 'foehn.family' object consists of a set of functions:

- optional arguments not listed here (stored inside the object environment when specified on initialization,
- left and right used for the censored and truncated Gaussian/logistic families are such examples).
- name: character, name of the family object.
- d: density function of the mixture distribution.
- p: distribution function of the mixture distribution.
- r: not required for optimization but nice for testing: returns random numbers from the mixed distribution to simulate data.
- loglik function returning the log-likelihood sum, used for model estimation (EM algorithm).
- posterior returns the (updated) posterior probabilities, used for model estimation (EM algorithm).
- theta returns the parameters of the distributions of the components of the mixture models. Used for model estimation (EM algorithm).

Examples are: 'foehnix_gaussian', 'foehnix_cgaussian', 'foehnix_tgaussian', 'foehnix_logistic', 'foehnix_clogistic', and 'foehnix_tlogistic' in 'R/families.R'.

Author(s)

Reto Stauffer

See Also

foehnix, foehnix.control.

`foehnix.noconcomitant.fit` *Fitting foehnix Mixture Model Without Concomitant Model.*

Description

Fitting method for two-component foehnix mixture model without concomitants. Typically not called directly, interfaced via `foehnix`.

Usage

```
foehnix.noconcomitant.fit(
  y,
  family,
  switch = FALSE,
  maxit = 100L,
  tol = 1e-05,
  verbose = TRUE,
  ...
)
```

Arguments

`y`	numeric vector, covariate for the components of the mixture model, dimension `N`.
`family`	object of class `foehnix.family`.
`switch`	logical whether or not the two components should be switched. By default (`switch = FALSE`) the component which shows higher values of `y` is assumed to be the foehn cluster! Depending on what your covariate is you might need to switch the clusters (by setting `switch = TRUE`).
`maxit`	positive integer, or vector of length 2 with positive integer values. Maximum number of iterations of the EM algorithm. Check manual of `foehnix.control` for more details.
`tol`	numeric, or vector of length 2 containing numeric values. Tolerance for the EM algorithm. Check manual of `foehnix.control` for more details.
`verbose`	logical, default `TRUE`. If set to `FALSE` verbose output will be suppressed.
`...`	currently unused.

Author(s)

Reto Stauffer

See Also

`foehnix.family`, `foehnix`, `foehnix.control`, `foehnix.noconcomitant.fit`, `foehnix.reg.fit`, `iwls_logit`, `coef.foehnix`.

foehnix.reg.fit *Fitting Regularized foehnix Mixture Model With Concomitant Model.*

Description

Fitting method for two-component foehnix mixture model with regularization. Typically not called directly, interfaced via foehnix.

Usage

```
foehnix.reg.fit(
  y,
  logitX,
  family,
  glmnet.control,
  switch = FALSE,
  maxit = 100L,
  tol = 1e-05,
  verbose = TRUE,
  ...
)
```

Arguments

y	numeric vector, covariate for the components of the mixture model, dimension N.
logitX	numeric matrix of dimension N x P, covariates for the concomitant model (logistic regression model matrix).
family	object of class foehnix.family.
glmnet.control	an object of class glmnet.control containing the arguments for the glmnet function.
switch	logical whether or not the two components should be switched. By default (switch = FALSE) the component which shows higher values of y is assumed to be the foehn cluster! Depending on what your covariate is you might need to switch the clusters (by setting switch = TRUE).
maxit	positive integer, or vector of length 2 with positive integer values. Maximum number of iterations of the EM algorithm and the concomitant model. Check manual of foehnix.control for more details.
tol	numeric, or vector of length 2 containing numeric values. Tolerance for the optimization (EM algorithm and concomitant model). Check manual of foehnix.control for more details.
verbose	logical, default TRUE. If set to FALSE verbose output will be suppressed.
...	currently unused.

Details

TODO: Method not yet implemented, as soon as the method itself has been written: update manual page!

Author(s)

Reto Stauffer

See Also

foehnix, foehnix.control, foehnix.noconcomitant.fit, foehnix.reg.fit, iwls_logit.

`foehnix.unreg.fit` *Fitting Unregularized foehnix Mixture Model With Concomitant Model.*

Description

Fitting method for two-component foehnix mixture model with additional concomitants. Typically not called directly, interfaced via foehnix.

Usage

```
foehnix.unreg.fit(
  y,
  logitX,
  family,
  switch = FALSE,
  maxit = 100L,
  tol = 1e-05,
  verbose = TRUE,
  ...
)
```

Arguments

y	numeric vector, covariate for the components of the mixture model, dimension N.
logitX	numeric matrix of dimension N x P, covariates for the concomitant model (logistic regression model matrix).
family	object of class foehnix.family.
switch	logical whether or not the two components should be switched. By default (switch = FALSE) the component which shows higher values of y is assumed to be the foehn cluster! Depending on what your covariate is you might need to switch the clusters (by setting switch = TRUE).

maxit	positive integer, or vector of length 2 with positive integer values. Maximum number of iterations of the EM algorithm and the concomitant model. Check manual of foehnix.control for more details.
tol	numeric, or vector of length 2 containing numeric values. Tolerance for the optimization (EM algorithm and concomitant model). Check manual of foehnix.control for more details.
verbose	logical, default TRUE. If set to FALSE verbose output will be suppressed.
...	currently unused.

Author(s)

Reto Stauffer

See Also

foehnix.family, foehnix, foehnix.control, foehnix.noconcomitant.fit, foehnix.reg.fit, iwls_logit.

---

foehnix_filter *Evaluates Data Filter Rules for foehnix Mixture Moel Calls*

Description

foehnix models allow to specify an optional foehnix_filter. If a filter is given only a subset of the data set provided to foehnix is used for the foehn classification. A typical example is a wind direction filter such that only observations (times) are used where the observed wind direction was within a user defined wind sector corresponding to the wind direction during foehn events for a specific location. However, the filter option allows to even implement complex filter rules if required. The 'Details' section contains further information and examples how this filter rules can be used.

Usage

```
foehnix_filter(x, filter, cols = NULL)

## S3 method for class 'foehnix.filter'
print(x, ...)
```

Arguments

x	object of class zoo or data.frame containing the observations.
filter	can be NULL (no filter applied), a function operating on x, or a named list with a simple filter rule (numeric of length two) or custom filter functions. Details provided in the 'Details' section.
cols	NULL or a character vector containing the names in 'x' which are not allowed to contain missing values. If NULL all elements have to be non-missing.
...	currently unused.

Details

Foehn winds often (not always) show a very specific wind direction due to the canalization of the air flow trough the local topography. The `foehnix_filter` option allows to subset the data according to a user defined set of filters from simple filters to complex filters.

These filters classify each observation (each row in `x`) as good (within filter), bad (outside filter), and ugly (at least one variable required to apply the filter was `NA`).

