

Package ‘ecotraj’

July 22, 2025

Type Package

Title Ecological Trajectory Analysis

Version 1.1.0

Date 2025-05-05

Description Analysis of temporal changes (i.e. dynamics) of ecological entities, defined as trajectories on a chosen multivariate space, by providing a set of trajectory metrics and visual representations [De Cáceres et al. (2019) <[doi:10.1002/ecm.1350](https://doi.org/10.1002/ecm.1350)>; and Sturbois et al. (2021) <[doi:10.1016/j.ecolmodel.2020.109400](https://doi.org/10.1016/j.ecolmodel.2020.109400)>]. Includes functions to estimate metrics for individual trajectories (length, directionality, angles, ...) as well as metrics to relate pairs of trajectories (dissimilarity and convergence). Functions are also provided to estimate the ecological quality of ecosystem with respect to reference conditions [Sturbois et al. (2023) <[doi:10.1002/ecs2.4726](https://doi.org/10.1002/ecs2.4726)>].

Depends R (>= 3.5.0), Rcpp (>= 0.12.12)

Imports Kendall, MASS

LinkingTo Rcpp

License GPL (>= 2)

URL <https://emf-creaf.github.io/ecotraj/>

LazyLoad yes

Encoding UTF-8

NeedsCompilation yes

RoxygenNote 7.3.2

Suggests ape, vegclust, knitr, rmarkdown, RColorBrewer, smacof, vegan, ggplot2, hrbrthemes, reshape2, scales, tidyr, viridis, testthat (>= 3.0.0)

LazyData true

BugReports <https://github.com/emf-creaf/ecotraj/issues>

Config/testthat/edition 3

Author Miquel De Cáceres [aut, cre] (ORCID:

<<https://orcid.org/0000-0001-7132-2080>>),

Nicolas Djeghri [aut] (ORCID: <<https://orcid.org/0000-0001-5740-3386>>),

Anthony Sturbois [aut] (ORCID: <<https://orcid.org/0000-0002-9219-4468>>),
Javier De la Casa [ctb]

Maintainer Miquel De Cáceres <miquelcaceres@gmail.com>

Repository CRAN

Date/Publication 2025-05-05 13:30:02 UTC

Contents

avoca	2
defineTrajectories	3
dynamicVariation	4
furseals	6
glenan	7
glomel	8
heatmapdata	8
is.metric	9
is.synchronous	10
isoscape	11
northseaZoo	12
pike	13
referenceEnvelopes	14
subsetTrajectories	16
trajectoryComparison	18
trajectoryCyclical	22
trajectoryCyclicalPlots	28
trajectoryMetrics	31
trajectoryPlot	36
trajectoryProjection	38
transformTrajectories	39
Index	42

avoca	<i>Avoca permanent plot dataset</i>
-------	-------------------------------------

Description

Example dataset with data from 8 permanent forest plots located on slopes of a valley in the New Zealand Alps. The study area is mountainous and centered on the Craigieburn Range (Southern Alps), South Island, New Zealand. Forests plots are almost monospecific, being the mountain beech (*Fuscospora cliffortioides*) the main dominant tree species. Previously forests consisted of largely mature stands, but some of them were affected by different disturbances during the sampling period (1972-2009) which includes 9 surveys.

Format

Three data items are included:

avoca_strat An object of class `stratifiedvegdata` (see function `stratifyvegdata` from package 'vegclust') with structural and compositional data.

avoca_sites A vector identifying sampled sites of each element in `avoca_strat`.

avoca_surveys A vector identifying surveys of each element in `avoca_strat`.

Source

New Zealand National Vegetation Survey (NVS) Databank (<https://nvs.landcareresearch.co.nz/>).

References

Allen, R. B., P. J. Bellingham, and S. K. Wiser. 1999. Immediate damage by an earthquake to a temperate montane forest. *Ecology* 80:708–714.

Harcombe, P. A., R. B. Allen, J. A. Wardle, and K. H. Platt. 1998. Spatial and temporal patterns in stand structure, biomass, growth and mortality in a monospecific *Nothofagus solandri* var. *cliffortioides* (Hook. f.) Poole forest in New Zealand. *Journal of Sustainable Forestry* 6:313–343.

