Package 'EWSmethods'

January 11, 2024

Type Package

Title Forecasting Tipping Points at the Community Level

Version 1.2.5

Maintainer Duncan O'Brien <duncan.a.obrien@gmail.com>

Description Rolling and expanding window approaches to assessing abundance based early warning signals, non-equilibrium resilience measures, and machine learning. See Dakos et al. (2012) <doi:10.1371/journal.pone.0041010>, Deb et al. (2022) <doi:10.1098/rsos.211475>, Drake and fen (2010) <doi:10.1038/nature09389>, Ushio et al. (2018) <doi:10.1038/nature25504> and Weinans et al. (2021) <doi:10.1021-87839-y> for methodological details. Graphical presentation of the outputs are also provided for clear and publishable figures. Visit the 'EWSmethods' website for more information, and tutorials.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Imports curl, dplyr (>= 1.0.6), egg, ggplot2, gtools, forecast, foreach, infotheo, mAr, moments, rEDM (>= 1.15.0), reticulate, scales, tidyr, zoo

RoxygenNote 7.2.3

URL https://github.com/duncanobrien/EWSmethods, https://duncanobrien.github.io/EWSmethods/

BugReports https://github.com/duncanobrien/EWSmethods/issues

Suggests devtools, doParallel, knitr, fs, parallel, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Depends R (>= 3.5)

Config/testthat/edition 3

NeedsCompilation no

Author Duncan O'Brien [aut, cre, cph] (<https://orcid.org/0000-0002-3420-5210>), Smita Deb [aut] (<https://orcid.org/0000-0001-7037-7055>),

Sahil Sidheekh [aut],
Narayanan Krishnan [aut],
Partha Dutta [aut] (<https://orcid.org/0000-0001-6067-1023>),
Christopher Clements [aut] (<https://orcid.org/0000-0001-5677-5401>)

Repository CRAN

Date/Publication 2024-01-11 08:50:07 UTC

R topics documented:

`CODrecovery` *Three Recovering Cod Populations*

Description

A dataset containing three simulated cod populations. Community 1 does not recovery whereas Community 100 and 200 do.

Usage

```
CODrecovery
```

Format

A list of three dataframes with 191 rows and 6 variables each:

community_id the identity of the simulated community

time time index

biomass population total biomass

mean.size average length of cod individuals

sd.size variation in length of cod individuals

inflection_pt the time index where transition occurs

Source

Clements C., McCarthy M., Blanchard J. (2019) Early warning signals of recovery in complex systems. Nature Communications, 10:1681.

Examples

```
data("CODrecovery", package = "EWSmethods")

cod_data <- CODrecovery$scenario2
```

---

conda_clean *Python Removal*

Description

Removes `ewsnet_init()` downloaded Anaconda binaries and environments.

Usage

```
conda_clean(envname, conda_path = reticulate::miniconda_path(), auto = FALSE)
```

Arguments

`envname`	A string naming the desired Python environment to remove.
`conda_path`	The location of Python install. By default, this follows `minicondata_path` defined by the `reticulate` package.
`auto`	Boolean. If `FALSE`, asks permission to uninstall Python, packages and specified environment. If `TRUE`, no user confirmation is required for activation.

Value

Does not return an object as is cleaning Python and its environments.

Examples

```
#Prior to running `conda_clean()`, you must restart
#your R session to detach any activated environments

conda_clean("EWSNET_env", auto = TRUE)
```

`default_weights_path` *Path to Model Weights*

Description

The default path for EWSNet model weights to use. Is the location of the EWSmethods package. If you'd like to instead set your own path, `ewsnet_reset()` contains the argument `weights_path` to do so.

Usage

```
default_weights_path()
```

Value

No return value, called for reference.

`deseason_ts` *Deseason Seasonal Time Series*

Description

Removes seasonal signals from time series using either averaging or time series decomposition methods. Three decomposition methods are available: traditional decompostion, loess decomposition and X11 decompostion.

Usage

```
deseason_ts(
  data,
  increment = c("month", "year", "week", "day"),
  method = c("average", "decompose", "stl"),
  order = NULL
)
```

Arguments

data	The dataframe to be transformed, The first column must be a vector of dates with all other columns the individual time series.
increment	The time-step increment in either month, year, week, or day. Provides the basis for deaseasoning.
method	String of either "average", "decompose", "stl" or "x11" indicating the method of deseasoning. "average" subtracts the average representative month/week/day-of-the-year from each time point whereas "decompose", "stl" and "x11" subtracts the seasonal component estimated by time series decomposition, loess decomposition and the X11 method respectively.
order	String indicating the date format of the date columns. Options are "dmy", "ymd" or "mdy".