No filter: If `filter = NULL` no filter will be applied and the whole data set provided is used to do the foehn classification (all observations will be treated as 'good').

Simple filter rules: The filter is a named list containing one or several numeric vectors of length 2 with finite numeric values. The name of the list element defines the column of the data set (input `x`), the numeric vector of length 2 the range which should be used to filter the data. This is the simplest option to apply the mentioned wind direction filter. Examples:

- `filter = list(dd = c(43, 223))`: applies the filter to column `x$dd`. The filter classifies observations/rows as 'good' (within filter) if `x$dd >= 43 & x$dd <= 223`.
- `filter = list(dd = c(330, 30)`: similar to the filter rule above, allows to specify a wind sector going trough 0 (if dd is wind direction in degrees between `[0, 360]`). The filter classifies observations/rows as 'good' (within filter) if `x$dd >= 330 | x$dd <= 30`.
- `filter = list(dd = c(43, 223), crest_dd = c(90, 270))`: two filter rules, one for `x$dd`, one for `x$crest_dd`. The filter classifies observations/rows as 'good' (within filter) if `x$dd >= 43 & x$dd <= 223` AND `x$crest_dd >= 330 | x$crest_dd <= 30`. If an observation/row does not fulfill one or the other rule the observation/row is classified as 'bad' (outside filter), if one of `x$dd` or `x$crest_dd` is `NA` the corresponding observation/row will be classified as 'ugly'.
- Filters are not restricted to wind direction (as shown in the examples above)!

Custom filter functions: Instead of only providing a segment/sector defined by two finite numeric values (see 'Simple filter' above) a named list of functions can be provided. These functions DO HAVE TO return a vector of logical values (`TRUE` (good),`FALSE` (bad), or `NA` (ugly)) of length `nrow{x}`. If not, an error will be thrown. The function will be applied to the column specified by the name of the list element. Some examples:

- `filter = list(dd = function(x) x >= 43 & x <= 223)`: The function will be applied to `x$dd`. A vector with `TRUE`, `FALSE`, or `NA` is returned for each `1:nrow{x}` which takes `NA` if `is.na(x$dd)`, `TRUE` if `x$dd >= 43 & x$dd <= 223` and `FALSE` else. Thus, this filter is the very same as the first example in the 'Simple filter' section above.
- `filter = list(ff = function(x) x > 2.0)`: Custom filter applied to column `x$ff`. A vector with `TRUE`, `FALSE`, and `NA` is returned for each observation `1:nrow{x}` which takes `NA` if `is.na(x$ff)`, `TRUE` if `x$ff > 2.0`, and `FALSE` else.
- `filter = list(ff = function(x) ..., dd = function(x) ...)`: two filter functions, one applied to `x$ff`, one to `x$dd`. Note that observations/rows will be classified as 'ugly' if one of the two filters returns `NA`. If no `NA` is returned the observation is classified as 'good' if both return `TRUE`, and as 'bad' (outside filter) if at least one returns `FALSE`.

Complex filters: If `filter` is a function this filter function will be applied to the full input object `x`. This allows to write functions of any complexity. As an example:

- `filter = function(x) (x$dd >= 43 & x$dd <= 223) & x$ff >= 2.0`: Input `x` to the filter function is the object as provided to the `foehnix_filter` function (`x`). Thus, the different columns of the object can be accessed directly trough their names (e.g., `x$dd`, `x$ff`). A vector of length `nrow(x)` with `TRUE`, `FALSE`, and `NA` has to be returned. Only those classified as 'good' (`TRUE`) will be used for classification.

Value

Returns a vector of integers corresponding to those rows in the data set `x` which fulfill all filte criteria. If input `filter = NULL` an integer vector `1:nrow(x)` is returned.

Author(s)

Reto Stauffer

Examples

```
# Loading example data set and conver to zoo time series
# time series object (station Ellboegen).
ellboegen <- demodata("ellboegen")

# Case 1:
# -----------------
# Filter for observations where the wind direction is
# within 100 - 260 (southerly flow):
idx_south <- foehnix_filter(ellboegen, list(dd = c(100, 260)))
print(idx_south)

# Same filter but for northerly flows, taking rows with
# wind direction observations (dd) smaller than 45 or
# larger than 315 degrees:
idx_north <- foehnix_filter(ellboegen, list(dd = c(315, 45)))
print(idx_north)

par(mfrow = c(1,3))
hist(ellboegen$dd,                 xlab = "dd", main = "all observations")
hist(ellboegen$dd[idx_south$good], xlab = "dd", main = "southerly winds")
hist(ellboegen$dd[idx_north$good], xlab = "dd", main = "northerly winds")

# Case 2:
# -----------------
# A second useful option is to add two filters:
# the wind direction at the target station (here Ellboegen)
# has to be within c(43, 223), the wind direction at the
# corresponding crest station (upstream, crest of the European Alps)
# has to show southerly flows with a wind direction from
# 90 degrees (East) to 270 degrees (West).

# Loading combined demo data set
data <- demodata()

# Now apply a wind filter
```

```
my_filter  <- list(dd = c(43, 223), crest_dd = c(90, 270))
filter_obj <- foehnix_filter(data, my_filter)
print(filter_obj)

# Subsetting the 'good' rows
data <- data[filter_obj$good,]

summary(subset(data, select = c(dd, crest_dd)))
```

`foehnix_glmnet` *Wrapper Function Around glmnet Logistic Regression*

Description

The `foehnix` function allows to estimate a regularized logistic regression model for the concomitants. In case `glmnet.control` is provided the EM algorithm uses this function to estimate the regression coefficients of the concomitant model. `glmnet.control` allows to specify options forwarded to `glmnet::glmnet` except family (`"binomial"`) and intercept (`TRUE`). Depending on `glmnet.control` the AIC, BIC, or log-likelihood criterion will be used.

Usage

```
foehnix_glmnet(y, x, arg)
```

Arguments

`y`	response vector (binary)
`x`	design matrix. If a column `"(Intercept)"` exists it will be dropped (as we force glmnet to estimate an intercept).
`arg`	list of arguments forwarded to `glmnet::glmnet`.

Value

Returns an object of class `ccmodel` for `foehnix` models.

Author(s)

Reto Stauffer

glmnet.control *Control Object to Allow for Regularized Concomitant Models*

Description

foehnix allows to estimate regularized concomitant models based on the R package glmnet. The glmnet.control object controls, if specified, the glmnet input arguments.

Usage

```
glmnet.control(min = "AIC", ...)

## S3 method for class 'glmnet.control'
print(x, ...)
```

Arguments

min	a character, either "AIC", "BIC", or "loglik". Default is "AIC".
...	additional arguments forwarded to glmnet::glmnet.
x	an object of class glmnet.control (print method).