Hurst, J. M., R. B. Allen, D. A. Coomes, and R. P. Duncan. 2011. Size-specific tree mortality varies with neighbourhood crowding and disturbance in a montane *Nothofagus* forest. *PLoS ONE* 6.

defineTrajectories	<i>Trajectory definition</i>
--------------------	------------------------------

Description

Defines data structures for trajectory analysis

Usage

```
defineTrajectories(d, sites, surveys = NULL, times = NULL)
```

Arguments

<code>d</code>	A symmetric matrix or an object of class dist containing the distance values between pairs of ecological states..
<code>sites</code>	A character vector indicating the ecological entity (site, individual, community) corresponding to each ecological state (other types are converted to character).
<code>surveys</code>	An integer vector indicating the survey corresponding to each ecological state (only necessary when surveys are not in order).
<code>times</code>	A numeric vector indicating survey times (if missing, survey times are made equal to surveys).

Value

An object (list) of class `trajectories` with the following elements:

- `d`: An object of class `dist` containing relationships between ecological states
- `metadata`: A data frame describing trajectory states, with the following columns:
 - `sites`: A character vector indicating the ecological entity corresponding to each ecological state.
 - `surveys`: An integer vector indicating the survey corresponding to each ecological state.
 - `times`: A numeric vector indicating survey times.

See Also

[subsetTrajectories](#)

Examples

```
#Description of entities (sites) and surveys
entities <- c("1","1","1","2","2","2")
surveys <- c(1,2,3,1,2,3)

#Raw data table
xy<-matrix(0, nrow=6, ncol=2)
xy[2,2]<-1
xy[3,2]<-2
xy[4:6,1] <- 0.5
xy[4:6,2] <- xy[1:3,2]
xy[6,1]<-1

d <- dist(xy)

# Defines trajectories
x <- defineTrajectories(d, entities, surveys)
x
```

dynamicVariation

Dynamic variation and variation decomposition

Description

- Function `dynamicVariation` assesses the amount of dynamic variation observed across trajectories and the relative contribution of each of them.
- Function `variationDecomposition` performs a sum of squares decomposition of total variation in three components: (1) across trajectories (entities); (2) across time points; (3) their interaction.

Usage

```
dynamicVariation(x, ...)

variationDecomposition(x)
```

Arguments

x An object of class `trajectories` (or its children subclasses `fd.trajectories` or `cycles`).

... Additional params to be passed to function [trajectoryDistances](#).

Details

Function `variationDecomposition` requires trajectories to be synchronous. The SS sum of temporal and interaction components correspond to the SS sum, across trajectories, of function [trajectoryInternalVariation](#).

Value

- Function `dynamicVariance` returns a list with three elements (dynamic sum of squares, dynamic variance and a vector of trajectory relative contributions)
- Function `variationDecomposition` returns a data frame with results (sum of squares, degrees of freedom and variance estimates) for each variance component and the total.

See Also

[defineTrajectories](#), [is.synchronous](#), [trajectoryDistances](#), [trajectoryInternalVariation](#)

Examples

```
#Description of entities and surveys
entities <- c("1","1","1","1","2","2","2","2","3","3","3","3")
surveys <- c(1,2,3,4,1,2,3,4,1,2,3,4)

#Raw data table
xy<-matrix(0, nrow=12, ncol=2)
xy[2,2]<-1
xy[3,2]<-2
xy[4,2]<-3
xy[5:6,2] <- xy[1:2,2]
xy[7,2]<-1.5
xy[8,2]<-2.0
xy[5:6,1] <- 0.25
xy[7,1]<-0.5
xy[8,1]<-1.0
xy[9:10,1] <- xy[5:6,1]+0.25
xy[11,1] <- 1.0
xy[12,1] <-1.5
xy[9:10,2] <- xy[5:6,2]
xy[11:12,2]<-c(1.25,1.0)
```

```

d <- dist(xy)

# Defines trajectories
x <- defineTrajectories(d, entities, surveys)

# Assessment of dynamic variation and individual trajectory contributions
dynamicVariation(x)

# Variation decomposition (entity, temporal and interaction) for synchronous
# trajectories:
variationDecomposition(x)

# check the correspondence with internal variation
sum(variationDecomposition(x)[c("time", "interaction"), "ss"])
sum(trajecoryInternalVariation(x)$internal_ss)

```

furseals

furseals dataset

Description

This is a subset of a data sets from Kernaléguen et al. (2015).