Value

Dataframe of deseasoned time series.

Examples

```
#Generate five random monthly time series
#of 5 years length.

spp_data <- matrix(nrow = 5*12, ncol = 5)
spp_data <- sapply(1:dim(spp_data)[2], function(x){
spp_data[,x] <- rnorm(5*12,mean=20,sd=5)})
multi_spp_data <- cbind("time" =
 seq(as.Date('2000/01/01'), as.Date('2004/12/01'), by="month"),
   as.data.frame(spp_data))

#Deseason using time series
#decomposition.

decomp_dat <- deseason_ts(data = multi_spp_data,
increment = "month",
method = "decompose",
order = "ymd")

#Deseason using loess

decomp_dat <- deseason_ts(data = multi_spp_data,
increment = "month",
method = "stl",
order = "ymd")
```

detrend_ts *Detrend Time Series*

Description

Removes directional signals from time series using loess, linear regression or gaussian detrending.

Usage

```
detrend_ts(data, method = "linear", bandwidth = NULL, span = 0.25, degree = 2)
```

Arguments

data	The dataframe to be detrended. The first column must be a vector of dates with all other columns the individual time series.
method	The method of detrending. Options include "linear" (residuals of a linear regression), loess (smoothing by local polynomial regression), gaussian (smoothing by a gaussian kernel), or first.difference.
bandwidth	If method = "gaussian", dictates the bandwidth of the gaussian kernel. If NULL, this is estimated from the data.
span	If method = "loess", controls the degree of smoothing as a proportion of points to be used (if span = 1, all points are used)
degree	If method = "loess", specifies the degree polynomials allowed. Options are normally 1 or 2.

Value

Dataframe of deseasoned time series.

Examples

```
#Generate five random monthly time series
#of 5 years length.

spp_data <- matrix(nrow = 5*12, ncol = 5)
spp_data <- sapply(1:dim(spp_data)[2], function(x){
spp_data[,x] <- rnorm(5*12,mean=20,sd=5)})
multi_spp_data <- cbind("time" =
 seq(as.Date('2000/01/01'), as.Date('2004/12/01'), by="month"),
   as.data.frame(spp_data))

detrend_dat <- detrend_ts(data = multi_spp_data,
method = "gaussian",
bandwidth = 2)
```

ewsnet_finetune *EWSNet Finetune*

Description

Communicates with EWSNet (https://ewsnet.github.io), a deep learning framework for modelling and anticipating regime shifts in dynamical systems, and finetunes the model to match the inputted training data. This overwrites the Pretrained weights bundled with `EWSmethods`. See `reset_ewsnet()` on how to reset these trained weights.

Usage

```
ewsnet_finetune(
  x,
  y,
  scaling = TRUE,
  envname,
  weights_path = default_weights_path()
)
```

Arguments

`x`	A numeric matrix to finetune EWSNet on. Each column represents a separate timeseries and each row is a timestep.
`y`	A numeric vector consisting of target labels for each training time series. Labels include: 0 (no transition), 1 (smooth transition) or 2 (critical transition).
`scaling`	Boolean. If `TRUE`, the time series will be scaled between 1 and 2 and scaled EWSNet model weights will be used. This is the recommended setting.
`envname`	A string naming the Python environment prepared by `ewsnet_init()`.
`weights_path`	A string naming the path to model weights installed by `ewsnet_reset()`.

Value

No return value, called for side effects.

Examples

```
#Activate python environment (only necessary
#on first opening of R session).

## Not run:
ewsnet_init(envname = "EWSNET_env")

## End(Not run)

#A dummy dataset of a hedgerow bird population
#monitored over 50 years that needs to be tuned.
```

```
abundance_data <- data.frame(time = seq(1:50),
 abundance = rnorm(50,mean = 20))

#Generate training data (this is random data as
#an example).

x <- matrix(nrow = 50, ncol = 10)
x <- sapply(1:dim(x)[2], function(i){
 x[,i] <- rnorm(50,mean=20,sd=10)})

#Label each time series.

y <- sample(0:2,10,replace = TRUE)

#Finetune EWSNet.

## Not run:
ewsnet_finetune(
 x = x,
 y = y,
 scaling = TRUE,
 envname = "EWSNET_env")

## End(Not run)

#Generate new EWSNet predictions.

## Not run:
pred <- ewsnet_predict(
 abundance_data$abundance,
 scaling = TRUE,
 ensemble = 15,
 envname = "EWSNET_env")

## End(Not run)
```

`ewsnet_init` *EWSNet Initialisation*

Description

Prepares your R session for communicating with Python. This function first searches your computer for an appropriate Python environment and activates it, importing the vital Python packages required. If no appropriate Python install or environment is found, after asking permission, miniconda is downloaded and an environment created.