Details

Note that the two input arguments intercept and family (to function glmnet) cannot be overruled by the foehnix user. In any case a binomial logit model with intercept will be estimated.

Author(s)

Reto Stauffer

IGN *Returns the Ignorance of an Object*

Description

Returns the ignorance (mean negative log-likelihood) of the object.

Usage

```
IGN(object, ...)
```

Arguments

object	the object to be evaluated.
...	forwarded to generic methods.

See Also

foehnix

---

image *foehnix Image Plot - Hovmoeller Diagram*

Description

The default image plot of a foehnix object is a Hovmoeller diagram.

Usage

```
## S3 method for class 'foehnix'
image(
  x,
  FUN = "freq",
  deltat = NULL,
  deltad = 7L,
  col = rev(gray.colors(20)),
  contours = FALSE,
  contour.col = "black",
  ...
)
```

Arguments

x	object of class foehnix.
FUN	character string or a custom aggregation function. See 'Details' section for more information.
deltat	integer, interval in seconds for the time of day axis. Has to be a fraction of 86400 (24 hours in seconds). It NULL (default) the interval of the time series object will be used.
deltad	integer, similar to deltat, the interval in days for the grid on the x-axis. Default is 7L (aggregate to weekly values).
col	vector of colors forwarded to image.default. By default a gray scale color map is used.
contours	logical TRUE or FALSE, whether or not the concours should be added.
contour.col	color for the contour lines, only used if contours = TRUE.
...	additional arguments (see 'Details' section).

Details

Plotting a Hovmoeller diagram based on the zoo time series object of the foehnix classification. Different plot types are available. The default functions (see list below) use na.rm = TRUE.

Input FUN can be one of the following character strings:

- freq: plotting frequencies (default).
- mean: mean probability.
- occ: plotting occurrence of foehn (where probability >= 0.5).
- noocc: contrary to occ: occurrence of no foehn (probability < 0.5).

FUN can also be a custom function used for time series aggregation (see aggregate.zoo).

Additional arguments which can be set:

- xlim: limits of abscissa. Numeric vector of length 2 with Julian days (0 - 365). If the values are decreasing (e.g., xlim = c(300, 100)) the abscissa will be adjusted to show continuous data around new years eve.
- ylim: limits of ordinate. Numeric vector of length 2 with seconds (seconds of the day; 0 = 00:00:00 UTC, 86400 = 24:00:00 UTC).
- xlab, ylab, main: axis labels and title of the plot.

---

is.standardized *Check if Matrix is Standardized*

Description

Returns TRUE if input x is a standardized matrix, else FALSE.

Usage

```
is.standardized(x, ...)

## S3 method for class 'matrix'
is.standardized(x, ...)
```

Arguments

x	a matrix or standardized matrix.
...	ignored.

Value

Returns logical TRUE if matrix x is standardized, else FALSE.

See Also

standardize.

`iwls_logit` *IWLS Solver for Binary Logistic Regression Model*

Description

Iterative weighted least squares solver for a logistic regression model. Used by `foehnix.unreg.fit` to estimate the concomitant model of the two-component foehnix mixture models.

Usage

```
iwls_logit(
  X,
  y,
  beta = NULL,
  lambda = NULL,
  standardize = TRUE,
  maxit = 100L,
  tol = 1e-08,
  verbose = FALSE,
  ...
)

## S3 method for class 'ccmodel'
coef(object, which = "coef", ...)

## S3 method for class 'ccmodel'
logLik(object, ...)

## S3 method for class 'ccmodel'
AIC(object, ...)

## S3 method for class 'ccmodel'
BIC(object, ...)

## S3 method for class 'ccmodel'
edf(object, ...)

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

Arguments

`X`	model matrix including intercept (if required).
`y`	response, can be binary or probabilities (values in `]0,1[`).
`beta`	initial regression coefficients. If not set (`beta = NULL`, default), all coefficients will be initialized with `0`.

lambda	if set to NULL (default) no penalty is used during optimization. A float can be provided for regularization (ridge/L2 penalty).
standardize	logical. If set to TRUE (default) the model matrix containing the concomitant variables will be standardized.
maxit	integer, maximum number of iterations, default 100.
tol	float, tolerance for the improvement to check for convergence.
verbose	logical, if set to FALSE output is suppressed.
...	currently unused.
object	object of type ccmodel (S3 methods).
which	character, defines whether the coefficients should be returned on the original scale (which = "coef") or the standardized scale (which = "beta"; identical if standardize was FALSE).

Details

Iterative (re-)weighted least squares solver for logistic regression model. The basic call (iwls_solver(X, y)) solves the unregularized problem. Input matrix X is the design matrix containing the concomitant variables for the logistic regression model. Matrix is of dimension N x P where N is the number of of observations, P the number of concomitant variables (including the intercept, if required). If more than one column contains constant values the script will throw an error (solution no more identifiable). y is the binary response vector of length N containing 0s and 1s.

beta can be used to specify the initial regression parameters. If not set (beta = NULL; default) all parameters will be initialized with 0s.

If lambda is set (float) a ridge penalty will be added to shrink the regression parameters.

The logical option standardize controls whether or not the model matrix (covariates) should be standardized using Gaussian standardization ((x - mean(x)) / sd(x)) for all columns with non-constant data. It is recommended to use standardization (standardize = TRUE) to avoid numerical problems.

maxit and tol allow to control the iterations of the IWLS solver. maxit is the maximum number of iterations allowed. tol is used to check the log-likelihood improvements. If the improvements compared with the previous iteration falls below this tolerance the optimizer converged. If maxit is reached the solver will stop, even if not converged.

Value

Returns an object of class ccmodel (concomitant model). The object contains the following information:

- lambda value used (or NULL).
- edf effective degrees of freedom. Equal to P (ncol(X)) if no regularization is used.
- loglik final log-likelihood sum of the model.
- AIC Akaike information criteria.
- BIC Bayesian information criteria.
- converged logical flag whether or not the algorithm converged (see maxit, tol).

- beta matrix of dimension P x 1 containing the estimated and possibly standardized regression coefficients (see input standardize). If input standardize = FALSE beta == coef.
- coef matrix of dimension P x 1 containing the destandardized regression coefficients.

Author(s)

Reto Stauffer

See Also

destandardize_coefficients, standardize, is.standardized

Examples

```
# Example data set
data("airquality")
airquality <- na.omit(airquality)
airquality$Ozone <- as.numeric(airquality$Ozone > 50)

# glm model
m1 <- glm(Ozone ~ ., data = airquality, family = binomial(link = "logit"))

# Setting up model.frame, response, and model matrix
mf <- model.frame(Ozone ~ ., data = airquality)
X  <- model.matrix(Ozone ~ ., data = airquality)
y  <- model.response(mf)