Format

furseals is an object of class data.frame composed of 1414 observations and 8 variables.

ID_SITA Fur seal ID used by Sturbois et al. (under review), from 1 to 47

ID Fur seal ID used by Kernaléguen et al. (2015) in the initial data set.

Species Fur seal species: the Antarctic fur seal *Arctocephalus gazella* or the subantarctic fur seal *A. tropicalis*.

Sexe Fur seal gender, either 'Male' or 'Female'.

Time Number of the whisker sections from 1 to 30.

Place Breeding place: Crozet, Amsterdam or Kerguelen

d13C delta 13C value

d15N delta 15N value

Details

Briefly, fur seals the Antarctic fur seal *Arctocephalus gazella* and subantarctic fur seal *A. tropicalis* whisker SI values yield unique long-term information on individual behaviour which integrates the spatial, trophic and temporal dimensions of the ecological niche. The foraging strategies of this two species of sympatric fur seals were examined in the winter 2001/2002 at Crozet, Amsterdam and Kerguelen Islands (Southern Ocean) using the stable isotope values of serially sampled whiskers. The subset of the initial data set is composed of consecutive whisker sections (3 mm-long) starting from the proximal (facial) end, with the most recently synthesized tissue remaining under the skin. Only individuals ($n = 47$) with whiskers totalizing at least 30 sections were selected in the initial data, and only those 30 sections were selected.

Author(s)

Kernaléguen, L., Arnould, J.P.Y., Guinet, C., Cherel, Y.

References

Kernaléguen, L., Arnould, J.P.Y., Guinet, C., Cherel, Y., 2015. Determinants of individual foraging specialization in large marine vertebrates, the Antarctic and subantarctic fur seals. *Journal of Animal Ecology* 1081–1091.

glenan	<i>Glenan dataset</i>
--------	-----------------------

Description

Maerl bed data set to illustrate Ecological Quality Assessment (EQA)

Format

Glenan is an object of class `data.frame` composed of 32 observations and 252 variables.

Abundance.x Abundance (number of individuals) of each taxon x

Surveys Indicates different Maerl bed surveys.

Treatment Combinations of fishing dredges and pressure levels. 'CTRL' stands for control. Fishing dredges are:

- (1) a clam dredge (CD), 70 to 90 kg, 1.5 m wide, 40 teeth of 11 cm each;
- (2) a queen scallop dredge (QSD), 120 kg, 1.8 m wide, with a blade;
- (3) a king scallop dredge (KSD), 190 kg, 1.8 m wide, 18 teeth of 10 cm each every 9 cm.

Details

Experimental data set built by Tauran et al. (2020) to study the impact of fishing dredges and varying fishing pressures on maerl beds, in the bay of Brest (Brittany, France).

References

Tauran, A., Dubreuil, J., Guyonnet, B., Grall, J., 2020. Impact of fishing gears and fishing intensities on maerl beds: An experimental approach. *Journal of Experimental Marine Biology and Ecology* 533, 151472. <https://doi.org/10.1016/j.jembe.2020.151472>

See Also

[referenceEnvelopes](#)

glomel	<i>Glomel vegetation dataset</i>
--------	----------------------------------

Description

Vegetation data set to illustrate Ecological Quality Assessment (EQA)

Format

Glomel is an object of class `data.frame` composed of 23 observations and 46 variables.

ID Station ID.

Ref Logical flag to indicate stations used to define the reference envelope.

Complementary Comments regarding the quality of the ecosystem.

... Percent cover values (derived from Braun-Blanquet ordinal scale) for 43 species of vascular plants.

Details

The nature reserve of Landes et Marais de Glomel (Brittany, France) is composed of temperate Atlantic wet heaths whose reference state is commonly considered dominated by plant communities associated to acid, nutrient poor soils that are at least seasonally water logged and dominated by *Erica tetralix* and *E. ciliaris*. The data set consists of 23 rows and 46 columns. The first five stations (rows) were used to define the reference envelope, and the next 18 stations (rows) where those for which the conservation status was to be assessed.