Usage

```
ewsnet_init(
  envname,
  conda_path = reticulate::miniconda_path(),
  pip_ignore_installed = FALSE,
  auto = FALSE
)
```

Arguments

envname	A string naming the desired Python environment to create/activate. If no Python or environment found, the functions prompts to install miniconda and the required python packages.
conda_path	The location to install Python. By default, this follows minicondata_path defined by the reticulate package.
pip_ignore_installed	Boolean. If FALSE, any packages already installed are loaded and not re-downloaded. However, if TRUE, packages are downloaded irregardless, overwriting any version already present (is analagous to updating if required).
auto	Boolean. If FALSE, asks permission to install Python and/or packages. If TRUE, no user confirmation is required for activation.

Value

Does not return an object as is simply preparing your R session.

Examples

```
## Not run:
ewsnet_init(envname = "EWSNET_env", auto = FALSE)

## End(Not run)
#Common errors at this stage result from 'reticulate's
#behaviour. For example, conflicts between 'ewsnet_init'
#and RETICULATE_PYTHON may occur if run inside a
#RStudio R project. To fix this, navigate to
#Preferences -> Python, untick 'Automatically
#activate project-local Python environments'
#and restart R.

## Not run:
reticulate::py_config()

## End(Not run)
#If successful, 'EWSNET_env forced by use_python
#function' will be printed.
```

ewsnet_predict *EWSNet Predict*

Description

Communicates with EWSNet (https://ewsnet.github.io), a deep learning framework for modelling and anticipating regime shifts in dynamical systems, and returns the model's prediction for the inputted univariate time series.

Usage

```
ewsnet_predict(
  x,
  scaling = TRUE,
  ensemble = 25,
  envname,
  weights_path = default_weights_path()
)
```

Arguments

x	A numeric vector of values to be tested.
scaling	Boolean. If TRUE, the time series will be scaled between 1 and 2 and scaled EWSNet model weights will be used. This is the recommended setting.
ensemble	A numeric value stating the number of models to average over. Options range from 1 to 25.
envname	A string naming the Python environment prepared by ewsnet_init().
weights_path	A string naming the path to model weights installed by ewsnet_reset().

Value

A dataframe of EWSNet predictions. Values represent the estimated probability that the quoted event will occur.

Examples

```
#A dummy dataset of a hedgerow bird population
#monitored over 50 years.

abundance_data <- data.frame(time = seq(1:50),
 abundance = rnorm(50,mean = 20))

#Activate python environment (only necessary
#on first opening of R session).

## Not run:
ewsnet_init(envname = "EWSNET_env")
```

```
## End(Not run)

#Generate EWSNet predictions.

## Not run:
pred <- ewsnet_predict(
 abundance_data$abundance,
 scaling = TRUE,
 ensemble = 15,
 envname = "EWSNET_env")

## End(Not run)
```

`ewsnet_reset` *Reset EWSNet Model Weights*

Description

Restores EWSNet model weights to the pretrained defaults published at https://ewsnet.github.io. This is vital on first use of EWSNet as no model weights are provided with 'EWSmethods'. The use of this function may also be necessary after finetuning to reset the model.

Usage

```
ewsnet_reset(
  weights_path = default_weights_path(),
  remove_weights = FALSE,
  auto = FALSE
)
```

Arguments

`weights_path`	A string naming the path to install model weights. Can be changed, but by default, attempts to add weights to the same location as the Python scripts bundled with EWSmethods.
`remove_weights`	Boolean. Should all weights be removed (`TRUE`) or should weights be re/downloaded (`FALSE`).
`auto`	Boolean. If `FALSE`, asks permission to download model weights from Google Drive. If `TRUE`, no user confirmation is required for re/download.

Value

No return value, called for side effects.