# Default call
m2 <- iwls_logit(X, y)
# With standardized coefficients
m3 <- iwls_logit(X, y, standardize = TRUE)
# No early stop, stop when maxit = 100 is reached. Will through
m4 <- iwls_logit(X, y, standardize = TRUE, tol = -Inf, maxit = 100)

# Comparing coefficients
print(cbind(coef(m1), m2$coef, m3$coef, m4$coef))
```

---

plot.foehnix *foehnix Model Assessment Plots*

Description

Visual representation foehnix model optimization.

Usage

```
## S3 method for class 'foehnix'
plot(x, which = NULL, log = TRUE, ..., ask = TRUE)
```

Arguments

x	a foehnix mixture model object.
which	NULL (default), character, character string, integer, or numeric. Allowed characters: loglik (1), loglikcontribution (2), coef (3), hist (4), and posterior (5).
log	logical, if TRUE the x-axis for loglik, loglikcontribution and coef is shown on the log scale.
...	additional arguments, unused.
ask	boolean, default is TRUE. User will be asked to show the next figure if multiple figures are requested. Can be set to FALSE to overwrite the default.

Details

There are currently three different plot types.

- "loglik" shows the log-likelihood sum path trough the iterations of the EM algorithm for parameter estimation.
- "loglikcontribution" shows the log-likelihood contribution (initial value subtracted; all paths start with 0).
- coef shows the development of the (standardized) coefficients during EM optimization. Parameters of the components are shown on the real scale, the coefficients of the concomitant model (if used) are shown on the standardized scale.
- hist plots empirical histograms of the main variable for the two clusters (split at posterior probability >= 0.5). In addition, the estimated parametric clusters are shown.
- posterior creates a histogram of the posterior probabilities. point masses at 0.0 and 1.0 indicate sharp separation of the clusters (posterior probabilities close to 0 or 1). The two additional input arguments breaks (numeric vector) and freq (logical) will be forwarded to hist(...).

Author(s)

Reto Stauffer

---

predict.foehnix *Predict Method for foehnix Mixture Models*

Description

Some details.

Usage

```
## S3 method for class 'foehnix'
predict(object, newdata = NULL, type = "response", ...)
```

Arguments

object	a foehnix mixture model object.
newdata	if NULL (default) the prediction of the underlying training data set will be returned (see also fitted.foehnix). Else newdata has to be a zoo object providing the required variables which have been used for model fitting and filtering (see foehnix).
type	character, one of "response" (default), "all".
...	additional arguments, ignored.

Details

Used for prediction (perform foehn diagnosis given the estimated parameters on a new data set (newdata). If no new data set is provided (newdata = NULL) the prediction is made on the internal data set, the data set which has been used to train the foehnix mixture model. If a new data set is provided (newdata) the foehn diagnosis will be performed on this new data set, e.g., based on a set of new observations when using foehnix for operational near real time foehn diagnosis.

Note that newdata has to be a zoo object containing the required information to perform the foehnix diagnosis, namely the variables used for classification (see formula.foehnix plus the ones used to filter the data (see foehnix input argument filter).

Usually type = "response" is used which returns the foehn probabilities. If type = "all" a set of additional values will be returned, namely:

- density1 density of the first component of the mixture model.
- density2 density of the second component (foehn component) of the mixture model.
- ccmodel probability from the concomitant model.

Note that the foehn probability is simply given by:

- ccmodel * density2 / ((1 - ccmodel) * density1 + ccmodel * density2)

Value

Returns a zoo object with foehn probabilities and (if type = "all") additional information. See 'Details' section for more information.

Author(s)

Reto Stauffer

standardize *Standardize (Model) Matrix*

Description

Function to standardize the columns of a matrix, used to standardize the model matrix before estimating the regression coefficients of a generalized linear model (iwls_logit).

Destandardize coefficients. Brings coefficients back to the "real" scale if standardized coefficients are used when estimating the logistic regression model (concomitant model).

Usage

```
standardize(x, ...)

## S3 method for class 'matrix'
standardize(x, ...)

## S3 method for class 'standardized'
scale(x, ...)

## S3 method for class 'standardized'
destandardize(x, ...)

destandardize_coefficients(beta, X)
```

Arguments

x	matrix of dimension N x p.
...	additional arguments, ignored.
beta	regression coefficients estimated on standardized data.
X	object of class standardize.

Value

Returns a matrix of the same dimension as input x but with standardized data. The return object is of class c("standardized", "matrix") which comes with some handy S3 methods.

Returns 'scaled:scale' used for standardization

Returns destandardized regression coefficients, same object as input beta.

Author(s)

Reto Stauffer

See Also

standardize. Used in foehnix and iwls_logit.

Examples

```
# Example data set
data("airquality")
airquality <- na.omit(airquality)

# Create model matrix
X <- model.matrix(Ozone ~ ., data = airquality)
print(head(X))

# Standardize
S <- standardize(X)
print(head(S))

is.standardized(X)
is.standardized(S)

# Get parameters used for standardization
center(S)
scale(S)

# Destandardize
D <- destandardize(S)

# Check
all.equal(D, X, check.attributes = FALSE)
```

---

tsplot *Time Series Plot of foehnix Models*

Description

Development time series plots of estimated foehnix models. TODO: they are very specific at the moment!

Usage

```
tsplot(
  x,
  start = NULL,
  end = NULL,
  ndays = 10,
  control = tsplot.control(...),
  ...,
  ask = TRUE
)
```

Arguments

x	object of type foehnix or a list with foehnix and univariate zoo objects, see 'Details'.
start	POSIXt object or an object which can be converted to POSIXt.
end	POSIXt object or an object which can be converted to POSIXt.
ndays	integer, number of days used when looping trough the time series.
control	an object of class tsplot.control.
...	additional arguments forwarded to tsplot.control. can be used to rename varaiables and to change the look of the time series plot. Please see 'Examples' and the manual of the function tsplot.control for more details.
ask	logical, default is TRUE.

Details

Development method to access plausability of the estimated foehn proabilities. For software release this method should either be removed or made much more general. At the moment the method heavily depends on the names of the data used as input for foehnix.

This time series plotting function creates a relatively specific plot and also expects, by default, a set of default variables and variable names. This function uses the data set provided on the data input argument when calling the foehnix method. As they might differ from the foehnix defaults the varnames input argument allows to specify custom names. The ones expected:

- t: dry air temperature
- crest_t: dry air temperature crest
- rh: relative humidity (in percent)
- crest_rh: relative humidity crest (in percent)
- diff_t: temperature difference between the crest and the valley station
- dd: meteorological wind direction ([0, 360])
- ff: wind speed

Custom names can be specified by renaming the defaults based on a named list. As an example: assume that wind direction is called winddir and wind speed is called windspd in your data set. Specify the following input to rename/respecify the defaults:

- varnames = list(dd = "winddir", ff = "windspd")

Please note: if a variable cannot be found (no matter whether the default variable names have been renamed or not) this specific variable will be ignored. Subplots will not be displayed if no data are available at all.

TODO: describe input 'x'

Author(s)

Reto Stauffer

See Also

tsplot.control.

Examples