Author(s)

Aline Bifulchi, Réserve Naturelle des landes et marais de Glomel

See Also

[referenceEnvelopes](#)

heatmapdata	<i>heatmapdata dataset</i>
-------------	----------------------------

Description

Espinasse et al. (2020) tested the application of isoscapes modelled from satellite data to the description of secondary production in the Northeast pacific. The output model fits in a $0.25^\circ \times 0.25^\circ$ spatial grid covering the region spanning from 46 to 62°N and from 195 to 235°E and supporting delta 13C and delta 15N isoscapes from 1998 to 2017.

Format

heatmapdata is an object of class dataframe composed of 9206 observations of 9 variables.

Latitude Latitude coordinate of the station, in degrees

Longitude Longitude coordinate of the station, in degrees

d13C delta 13C modelled value

d15N delta 15N modelled value

station Station ID

Years Period corresponding to the calculation of trajectory metrics

Angles Angle alpha (i.e direction) in the stable isotope space

Lengths Net change values (i.e direction) in the stable isotope space

Angles2 Angle alpha values (i.e direction) in the stable isotope space transformed for a potential use with function geom_spoke

Details

This data sets is composed of trajectory metrics calculated by Sturbois et al. (2021) for all stations within all inter-annual consecutive periods between 1998 and 2017 calculated from the whole data set of Espinasse et al. (2020) for a 1° x 1° spatial grid.

Author(s)

Espinasse, B., Hunt, B.P.V., Batten, S.D., Pakhomov, E.A.

References

Espinasse, B., Hunt, B.P.V., Batten, S.D., Pakhomov, E.A., 2020. Defining isoscapes in the North-east Pacific as an index of ocean productivity. *Global Ecol Biogeogr* 29, 246–261.

See Also

isoscape

is.metric

Metricity

Description

Checks whether the input dissimilarity matrix is metric (i.e. all triplets fulfill the triangle inequality).

Usage

```
is.metric(x, tol = 1e-04)
```

Arguments

- x Either an object of class trajectories, a symmetric [matrix](#) or an object of class [dist](#) containing the distance values between pairs of ecological states.
- tol Tolerance value for metricity

Value

A boolean indicating metric property

Author(s)

Miquel De Cáceres, CREAF

is.synchronous	<i>Synchronicity in trajectory observations</i>
----------------	---

Description

Checks whether trajectories are synchronous, meaning that observation times are equal

Usage

```
is.synchronous(x)
```

Arguments

- x An object of class trajectories (or its children subclasses `fd.trajectories` or `cycles`)

Value

A boolean indicating whether trajectories are synchronous

See Also

[defineTrajectories](#)

Examples

```
#Description of sites and surveys
sites <- c("1","1","1","2","2","2")
surveys <- c(1,2,3,1,2,3)

#Raw data table
xy<-matrix(0, nrow=6, ncol=2)
xy[2,2]<-1
xy[3,2]<-2
xy[4:6,1] <- 0.5
```

```

xy[4:6,2] <- xy[1:3,2]
xy[6,1]<-1

#Synchronous trajectories
x1 <- defineTrajectories(dist(xy), sites, surveys)
is.synchronous(x1)

# Non synchronous trajectories
x2 <- defineTrajectories(dist(xy[1:5,]), sites[1:5], surveys[1:5])
is.synchronous(x2)

```

isoscape

isoscape dataset

Description

This data sets is a subset from Espinasse et al. (2020).

Format

isoscape is an object of class dataframe composed of 978 observations of 6 variables.

Latitude Latitude coordinate of the station, in degrees

Longitude Longitude coordinate of the station, in degrees

d13C delta 13C modelled value

d15N delta 15N modelled value

station station ID

Year Year corresponding to modelled stable isotope values

Details

Briefly, Espinasse et al. (2020) tested the application of isoscapes modelled from satellite data to the description of secondary production in the Northeast pacific. The output model fits in a $0.25^\circ \times 0.25^\circ$ spatial grid covering the region spanning from 46 to 62°N and from 195 to 235°E and supporting delta 13C and delta 15N isoscapes from 1998 to 2017. The subset is composed of modelled delta 13C and delta 15N values of a $1^\circ \times 1^\circ$ spatial grid from the original modelled dataset for 2013 and 2015.