Examples

```
# on first use of EWSNet via `EWSmethods`
ewsnet_reset(remove_weights = FALSE, auto = TRUE,
weights_path = tempdir())



# to remove all downloaded weights
ewsnet_reset(remove_weights = TRUE, auto = TRUE,
weights_path = tempdir())
```

FI *Calculate Fisher Information*

Description

Uses a multivariate array of time series to estimate Fisher information following the approach of Karunanithi et al. (2010).

Usage

```
FI(data, sost, winsize = 50, winspace = 1, TL = 90)
```

Arguments

`data`	A numeric matrix of individual time series across the columns. These could be different species, populations or measurements. The first column must be an equally spaced time vector.
`sost`	A 1 x n matrix where n is a length equal to the number of time series in `data`. Each value is the 'size of state' tolerable for that time series and typically is represented by the standard deviation of the time series during a reference period.
`winsize`	Numeric value. Defines the window size of the rolling window as a percentage of the time series length.
`winspace`	Numeric value. The number of data points to roll the window over in each iteration. Must be less than `winsize`.
`TL`	Numeric value. The 'tightening level' or percentage of points shared between states that allows the algorithm to classify data points as the same state.

Value

A list containing three objects:

`FI`	A dataframe of Fisher information estimates and the last time point contributing to each window.

midt_win	A numeric vector of the time index at the centre of the window for that associated value in FI.
t_win	A n x m numeric matrix where the length of n is the winspace and length of m is the number of window shifts made. Values are consequently the timepoint indices that contribute to that window.

Examples

```
#Load the multivariate simulated
#dataset `simTransComms`

data(simTransComms)

#Estimate the size-of-states for each
#time series in the first community.
#This is typically suggested
#to be the standard deviation of a
#reference period or the entire time
#series

eg.sost <- t(apply(simTransComms$community1[,3:7], MARGIN = 2, FUN = sd))
 #transpose required to ensure a 1 x n matrix

egFI <- FI(data = simTransComms$community1[1:50,2:7],
sost =  eg.sost,
winsize = 10,
winspace = 1,
TL = 90)
```

---

multiAR *Multivariate Jacobian Index Estimated From Multivariate Autocorrelation Matrix*

Description

Estimate the dominant Jacobian eigenvalue of a multivariate time series using autocorrelated stochastic differential equations

Usage

```
multiAR(data, scale = TRUE, winsize = 50, dt = 1)
```

Arguments

data	Numeric matrix with time in first column and species abundance in the second
scale	Boolean. Should data be scaled prior to estimating the Jacobian.
winsize	Numeric. Defines the window size of the rolling window as a percentage of the time series length.
dt	Numeric An appropriate time step

Value

A dataframe where the first column is last time index of the window and the second column is the estimated index value. A value <1.0 indicates stability, a value >1.0 indicates instability.

Source

Williamson and Lenton (2015). Detection of bifurcations in noisy coupled systems from multiple time series. Chaos, 25, 036407

Examples

```
#Load the multivariate simulated
#dataset `simTransComms`

data(simTransComms)

#Subset the second community prior to the transition

pre_simTransComms <- subset(simTransComms$community2,time < inflection_pt)

#Estimate the univariate stability index for the first species in
#the second community

egarJ <- multiAR(data = pre_simTransComms[,2:7],
winsize = 25, dt = 1)
```

`multiEWS` *Multivariate Early Warning Signal Assessment*

Description

A single function for performing early warning signal (EWS) assessment on multivariate systems where multiple time series have been measured. Both methods of EWS assessment can be performed (rolling or expanding windows) with the assessments returned as a dataframe. The two methods of dimension reduction used to perform these assessments are Principal Component Analysis and Maximum/Minimum Autocorrelation Factors.

Usage

```
multiEWS(
  data,
  metrics = c("meanAR", "maxAR", "meanSD", "maxSD", "eigenMAF", "mafAR", "mafSD",
    "pcaAR", "pcaSD", "eigenCOV", "maxCOV", "mutINFO"),
  method = c("expanding", "rolling"),
  winsize = 50,
  burn_in = 5,
  threshold = 2,
```

```
  tail.direction = "one.tailed"
)
```

Arguments

data	A dataframe where the first column is an equally spaced time vector and all other columns are individual time series. These could be different species, populations or measurements.
metrics	String vector of early warning signal metrics to be assessed. Options include: "meanSD", "maxSD", "meanAR", "maxAR", "eigenMAF", "mafAR", "mafSD", "pcaAR", "pcaSD", "eigenCOV", "maxCOV" and "mutINFO".
method	Single string of either "expanding" or "rolling". "expanding" calls composite, expanding window EWS assessment. "rolling" calls typical, rolling window EWS assessment.
winsize	Numeric value. If method = "rolling", defines the window size of the rolling window as a percentage of the time series' length.
burn_in	Numeric value. If method = "expanding", defines the number of data points to 'train' signals prior to EWS assessment.
threshold	Numeric value of either 1 or 2. Threshold*sigma is the value which, if the EWS strength exceeds it, constitutes a "signal".
tail.direction	String of either "one.tailed" or "two.tailed". "one.tailed" only indicates a warning if positive threshold sigma exceeded. "two.tailed" indicates a warning if positive OR negative threshold*sigma exceeded.