```
# Loading demo data for Tyrol (Ellboegen and Sattelberg)
data <- demodata("tyrol")
filter <- list(dd = c(43, 223), crest_dd = c(90, 270))

# Create foehnix foehn classification model, provide full data
# set with all parameters
mod1 <- foehnix(diff_t ~ ff + rh, data = data,
                filter = filter, switch = TRUE, verbose = FALSE)

# Create foehnix foehn classification model, provide full data
# set with all parameters
sub  <- subset(data, select = c(rh, ff, dd))
mod2 <- foehnix(ff ~ rh, data = sub, filter = list(dd = c(43, 223)),
                verbose = FALSE)

# Plotting the time series of the a period in 2018
tsplot(mod1, start = "2018-03-01", end = "2018-03-10")

# The same for the second model which is based on a subset
# of the observation data and only includes 'ff' (wind speed)
# 'dd' (wind direction), and 'rh' (relative humitity) of the
# target station. Thus, only a subset of all subplots will be shown.
tsplot(mod2, start = "2018-03-01", end = "2018-03-10")

# To compare the estimated foehn probabilities of both models,
# both 'foehnix' objects can be provided as a list. The plots
# are based on the data of the first object (mod1), the probabilities
# of the second 'foehnix' model will be added in the last subplot.
tsplot(list(mod1, mod2), start = "2018-03-01", end = "2018-03-10")

# If the input is a named list, the names are used for the legend
tsplot(list("full model" = mod1, "subset model" = mod2), start = "2018-03-01", end = "2018-03-10")

# The first element in the list has to be a 'foehnix' object,
# but as additional inputs univariate time series with probabilities
# (in the range [0,1]) can be provided. This allows to compare
# 'foehnix' classification against other classification algorithms,
# if available.
probs <- fitted(mod2) # Time series of probabilities of 'mod2'
tsplot(list("full model" = mod1, "zoo" = probs), start = "2018-03-01", end = "2018-03-10")

# Additional arguments can be provided to
# - use differt names in the time series plots
# - change the look of the plot.
# Some examples

# Imagine that the variable names in the data set have the
```

```
# following names:
# - winddir: wind direction
# - windspd: wind speed
data <- demodata("ellboegen")
names(data)[which(names(data) == "dd")] <- "winddir"
names(data)[which(names(data) == "ff")] <- "windspd"

# Estimate a foehnix model
mod3 <- foehnix(windspd ~ rh, data = data, filter = list(winddir = c(43, 223)), verbose = FALSE)

# The time serie plot expects wind speed and wind direction
# to be called 'dd' and 'ff', but they can be renamed. If only
# a string is provided (e.g., dd = "winddir") this specifies
# the name of the variable in the data set ('data').
tsplot(mod3, dd = "winddir", ff = "windspd", start = "2018-03-01", end = "2018-03-10")

# For each element a list can be provided:
# - 'name': new name of the variable
# - 'color': color used for plotting
# - 'label': label for the axis on the plot
# See also ?tsplot.control for more information.
tsplot(mod3, dd = list(name = "winddir", col = "cyan",    ylab = "WIND DIRECTION LABEL"),
             ff = list(name = "windspd", col = "#FF00FF", ylab = "WIND SPEED LABEL"),
             rh = list(col = 4, ylab = "RELHUM LABEL"),
             t  = list(col = 5, ylab = "TEMPERATURE LABEL"),
             prob = list(col = "yellow", ylab = "PROBABILITY LABEL"),
             start = "2018-03-01", end = "2018-03-10")
```

`tsplot.control` *foehnix Time Series Plot Config*

Description

The foehnix package comes with a default time series plotting function (see `tsplot`) with a pre-specified behavior regarding variable names, colors, and labels. The `tsplot` function, however, allows to set some custom specifications, e.g., if the names of your observations (variable name of the observations in the time series object used to train the `foehnix` model) are different, if different labels are required, or if you do not like our pretty colors! This function returns a control object for the time series plots for customization.

Usage

```
tsplot.control(style = c("default", "bw", "advanced"), windsector = NULL, ...)

tsplot_get_control(x, var, property, args = list())
```

Arguments

style	character, name of the style template (default, advanced, bw) or path to a file containing the required information.
windsector	vector or list to highlight specific wind sectors. See foehnix:::windsector_convert for details
...	a set of named inputs to overwrite the defaults. see 'Details' section.
x	a tsplot.control object.
var	used when calling the tsplot_get_control function. Name of the (original!) variable name
property	the property which should be returned by tsplot_get_control
args	list of named arguments used when calling tsplot_get_control

Details

By default the tsplot function expects that the variable names are called

- t: dry air temperature (degrees Celsius)
- crest_t: dry air temperature crest (degrees Celsius)
- rh: relative humidity (percent)
- crest_rh: relative humidity crest (percent)
- diff_t: temperature difference to a nearby crest station
- dd: wind direction (meteorological degrees)
- dd: wind direction crest (meteorological degrees)
- ff: wind speed (meters per second)
- crest_ff: wind speed crest (meters per second)
- ffx: gust speed (meters per second)
- crest_ffx: gust speed crest (meters per second)

Different style presets are available. Styles can be accessed using the style input argument. Different styles also enable/disable specific observations (e.g., the "default" style supresses the plotting of crest-station observations, even if present).

Can be set to NULL to be disabled. If the variable exists in the data set but was (manually) set to NULL it will be neglected during plotting. E.g., tsplot(mod, crest_ff = NULL, crest_ffx = NULL) to disable crest station wind speed and gust speed (on plot).

If tsplot can find these variables, it will plot the corresponding observations, label the plots, and uses a set of default colors. The tsplot.control function allows to overrule the defaults by specifying custom values, e.g.: imagine that your wind speed variable is not ff but windspd. You can easily tell tsplot that it has to use this custom name by calling tsplot as follows:

- tsplot(x, ff = "windspd")

If your wind speed variable is not even in meters per second but knots, and you dislike our default color, you can also provide a list to overwrite the default name, default color and default label by calling:

- `tsplot(x, ff = list(name = "windspd", color = "#0000ff", label = "wind speed in knots")`

In the same way it can be used to overrule the defaults for all other variables used in the time series plotting function (see list above).

Value

Returns an object of class `c("tsplot.control", "data.frame")` with the specification for the time series plot.

Author(s)

Reto Stauffer

`uv2ddff` *Convert u and v Wind Components to Wind Speed and Wind Direction*

Description

Takes u/v wind components (zonal and meridional wind components) as input and returns wind speed and meteorological wind direction.

Usage