Author(s)

Espinasse, B., Hunt, B.P.V., Batten, S.D., Pakhomov, E.A.

References

Espinasse, B., Hunt, B.P.V., Batten, S.D., Pakhomov, E.A., 2020. Defining isoscapes in the North-east Pacific as an index of ocean productivity. *Global Ecol Biogeogr* 29, 246–261.

See Also

heatmapdata

northseaZoo

North Sea zooplankton dataset

Description

A multi-annual (1958-2021), monthly resolved dataset of zooplankton community composition in the Northern and Southern North Sea used to illustrate Cyclical Ecological Trajectory Analysis (CETA)

Format

northseaZoo is an object of class `list` composed of 3 objects:

Hellinger a `data.frame` containing Hellinger-transformed zooplankton taxa abundances.

times a vector indicating the date (in year) associated to each line in Hellinger.

sites a vector indicating the site ("NNS" = Northern North Sea, "SNS" = Southern North Sea) associated to each line in Hellinger.

Details

The data describes the zooplankton community in the North Sea sampled by the **Continuous Plankton Recorder (CPR)** survey. The CPR survey operates through towing of CPR samplers across commercial routes of merchant ships (plankton silk mesh = 270 microm, sampling depth = 5-10 m). When brought back to the laboratory, plankton is counted and identified taxonomically following standardized protocols. The raw data provided by the survey ([doi:10.17031/66f12be296d70](https://doi.org/10.17031/66f12be296d70)), was reformed into two monthly-resolved time series of the commonest zooplankton taxa in the Northern North Sea ("NNS") and the Southern North Sea ("SNS"). During data processing, a smoothing was performed by taking a rolling average (for each month, 5 values were averaged: a 3 months window + the corresponding month of the previous and next years). The abundances were finally Hellinger-transformed, making them amenable to ecological diversity study.

Author(s)

Nicolas Djeghri, Université de Bretagne Occidentale, France

Pierre Hélaouët and CPR survey staff, Marine Biological Association, United Kingdom

See Also

[trajectoryCyclical](#)

pike

*pike dataset***Description**

This data sets comes from Cucherousset et al. (2013).

Format

pike is an object of class dataframe composed of 58 observations of 10 variables.

trophic_status_initial Initial trophic status at release

ID ID used for each individual by Cucherousset et al. (2013)

Time Time of the stable isotope measurement: 1 (Release) or 2 (Departure)

Time_L Time of the stable isotope measurement as string, either 'Release' or 'Departure'

Date Date of release (common for all individuals) or recapture (variable dependind of the date of departure)

Size_mm Size (length) of juvenile pike, in mm

d13C delta 13C values

d15N delta 15N values

Residence_time Number of days between the release and the recapture

Trophic_status_final Trophic status at the end of the study

Details

Briefly, Cucherousset et al. (2013) released 192 individually tagged, hatchery-raised, juvenile pike (*Esox lucius* L.) with variable initial trophic position (fin delta 13C/delta 15N values). Based on delta values, individuals were classified into zooplanktivorous (delta 15N < 10 ‰) and piscivorous (delta 15N > 10 ‰) as cannibalism is commonly observed in this species. Individuals were released in a temporarily flooded grassland where pike eggs usually hatch of the Brière marsh (France) to identify the determinants of juvenile natal departure. The release site was connected through a unique point to an adjacent pond used as a nursery habitat. Fish were continuously recaptured when migrating from flooded grassland to adjacent pond. Recaptured individuals (n = 29) were anaesthetized, checked for tags, measured for fork length, fin-clipped to quantify changes in delta 13C and delta 15N values, and released.

Author(s)

Cucherousset, J., Paillisson, J.-M., Roussel, J.-M.

References

Cucherousset, J., Paillisson, J.-M., Roussel, J.-M., 2013. Natal departure timing from spatially varying environments is dependent of individual ontogenetic status. *Naturwissenschaften* 100, 761–768.