Value

A list containing up to two objects: EWS outputs through time (EWS), and an identifier string (method).

EWS$raw	Dataframe of EWS measurements through time. If method = "expanding", then each metric has been rbound into a single dataframe and extra columns are provided indicating whether the threshold*sigma value has been exceeded (i.e. "threshold.crossed"). If method = "rolling", then each metric's evolution over time is returned in individual columns.
EWS$dimred.ts	Dataframe containing the dimension reduction time series
EWS$cor	Dataframe of Kendall Tau correlations. Only returned if method = "rolling".

Examples

```
#Generate a random five species, non-transitioning
#ecosystem with 50 years of monitoring data.

spp_data <- matrix(nrow = 50, ncol = 5)
spp_data <- sapply(1:dim(spp_data)[2], function(x){
 spp_data[,x] <- rnorm(50,mean=20,sd=5)})
 multi_spp_data <- as.data.frame(cbind("time" =
 seq(1:50), spp_data))
```

```
#Rolling window early warning signal assessment of
#the ecosystem.

roll_ews <- multiEWS(
 data = multi_spp_data,
 method = "rolling",
 winsize = 50)

#Expanding window early warning signal assessment of
#the ecosystem.

exp_ews <- multiEWS(
 data = multi_spp_data,
 method = "expanding",
 burn_in = 10)
```

`multiJI` *S-map Jacobian index function*

Description

Calculate a stability metric from the s-map estimated Jacobian

Usage

```
multiJI(data, winsize = 50, theta_seq = NULL, scale = TRUE)
```

Arguments

`data`	Numeric matrix with time in first column and species abundances in other columns
`winsize`	Numeric. Defines the window size of the rolling window as a percentage of the time series length.
`theta_seq`	Numeric vector of thetas (nonlinear tuning parameters) to estimate the Jacobian over. If 'NULL', a default sequence covering '0:8' is provided.
`scale`	Boolean. Should data be scaled within each window prior to estimating the Jacobian.

Value

A dataframe where the first column is last time index of the window and the second column is the estimated index value. A value <1.0 indicates stability, a value >1.0 indicates instability.

Source

Ushio, M., Hsieh, Ch., Masuda, R. et al. (2018) Fluctuating interaction network and time-varying stability of a natural fish community. Nature 554, 360–363.

Examples

```
#Load the multivariate simulated
#dataset `simTransComms`

data(simTransComms)

#Subset the third community prior to the transition

pre_simTransComms <- subset(simTransComms$community3,time < inflection_pt)

#Estimate the stability index for the third community
#(trimmed for speed)

egJI <- multiJI(data = pre_simTransComms[1:10,2:5],
winsize = 75)
```

`multi_smap_jacobian` *S-map Inferred Jacobian*

Description

Performs the S-map on a multivariate time series to infer the Jacobian matrix at different points in time across thetas.

Usage

```
multi_smap_jacobian(data, theta_seq = NULL, scale = TRUE)
```

Arguments

`data`	Numeric matrix with time in first column and species abundances in other columns
`theta_seq`	Numeric vector of thetas (nonlinear tuning parameters) to estimate the Jacobian over. If 'NULL', a default sequence is provided.
`scale`	Boolean. Should data be scaled prior to estimating the Jacobian.

Value

A list containing three objects:

`smap_J`	Jacobian matrices for each point in time. It is recommended to just use the last estimate.
`rho`	Pearson correlation between observed and predicted for each species.
`smap_intercept.r`	Intercepts of the regression fit.

Source

Medeiros, L.P., Allesina, S., Dakos, V., Sugihara, G. & Saavedra, S. (2022) Ranking species based on sensitivity to perturbations under non-equilibrium community dynamics. Ecology Letters, 00, 1– 14.