```
uv2ddff(u, v = NULL, rad = FALSE)
```

Arguments

`u`	`numeric` vector with u components or a `data.frame` or `zoo` object containing a column named `u` and a column named `v`.
`v`	`NULL` (default) or `numeric` vector. If input `u` is a single vector `v` has to be specified.
`rad`	`logical`, default is `FALSE`. Returns wind direction in radiant rather than degrees.

Details

Note: if both, `u` and `v` are provided they do have to be of the same length OR one of them has to be of length 1. The one with length 1 will be recycled.

Value

Returns a `data.frame` or `zoo` object (depending on input `u`) with two columns named `dd` and `ff` containing wind speed (same physical unit as input u/v) and wind direction. Wind direction is either in meteorological degrees (0 from North, from 90 East, 180 from South, and 270 from West) or in mathematical radiant if input `rad = TRUE`.

Author(s)

Reto Stauffer

See Also

ddff2uv

Examples

```
## Generate data.frame with u/v components for all 4 main wind directions
data <- data.frame(name = c("N","E","S","W"),
                   u    = c( 0, -1 , 0,  1 ),
                   v    = c(-1,  0,  1,  0 ))
## Use data.frame input
ddff <- uv2ddff(data)
cbind(data, ddff)
## Use u/v components as two separate vector inputs
ddff <- uv2ddff(data$u, data$v)
cbind(data, ddff)
## Radiant
ddff <- uv2ddff(data, rad = TRUE)
cbind(data, ddff)

## Use with zoo
library("zoo")
set.seed(100)
Sys.setenv("TZ" = "UTC")
data <- zoo(data.frame(u = rnorm(20, 2, 5), v = rnorm(20, 0, 4)),
            as.POSIXct("2019-01-01 12:00") + 0:19 * 3600)
head(data)
class(data)
## Calculate dd/ff
ddff <- uv2ddff(data)
head(ddff)
class(ddff)
```

---

windrose *Windrose Plot for foehnix Mixture Models and Observations*

Description

foehnix Mixture Model Windrose Plot

Usage

```
windrose(x, ...)

## S3 method for class 'foehnix'
windrose(
  x,
  type = NULL,
  which = NULL,
```

```
  ddvar = "dd",
  ffvar = "ff",
  breaks = NULL,
  ncol = NULL,
  maxpp = Inf,
  ...,
  main = NULL
)
```

Arguments

x	object of class foehnix if the windrose plot is applied to a model, or a vector of wind directions for windrose.default (see 'Details' section).
...	forwarded to windrose.default.
type	NULL or character, one of density or histogram. If NULL both types will be plotted.
which	NULL or character, one of unconditional, nofoehn, or foehn. If NULL all three will be plotted.
ddvar	character, name of the column in the training data set which contains the wind direction information.
ffvar	character, name of the column in the training data set which contains the wind speed (or gust speed) data.
breaks	NULL or a numeric vector to define the wind speed (ff) breaks.
ncol	integer, number of columns of subplots.
maxpp	integer (>0), maximum plots per page. Not all plots fit on one page the script loops trough.
main	NULL (default) or character. If NULL the function will add default figure titles.

Details

Windrose plot based on a foehnix mixture model object.

Allows to draw windrose plots from foehnix mixture model object or a set of observations. If input x to windrose is an object of class foehnix (as returned by foehnix) the data set which the classification is based on is used to plot the windrose. If inputs dd and ff are given (univariate zoo time series objects or numeric vectors) windrose plots can be plotted for observations without the need of a foehnix object.

Two types are available: circular density plots and circular histograms. If the input argument x is a foehnix object an additional input argument which is available to specify the subset which should be used to create the windrose plots. The following inputs are allowed:

- which = "unconditional": unconditional windrose (all observations of the data set used to estimate the foehnix model).
- which = "nofoehn": windrose of all observations which have not been classified as foehn (probability < 0.5).
- which = "foehn": windrose of all observations classified as foehn events (probability >= 0.5).

- which = NULL: all three subsets will be plotted (individual windroses).

Specific combinations can be specified by calling the windrose function with e.g., type = "histogram" and which = c("foehn", "nofoehn") (will show histograms for foehn and no foehn events), or type = NULL and which = "foehn" (will show density and histogram plot for foehn events). By default (type = NULL, which = NULL) all combinations will be plotted.

If dd and ff are set only the type argument is available (type = "histogram" or type = "density").

Author(s)

Reto Stauffer

Examples

```
# Loading combined demo data set
data <- demodata("tyrol") # default

# Before estimating a model: plot a wind rose for all observations
windrose(data$dd, data$ff, type = "histogram")
windrose(data$dd, data$ff, type = "density")

# Estimate a foehnix foehn classification model
filter <- list(dd = c(43, 223), crest_dd = c(90, 270))
mod <- foehnix(diff_t ~ ff + rh, data = data,
               filter = filter, verbose = FALSE)

# Plotting wind roses
windrose(mod)

# Only density windrose for foehn events
windrose(mod, type = "density", which = "foehn")

# Using custom names: by default wind direction is expected
# to be called 'dd', wind speed is expected to be called 'ff'.
# However, ddvar and ffvar allow to change that (only if
# windrose is called with a foehnix object as input).
# An example:
# - make a copy of data to data2
# - rename dd to winddir, crest_dd to crest_winddir
# - estimate the same foehnix model as above using the
#   new variable names
# - plot windrose with custom names for wind direction (winddir)
#   and wind speed (windspd).

data2 <- data
names(data2)[which(names(data2) == "ff")]       <- "windspd"
names(data2)[which(names(data2) == "dd")]       <- "winddir"
names(data2)[which(names(data2) == "crest_dd")] <- "crest_winddir"
print(head(data2))

filter2 <- list(winddir = c(43, 223), crest_winddir = c(90, 270))
mod2 <- foehnix(diff_t ~ windspd + rh, data = data2,
```

```
                filter = filter2, verbose = FALSE)

windrose(mod2, type = "density", which = "foehn",
         ddvar = "winddir", ffvar = "windspd")
```

`windrose.default` *Default Wind Rose Plot*

Description

Windrose plot, can handle two type of plots (see `type`) using the same information.

Usage