Description

Functions to assess the variability of ecological reference envelopes and to assess the ecological quality of target stations/observations with respect to reference envelopes (Sturbois et al., under review).

Usage

```
trajectoryEnvelopeVariability(  
  d,  
  sites,  
  surveys = NULL,  
  envelope = NULL,  
  nboot.ci = NULL,  
  alpha.ci = 0.05,  
  ...  
)  
  
stateEnvelopeVariability(d, envelope = NULL, nboot.ci = NULL, alpha.ci = 0.05)  
  
compareToTrajectoryEnvelope(  
  d,  
  sites,  
  envelope,  
  surveys = NULL,  
  m = 1.5,  
  comparison_target = "trajectories",  
  distances_to_envelope = FALSE,  
  distance_percentiles = FALSE,  
  ...  
)  
  
compareToStateEnvelope(  
  d,  
  envelope,  
  m = 1.5,  
  nboot.ci = NULL,  
  alpha.ci = 0.05,  
  distances_to_envelope = FALSE,  
  distance_percentiles = FALSE,  
  ...  
)
```

Arguments

<code>d</code>	A symmetric matrix or an object of class dist containing the distance values between pairs of ecological states (see details).
<code>sites</code>	A vector indicating the site corresponding to each ecological state.
<code>surveys</code>	A vector indicating the survey corresponding to each ecological state (only necessary when surveys are not in order).
<code>envelope</code>	A vector indicating the set of sites that conform the reference envelope (other sites will be compared to the envelope)
<code>nboot.ci</code>	Number of bootstrap samples for confidence intervals. If <code>nboot.ci = NULL</code> then confidence intervals are not estimated.
<code>alpha.ci</code>	Error in confidence intervals.
<code>...</code>	Additional parameters for function trajectoryDistances
<code>m</code>	Fuzziness exponent for quality value assessment
<code>comparison_target</code>	String indicating the component to be compared to the reference envelope. Either 'trajectories' (to compare complete trajectories) or 'states' (to compare individual trajectory states).
<code>distances_to_envelope</code>	Flag to indicate that distances to envelope should be included in the result
<code>distance_percentiles</code>	Flag to include the percentage of distances to the envelope (among sites corresponding to the reference) that are smaller than that of the site.

Details

Functions `stateEnvelopeVariability` and `trajectoryEnvelopeVariability` are used to assess the variability of reference envelopes. Functions `compareToStateEnvelope` and `compareToTrajectoryEnvelope` are used to evaluate the ecological quality of stations/observations with respect to a predefined reference envelope.

Value

- Functions `stateEnvelopeVariability` and `trajectoryEnvelopeVariability` are used to assess the variability of reference envelopes.
- Functions `compareToStateEnvelope` and `compareToTrajectoryEnvelope` return data frame with columns identifying the envelope and the Q statistic for the ecological quality with respect to the envelope. If `nboot.ci != NULL` extra columns are added to indicate the boundaries of a confidence interval for Q, built using bootstrap samples of the reference envelope.

Author(s)

Miquel De Cáceres, CREAF

Anthony Sturbois, Vivarmor nature, Réserve Naturelle nationale de la Baie de Saint-Brieuc

References

Sturbois, A., De Cáceres, M., Bifulchi, A., Bioret, F., Boyé, A., Gauthier, O., Grall, J., Grémare, A., Labrune, C., Robert, A., Schaal, G., Desroy, N. (2023). Ecological Quality Assessment: a general multivariate framework to report the quality of ecosystems and their dynamics with respect to reference conditions. *Ecosphere*.