Examples

```
#Load the multivariate simulated
#dataset `simTransComms`

data("simTransComms")

#Subset the third community prior to the transition

pre_simTransComms <- subset(simTransComms$community3,time < inflection_pt)


#Estimate the Jacobian using s-map (typically only
#the final estimate is informative)
est_jac <- multi_smap_jacobian(pre_simTransComms[1:10,2:7])
```

`mvi` *Multivariate Variance Index function*

Description

Calculate a multivariate variance following Brock, W. A., and S. R. Carpenter. 2006. Variance as a leading indicator of regime shift in ecosystem services. Ecology and Society 11(2): 9.

Usage

```
mvi(data, winsize = 50)
```

Arguments

`data`	A numeric matrix of species abundances, names across columns, time across rows. The first column is a time vector, the remainder are species values.
`winsize`	Numeric. Defines the window size of the rolling window as a percentage of the time series length.

Value

A matrix where the first column is last time index of the window and the second column is the estimated index value.

Source

Brock, W.A. & Carpenter, S.R. (2006) Variance as a leading indicator of regime shift in ecosystem services. Ecology and Society 11(2): 9.

Examples

```
#Load the multivariate simulated
#dataset `simTransComms`

data(simTransComms)

#Estimate the MVI for the second community

egMVI <- mvi(data = simTransComms$community2[,2:7],
winsize = 10)
```

---

perm_rollEWS *Significance Testing of Rolling Window Early Warning Signals*

Description

A function for identifying whether a warning has been generated from rolling early warning signal data using permutation tests. If a parallel connection is setup via `parallel` or `future` prior to usage of `perm_rollEWS()`, then the function is parallelised.

Usage

```
perm_rollEWS(
  data,
  metrics,
  winsize = 50,
  variate = c("uni", "multi"),
  perm.meth = "arima",
  iter = 500
)
```

Arguments

`data`	A dataframe where the first column is an equally spaced time vector and the remainder column are the time series to be assessed. If a two column dataframe is provided, and `variate = "uni"`, `uniEWS()` is called, whereas if number of columns exceeds two & `variate = "multi"`, then `multiEWS()` is called.
`metrics`	String vector of early warning signal metrics to be assessed. For `variate = "uni"` these include: `"ar1"`, `"cv"`, `"SD"`, `"acf"`, `"rr"`, `"dr"`, `"skew"` and `"kurt"`. For `variate = "multi"`, pptions include: `"meanSD"`, `"maxSD"`, `"meanAR"`, `"maxAR"`, `"eigenMAF"`, `"mafAR"`, `"mafSD"`, `"pcaAR"`, `"pcaSD"`, `"eigenCOV"`, `"maxCOV"` and `"mutINFO"`.

winsize	Numeric value. Defines the window size of the rolling window as a percentage of the time series length.
variate	String. Is a "uni"variate or "multi"variate assessment to be made.
perm.meth	String dictating the pseudo-randomisation technique to be used. Options include: "arima" (sampled from predictions of an ARIMA model), "red.noise" (red noise process using data mean, variance and autocorrelation coef) or "sample" (sampled from observed data without replacement).
iter	Numeric value. The number of permutations.

Value

A list containing up to two objects: EWS outputs through time (EWS), and an identifier string (method).

EWS$raw	Dataframe of EWS measurements through time. Each metric's evolution over time is returned in individual columns.
EWS$cor	Dataframe of Kendall Tau correlations and permuted p-values.
EWS$dimred.ts	Dataframe containing the dimension reduction time series. Only returned if variate = "multi".

Examples

```
data(simTransComms)

#Permute p value for a univariate
#time series using resampling

#(data trimmed for speed)
perm_uni <- perm_rollEWS(
data = simTransComms$community1[1:10,2:3],
 winsize = 75,
 variate = "uni",
 metrics = c("ar1", "SD", "skew"),
 perm.meth = "sample",
 iter = 25)

 #' #Permute p value for a multivariate
#community using a red.noise process


#if parallelisation desired,
#this can be achieved using the
#below code
cl <- parallel::makeCluster(2)



doParallel::registerDoParallel(cl)
```

```
perm_multi <- perm_rollEWS(
data = simTransComms$community1[1:10,2:7],
 winsize = 75,
 variate = "multi",
 metrics = c("meanAR", "maxAR", "meanSD"),
 perm.meth = "red.noise",
 iter = 25)


parallel::stopCluster(cl)
```

`plot.EWSmethods` *Plot an EWSmethods object*

Description

A function for visualising the output of `uniEWS` or `multiEWS` using ggplot2 inspired figures.