```
## Default S3 method:
windrose(
  x,
  ff,
  interval = 10,
  type = "density",
  windsector = NULL,
  windsector.col = "gray90",
  main = NULL,
  hue = c(10, 100),
  power = 0.5,
  ...,
  dd = NULL,
  filter = NULL,
  ffvar = "ff",
  ddvar = "dd",
  breaks = NULL,
  legend.pos = c("right", "left"),
  legend.title = NULL,
  labels.angle = 225
)
```

Arguments

`x`	either an univariate object (`numeric` vector or `zoo`) containing meteorological wind directions or a multivariate `zoo` or `data.frame` object containing both, wind speed and wind meteorological wind direction. More details provided in the 'Details' section.
`ff`	zoo object or numeric vector with sind speed data (`>= 0`).
`interval`	numeric, a fraction of 360. If 360 cannot be devided by `interval` without a rest the script will stop. Interval for the segments along the wind direction `dd`. Default `10`.

type	NULL or character, one of density or histogram. If NULL both types will be plotted.
windsector	list, matrix, or data frame for highlighting one or multiple wind sectors. See 'Examples' ('Highlighting wind sectors').
windsector.col	color of the wind sector (if provided).
main	character or expression, figure title. If not specified, a default one will be used.
hue	numeric vector of length 1 or two (for type = "density" two are used, for type = "histogram" only the first one will be used at the moment). Hue(s) for colorspace::heat_hcl(...).
power	numeric >0, power parameter for the alpha channel if type = "histogram". If 1 the alpha channel is linear for the whole range of ff, values != 1 change the alpha behavior.
...	additional optional arguments forwarded, see 'Details'.
dd	can be used if univariate objects or vectors are provided for wind speed and meteorological wind direction (see 'Details' section).
filter	a custom set of filter rules (see foehnix_filter).
ffvar	string, custom name of the wind speed variable in x if x is a multivariate object (see 'Details' section).
ddvar	string, custom name of the wind direction variable in x if x is a multivariate object (see 'Details' section).
breaks	NULL or a sequence of numeric values to set the breaks along ff.
legend.pos	character, either right (default) or left. Where to put the legend.
legend.title	character or expression to set legend title.
labels.angle	numeric, default is 225. Where to draw the labels in the density plot. labels.angle is in meteorological degrees.

Details

The windrose function can be used in different ways. The main purpose is to plot (conditional) wind roses for foehnix objects (uses windrose.foehnix), but there is this generic windrose.default method which can be used to create wind roses on non-foehnix objects.

windrose.default can either with called with two univariate inputs for wind direction and wind speed, or with a multivariate zoo object or data.frame which provides both, wind speed and wind direction. Moreover, additional variables which can be used in combination with the filter option. Examples are provided in the example section.

Univariate inputs/vectors: the windrose.default function requires both, wind direction and wind speed. If univariate objects/vectors are used at least the two inputs dd (identical to x) and ff have to be specified. dd (x) is the meteorological wind direction in degrees (]0, 360[, 0/360 is 'wind from north', 90 'wind from east', 180 'wind from south', and 270 'wind from west'). ff has to be of the same length as dd containing the corresponding wind speed.

Multivariate input: rather than providing dd (x) and ff a multivariate zoo object or data.frame can be provided when calling the function containing (at least) wind direction and wind speed. By

default, the method expects the wind direction variable to be named "dd", the wind speed named "ff". Custom names can be specified using the two input arguments ffvar and ddvar.

Custom filter: the optional filter input can be used if input x is a multivariate object and provides a convenient way to subset/filter the data. A detailed description how to define the filter rules can be found on the documentation page of the foehnix_filter method.

Additional optional arguments: The function allows to provide additional arguments for better customization. These arguments are forwarded to selected calls. Currently implemented:

- For type = "density": border, lty, and lty forwarded to polygon(...).

Examples

```
# Loading observation data for station Ellboegen.
# The object returned is a zoo time series object
# containing (see ?ellboegen).
data <- demodata("ellboegen")
class(data)
head(data)

# Extract dd/ff and create a windrose,
# using two vectors as input:
dd <- as.vector(data$dd)
ff <- as.vector(data$ff)
windrose(dd, ff,
         main = "Demo Windrose\nUse vector inputs")
windrose(dd = dd, ff = ff,
         main = "Demo Windrose\nUse vector inputs")

# Using univariate zoo objects as input:
windrose(dd = data$dd, ff = data$ff,
         main = "Demo Windrose\nUnivariate zoo objects as inputs")

# Or specify a multivariate zoo object or data.frame
# object on input x:
windrose(data,
         main = "Demo Windrose\nUse multivariate zoo object")
windrose(as.data.frame(data),
         main = "Demo Windrose\nUse multivariate data.frame object")

# Custom names for ff/dd
copy <- data
names(copy)[1:2] <- c("wind_dir", "wind_spd")
windrose(copy, main = "Demo Windrose\nMultivariate zoo, custom names",
         ddvar = "wind_dir", ffvar = "wind_spd")

# Highlighting wind sectors
windrose(data, windsector = list(c(43, 223)),
         main = "Windrose\nUnnamed Wind Sector")
windrose(data, windsector = matrix(c(43, 223), nrow = 1),
         main = "Windrose\nUnnamed Wind Sector")
windrose(data, windsector = data.frame(from = 43, to = 223),
         main = "Windrose\nUnnamed Wind Sector")
```

```
windrose(data, windsector = list(A = c(100, 160), B = c(310, 10)),
         main = "Windrose\nNamed Wind Sector")
sectors <- matrix(c(100, 160, 310, 10), nrow = 2, byrow = TRUE,
                  dimnames = list(c("Down", "Up"), NULL))
windrose(data, windsector = sectors,
         main = "Windrose\nUnnamed Wind Sector")
sectors <- matrix(seq(0, 350, by = 10), ncol = 2, byrow = TRUE)
windrose(data, windsector = sectors, main = "Yey")

# Custom filter: for details, see ?foehnix_filter
# Example 1:
# - Plot windrose for all observations where the wind
#   direction was within 43-233 degrees (south southeast)
filter1 <- list(dd = c(43, 223))
windrose(data, main = "Custom filter on wind direction,\n43 - 223 degrees",
         type = "hist", filter = filter1)

# Example 2:
# - Plot windrose for all observations where the wind
#   speed was > 5 meters per second.
filter2 <- list(ff = function(x) x > 10)
windrose(data, main = "Custom filter on wind speed\nonly observations where ff > 5",
         type = "hist", filter = filter2)

# Example 3:
# - Plot windrose for specific months (create new variable
#   'month' in advance):
data$month <- as.POSIXlt(index(data))$mon + 1
par(mfrow = c(1,2))
windrose(data, main = "Wind rose for January",
         filter = list(month = function(x) x == 1))
windrose(data, main = "Wind rose for July",
         filter = list(month = function(x) x == 7))