See Also

[trajectoryMetrics](#), [glomel](#)

Examples

```
data(glomel)

# Extract compositional data matrix
glomel_comp <- as.matrix(glomel[,!(names(glomel) %in% c("ID", "Ref", "Complementary"))])
rownames(glomel_comp) <- glomel$ID

# Calculate Bray-Curtis distance matrix
glomel_bc <- vegan::vegdist(glomel_comp, method = "bray")

# Define reference envelope (5 stations) by observation ID
glomel_env <- glomel$ID[glomel$Ref]

# Assess quality with respect to reference envelope
compareToStateEnvelope(glomel_bc, glomel_env)
```

subsetTrajectories	<i>Trajectory subsetting</i>
--------------------	------------------------------

Description

Subsets data structures for trajectory analysis

Usage

```
subsetTrajectories(
  x,
  site_selection = NULL,
  subtrajectory_selection = NULL,
  survey_selection = NULL
)
```


Arguments

- x** An object of class trajectories (or its children subclasses fd.trajectories or cycles)
- site_selection** A character vector indicating the subset of entity (site) trajectories to be selected (if NULL, all sites are included).
- subtrajectory_selection** A character vector indicating the subset of cycles or fixed date trajectories to be selected (only used when x is of class fd.trajectories or cycles).
- survey_selection** An integer vector indicating the subset of surveys to be included (if NULL, all surveys are included).

Details

When using function subsetTrajectories on cycles or fixed-date trajectories then the parameter `site_selection` applies to sites (hence allows selecting multiple cycles or fixed-date trajectories). Specific cycles or fixed-date trajectories can be selected using `trajectory_selection`.

Value

An object (list) of class trajectories (or its children subclasses fd.trajectories or cycles), depending on the input.

See Also

[defineTrajectories](#), [trajectoryCyclical](#)

Examples

```
#Description of entities (sites) and surveys
entities <- c("1","1","1","2","2","2")
surveys <- c(1,2,3,1,2,3)

#Raw data table
xy<-matrix(0, nrow=6, ncol=2)
xy[2,2]<-1
xy[3,2]<-2
xy[4:6,1] <- 0.5
xy[4:6,2] <- xy[1:3,2]
xy[6,1]<-1

d <- dist(xy)

# Defines trajectories
x <- defineTrajectories(d, entities, surveys)
x

# Extracts (subset) second trajectory
x_2 <- subsetTrajectories(x, "2")
x_2
```

trajectoryComparison *Trajectory comparison*

Description

Functions to compare pairs of trajectories or trajectory segments.

- Function `segmentDistances` calculates the distance between pairs of trajectory segments.
- Function `trajectoryDistances` calculates the distance between pairs of trajectories.
- Function `trajectoryConvergence` performs the Mann-Kendall trend test on (1) the distances between trajectories; (2) the distance between points of one trajectory to the other; or (3) the variance of states among trajectories.
- Function `trajectoryShifts` calculates trajectory shifts (i.e. advances and delays) between trajectories assumed to follow a similar path but with different speeds or time lags.

Usage

```
segmentDistances(x, distance.type = "directed-segment", add = TRUE)
```

```
trajectoryDistances(
  x,
  distance.type = "DSPD",
  symmetrization = "mean",
  add = TRUE
)
```

```
trajectoryConvergence(x, type = "pairwise.asymmetric", add = TRUE)
```

```
trajectoryShifts(x, add = TRUE)
```

Arguments

<code>x</code>	An object of class <code>trajectories</code> .
<code>distance.type</code>	The type of distance index to be calculated (see section Details).
<code>add</code>	Flag to indicate that constant values should be added (local transformation) to correct triplets of distance values that do not fulfill the triangle inequality.
<code>symmetrization</code>	Function used to obtain a symmetric distance, so that $DSPD(T1,T2) = DSPD(T2,T1)$ (e.g., mean, max or min). If <code>symmetrization = NULL</code> then the symmetrization is not conducted and the output dissimilarity matrix is not symmetric.
<code>type</code>	A string indicating the convergence test, either <code>"pairwise.asymmetric"</code> , <code>"pairwise.symmetric"</code> or <code>"multiple"</code> (see details).

Details

Ecological Trajectory Analysis (ETA) is a framework to analyze dynamics of ecological entities described as trajectories in a chosen space of multivariate resemblance (De Cáceres et al. 2019). ETA takes trajectories as objects to be analyzed and compared geometrically.