Usage

```
## S3 method for class 'EWSmethods'
plot(
  x,
  ...,
  y_lab = "Generic indicator name",
  trait_lab = "Generic trait name",
  trait_scale = 1000
)
```

Arguments

`x`	An EWSmethods object created by `uniEWS` or `multiEWS`
`...`	Additional arguments to pass to the plotting functions.
`y_lab`	String label. Labels the abundance y axis.
`trait_lab`	String label. If `trait` argument populated in `uniEWS` or `multiEWS`, & `"trait"` supplied in metrics, labels the right side y axis which represents trait values through time.
`trait_scale`	Numeric value. Scales trait y axis relative to abundance y axis.

Value

A ggplot2 object.

Examples

```
data(simTransComms)

#Subset the third community prior to the transition

pre_simTransComms <- subset(simTransComms$community3,time < inflection_pt)

#Perform multivariate EWS assessments
roll_ews <- multiEWS(
 data = pre_simTransComms[,2:7],
 method = "rolling",
 winsize = 50)

 #Plot outcome
## Not run:
 plot(roll_ews)

## End(Not run)

#Perform univariate EWS assessments on
#simulated data with traits

abundance_data <- data.frame(time = seq(1:50),
 abundance = rnorm(50,mean = 20),
 trait = rnorm(50,mean=1,sd=0.25))

trait_ews <- uniEWS(
 data = abundance_data[,1:2],
 metrics = c("ar1","SD","trait"),
 method = "expanding",
 trait = abundance_data[,3],
 burn_in = 10)

#Plot outcome
## Not run:
 plot(trait_ews, y_lab = "Abundance",
 trait_lab = "Trait value",
 trait_scale = 10)

## End(Not run)
```

---

simTransComms *Three Simulated Transitioning Communities.*

Description

A dataset containing three simulated five species communities stressed through a critical transition.

Usage

```
simTransComms
```

Format

A list of three dataframes with 301 rows and 7 variables each:

community_id the identity of the simulated community

time time index

spp_1 density of species 1

spp_2 density of species 1

spp_3 density of species 1

spp_4 density of species 1

spp_5 density of species 1

inflection_pt the time index where transition occurs

Examples

```
data("simTransComms", package = "EWSmethods")

community_data <- simTransComms$community1
```

---

uniEWS *Univariate Early Warning Signal Assessment*

Description

A function for performing early warning signal (EWS) assessment on univariate time series. Both rolling and expanding window methods of EWS assessment can be performed with the assessments returned as a dataframe.

Usage

```
uniEWS(
  data,
  metrics,
  method = c("expanding", "rolling"),
  winsize = 50,
  burn_in = 5,
  threshold = 2,
  tail.direction = "one.tailed",
  trait = NULL
)
```

Arguments

data	A dataframe where the first column is an equally spaced time vector and the second column is the time series to be assessed.
metrics	String vector of early warning signal metrics to be assessed. Options include: "ar1", "cv", "SD", "acf", "rr", "dr", "skew", "kurt", and "trait" (only available if method = "expanding").
method	Single string of either "expanding" or "rolling". "expanding" calls composite, expanding window EWS assessment. "rolling" calls typical, rolling window EWS assessment.
winsize	Numeric value. If method = "rolling", defines the window size of the rolling window as a percentage of the time series length.
burn_in	Numeric value. If method = "expanding", defines the number of data points to 'train' signals prior to EWS assessment.
threshold	Numeric value of either 1 or 2. Threshold*sigma is the value which, if the EWS strength exceeds it, constitutes a "signal".
tail.direction	String of either "one.tailed" or "two.tailed". "one.tailed" only indicates a warning if positive threshold sigma exceeded. "two.tailed" indicates a warning if positive OR negative threshold*sigma exceeded.
trait	A vector of numeric trait values if desired. Can be NULL

Value

A list containing up to two objects: EWS outputs through time (EWS), and an identifier string (method).

EWS$raw	Dataframe of EWS measurements through time. If method = "expanding", then each metric has been rbound into a single dataframe and extra columns are provided indicating whether the threshold*sigma value has been exceeded (i.e. "threshold.crossed"). If method = "rolling", then each metric's evolution over time is returned in individual columns.
EWS$cor	Dataframe of Kendall Tau correlations. Only returned if method = "rolling".