# Example 4:
# Similar to example 3, but for
data$hour <- as.POSIXlt(index(data))$hour
par(mfrow = c(1,2))
windrose(data, main = "Wind rose for midnight (00 UTC)",
         filter = list(hour = function(x) x == 0))
windrose(data, main = "Wind rose for noon (12 UTC)",
         filter = list(hour = function(x) x == 12))

# Example 5:
# A more complex filter: midnight, winter (Dez/Jan/Feb)
# for wind speeds exceeding 5 meters per second.
filter5 <- function(x) x$month %in% c(12, 1, 2) & x$hour == 0 & x$ff > 5
par(mfrow = c(1,2))
windrose(data, main = "DJF 12 UTC (ff > 5)\nComplex Filter", filter = filter5)
windrose(data, main = "DJF 12 UTC (ff > 5)\nComplex Filter", filter = filter5, type = "hist")
```

windrose_add_windsector *Adding Wind Sector for Windrose Plots*

Description

The windrose plot allows to specify one or multiple segments to be highlighted on the plot. This is a helper function to place the segments.

Usage

```
windrose_add_windsector(x, ff, col, offset = 0.02)
```

Arguments

x	list containing vectors of length 2 or a matrix with two rows. If the list is named, or the matrix has row names, the names will be used to label the segments.
ff	wind speed (in case of plot type "density", or density in case of plot type "histogram". Used to draw the polygon and to plot the labels (if needed).
col	color of the wind sector.
offset	offset for text adjustment.

Author(s)

Reto Stauffer

windrose_get_cols *Get Color Matrix and Legend for Windrose Plot*

Description

The windrose method provides one windrose/wind count plot type. Based on the count matrix (windrose_get_counts) this function returns a matrix with hex colors (with alpha channel) and a data.frame used for the legend.

Usage

```
windrose_get_cols(x, col, p = 1, ncol = 50L)
```

Arguments

x	2-D matrix with counts (see windrose_get_counts).
col	single hex color, or a vector of hex colors. Alpha channel will be removed.
p	numeric value, power parameter for non-linear color transformation.
ncol	integer, if col is a single value the color will be repeated ncol times. Default is 50L.

Value

Returns a list with two elements: `colormatrix` of the same dimension as `x` with colors for the different bins based on the counts, and `legend` with levels and colors for the color legend of the plot.

`windrose_get_counts` *Get Count Matrix for Windrose Plot*

Description

The `windrose` method provides one windrose/wind count plot type. This function returns a matrix with counts for different multivariate bins (binning along wind direction dd and wind speed ff).

Usage

```
windrose_get_counts(
  x,
  dd.breaks = seq(0, 360, by = 30),
  ff.breaks = pretty(x$ff)
)
```

Arguments

`x`	data object of type zoo or data.frame. Needs to contain at least the two columns `dd` (meteorological wid direction in degrees, `]0, 360[`) and `ff` with wind speed (range >= 0).
`dd.breaks`	numeric vector with breaks along wind direction. Default is `seq(0, 360, by = 30)`.
`ff.breaks`	numeric vector with breaks along wind speed. default is `pretty(x$ff)`.

Value

Returns a matrix of dimension `length(dd.breaks)` x `length(ff.breaks)` with counts >= `0`.

`windrose_get_density` *Get Density Matrix for Windrose Plot*

Description

The `windrose` method provides one 'density wind rose' method (the classical wind rose plot). This function returns a matrix with densities for different multivariate bins (binning along wind direction dd and wind speed ff).

Usage

```
windrose_get_density(
  x,
  dd.breaks = seq(0, 360, by = 30),
  ff.breaks = pretty(x$ff)
)
```

Arguments

x	data object of type zoo or data.frame. Needs to contain at least the two columns dd (meteorological wid direction in degrees,]0, 360[) and ff with wind speed (range >= 0).
dd.breaks	numeric vector with breaks along wind direction. Default is seq(0, 360, by = 30).
ff.breaks	numeric vector with breaks along wind speed. default is pretty(x$ff).

Value

Returns a matrix of dimension length(dd.breaks) x length(ff.breaks) with densities >= 0.

---

windsector_convert *Convert input "windsector" to list if needed*

Description

Converts user inputs to list. Wind sector information is used to highlight specific wind sectors on the plots (e.g., tsplot, windrose).

Usage

```
windsector_convert(x)
```

Arguments

x	NULL, list, matrix, or data.frame. see 'Details' for more information.

Details

Some foehnix functions allow to specify custom wind sectors. This function is a convenience function which takes inputs in different formats (e.g., as matrix, data.frame, ...) and converts the information in the format the foehnix functions expect wind sector definitions.

Value

Returns a list object with the windsector information as used by the different functions and methods of the foehnix package.

Returns NULL (if input was NULL) or a named or unnamed with one or multiple entries. Each entry is a numeric vector of length two and specifies a wind sector.

Author(s)

Reto Stauffer

Examples

```
# No wind sector (NULL) simply returns NULL
foehnix:::windsector_convert(NULL)

# Input can also be:
# - unnamed list
# - a matrix
# - a data.frame without row names
foehnix:::windsector_convert(matrix(c(10, 30, 90, 140), byrow = TRUE, ncol = 2))
foehnix:::windsector_convert(list(c(10, 30), c(90, 140)))
foehnix:::windsector_convert(data.frame(from = c(30, 90), to = c(30, 140)))

# Or named objects can be used
# - named list
# - matrix with rownames
# - data.frame with rownames
# If names are set these names will be used to label the
# highlighted wind sectors.
foehnix:::windsector_convert(matrix(c(10, 30, 90, 140), byrow = TRUE, ncol = 2,
                         dimnames = list(c("A", "B"), c("from", "to"))))
foehnix:::windsector_convert(list(A = c(10, 30), B = c(90, 140)))
foehnix:::windsector_convert(structure(data.frame(from = c(30, 90), to = c(30, 140)),
                                       row.names = c("foo", "bar")))
```

---

write.csv *Data Output*

Description

Generic method to write data to CSV. By default this method falls back to `utils::write.csv(...)`.

Usage

```
write.csv(...)
```

Arguments

`...`	arguments passed to generic method.

See Also

foehnix()

`write.csv.foehnix` *Write Estimated Probabilities to CSV File*

Description

Write the results of a `foehnix` model into a CSV text file. Custom date/time information can be specified using the input argument `format`. By default UNIX timestamp will be used.

Usage

```
## S3 method for class 'foehnix'
write.csv(x, file, info = TRUE, format = NULL, ...)
```

Arguments

`x`	a `foehnix` object
`file`	character, name of the target file
`info`	logical, whether or not header information should be written
`format`	NULL or a character to specify the format in which the date/time information should be written to the `file` (forwarded to `strftime`)
`...`	currently ignored

Value

Invisible return of the data.frame written to the output file.

Author(s)

Reto Stauffer

Index