Examples

```
#A dummy dataset of a hedgerow bird population over
#25 years where both the number of individuals and
#the average bill length has been measured.

abundance_data <- data.frame(time = seq(1:25),
 abundance = rnorm(25,mean = 20),
 trait = rnorm(25,mean=1,sd=0.5))

#The early warning signal metrics to compute.

ews_metrics <- c("SD","ar1","skew")

#Rolling window early warning signal assessment of
```

```
#the bird abundance.

roll_ews <- uniEWS(
 data = abundance_data[,1:2],
 metrics =  ews_metrics,
 method = "rolling",
 winsize = 50)

#Expanding window early warning signal assessment of
#the bird abundance (with plotting).

exp_ews <- uniEWS(
 data = abundance_data[,1:2],
 metrics = ews_metrics,
 method = "expanding",
 burn_in = 10)

#Expanding window early warning signal assessment of
#the bird abundance incorporating the trait
#information.

ews_metrics_trait <- c("SD","ar1","trait")

trait_exp_ews <- uniEWS(
 data = abundance_data[,1:2],
 metrics = ews_metrics_trait,
 method = "expanding",
 burn_in = 10,
 trait = abundance_data$trait)
```

`uniJI` *Univariate S-map Jacobian index function*

Description

Calculate a stability metric from the s-map estimated Jacobian of a univariate time series

Usage

```
uniJI(data, winsize = 50, theta_seq = NULL, E = 1, tau = NULL, scale = TRUE)
```

Arguments

`data`	Numeric matrix with time in first column and species abundance in the second
`winsize`	Numeric. Defines the window size of the rolling window as a percentage of the time series length.
`theta_seq`	Numeric vector of thetas (nonlinear tuning parameters) to estimate the Jacobian over. If 'NULL', a default sequence is provided.

E	Numeric. The embedding dimension. Is suggested to be positive.
tau	Numeric. The time-delay offset to use for time delay embedding. Suggested to be positive here, but if not provided, is set to 10% the length of the time series.
scale	Boolean. Should data be scaled prior to estimating the Jacobian.

Value

A dataframe where the first column is last time index of the window and the second column is the estimated index value. A value <1.0 indicates stability, a value >1.0 indicates instability.

Source

Grziwotz, F., Chang, C.-W., Dakos, V., van Nes, E.H., Schwarzländer, M., Kamps, O., et al. (2023). Anticipating the occurrence and type of critical transitions. Science Advances, 9.

Examples

```
#Load the multivariate simulated
#dataset `simTransComms`

data(simTransComms)

#Subset the second community prior to the transition

pre_simTransComms <- subset(simTransComms$community2,time < inflection_pt)

#Estimate the univariate stability index for the first species in
#the second community

egJI <- uniJI(data = pre_simTransComms[1:25,2:3],
winsize = 75, E = 3)
```

---

uni_smap_jacobian *S-map Inferred Jacobian*

Description

Performs the S-map on a univariate time series to infer the Jacobian matrix at different points in time across thetas.

Usage

```
uni_smap_jacobian(data, theta_seq = NULL, E = 1, tau = NULL, scale = TRUE)
```

Arguments

data	Numeric matrix with time in first column and species abundance in the second
theta_seq	Numeric vector of thetas (nonlinear tuning parameters) to estimate the Jacobian over. If 'NULL', a default sequence is provided.
E	Numeric. The embedding dimension. Is suggested to be positive.
tau	Numeric. The time-delay offset to use for time delay embedding. Suggested to be positive here, but if not provided, is set to 10% the length of the time series.
scale	Boolean. Should data be scaled prior to estimating the Jacobian.

Value

A list containing three objects:

smap_J	Jacobian matrices across taus. It is recommended to average across these matrices.
eigenJ	Absolute maximum eigenvalue.
reJ	Real component of dominant eigenvalue
imJ	Imaginary component of dominant eigenvalue.

Source

Grziwotz, F., Chang, C.-W., Dakos, V., van Nes, E.H., Schwarzländer, M., Kamps, O., et al. (2023). Anticipating the occurrence and type of critical transitions. Science Advances, 9.

Examples

```
#Load the multivariate simulated
#dataset `simTransComms`

data("simTransComms")

#Subset the second community prior to the transition

pre_simTransComms <- subset(simTransComms$community2,time < inflection_pt)
winsize <- round(dim(pre_simTransComms)[1] * 50/100)

#Estimate the Jacobian for the first 50 timepoints of the
#second species using s-map
est_jac <- uni_smap_jacobian(pre_simTransComms[1:50,2:3])
```

Index