The input distance matrix `d` should ideally be metric. That is, all subsets of distance triplets should fulfill the triangle inequality (see utility function `is.metric`). All ETA functions that require metricity include a parameter `'add'`, which by default is `TRUE`, meaning that whenever the triangle inequality is broken the minimum constant required to fulfill it is added to the three distances. If such local (and hence, inconsistent across triplets) corrections are not desired, users should find another way to modify `d` to achieve metricity, such as PCoA, metric MDS or non-metric MDS (see vignette 'Introduction to Ecological Trajectory Analysis'). If parameter `'add'` is set to `FALSE` and problems of triangle inequality exist, ETA functions may provide missing values in some cases where they should not.

The resemblance between trajectories is done by adapting concepts and procedures used for the analysis of trajectories in space (i.e. movement data) (Besse et al. 2016).

Parameter `distance.type` is the type of distance index to be calculated which for function `segmentDistances` has the following options (Besse et al. 2016, De Cáceres et al. 2019):

- `Hausdorff`: Hausdorff distance between two segments.
- `directed-segment`: Directed segment distance (default).
- `PPA`: Perpendicular-parallel-angle distance.

In the case of function `trajectoryDistances` the following values are possible (De Cáceres et al. 2019):

- `Hausdorff`: Hausdorff distance between two trajectories.
- `SPD`: Segment Path Distance.
- `DSPD`: Directed Segment Path Distance (default).
- `TSPD`: Time-Sensitive Path Distance (experimental).

Function `trajectoryConvergence` is used to study convergence/divergence between trajectories. There are three possible tests, the first two concerning pairwise comparisons between trajectories.

1. If `type = "pairwise.asymmetric"` then all pairwise comparisons are considered and the test is asymmetric, meaning that we test for trajectory A approaching trajectory B along time. This test uses distances of orthogonal projections (i.e. rejections) of states of one trajectory onto the other.
2. If `type = "pairwise.symmetric"` then all pairwise comparisons are considered but we test whether the two trajectories become closer along surveys. This test requires the same number of surveys for all trajectories and uses the sequence of distances between states of the two trajectories corresponding to the same survey.
3. If `type = "multiple"` then the function performs a single test of convergence among all trajectories. This test needs trajectories to be synchronous. In this case, the test uses the sequence of variability between states corresponding to the same time.

In all cases, a Mann-Kendall test (see [MannKendall](#)) is used to determine if the sequence of values is monotonously increasing or decreasing.

Function `trajectoryShifts` is intended to be used to compare trajectories that are assumed to follow a similar pathway. The function evaluates shifts (advances or delays) due to different trajectory speeds or the existence of time lags between them. This is done using calls to [trajectoryProjection](#). Whenever the projection of a given target state on the reference trajectory does not exist the shift cannot be evaluated (missing values are returned).

Value

Function `trajectoryDistances` returns an object of class `dist` containing the distances between trajectories (if `symmetrization = NULL` then the object returned is of class `matrix`).

Function `segmentDistances` list with the following elements:

- `Dseg`: Distance matrix between segments.
- `Dini`: Distance matrix between initial points of segments.
- `Dfin`: Distance matrix between final points of segments.
- `Dinifin`: Distance matrix between initial points of one segment and the final point of the other.
- `Dfinini`: Distance matrix between final points of one segment and the initial point of the other.

Function `trajectoryConvergence` returns a list with two elements:

- `tau`: A single value or a matrix with the statistic (Mann-Kendall's tau) of the convergence/divergence test between trajectories. If `type = "pairwise.symmetric"` then the matrix is square and if `type = "pairwise.asymmetric"` the statistic of the test of the row trajectory approaching the column trajectory. If `type = "multiple"` tau is a single value.
- `p.value`: A single value or a matrix with the p-value of the convergence/divergence test between trajectories. If `type = "pairwise.symmetric"` then the matrix of p-values is square and if `type = "pairwise.asymmetric"` then the p-value indicates the test of the row trajectory approaching the column trajectory. If `type = "multiple"` p-value is a single value.

Function `trajectoryShifts` returns an object of class `data.frame` describing trajectory shifts (i.e. advances and delays). The columns of the `data.frame` are:

- `reference`: the site (trajectory) that is taken as reference for shift evaluation.
- `site`: the target site (trajectory) for which shifts have been computed.
- `survey`: the target trajectory survey for which shift is computed.
- `time`: the time corresponding to target trajectory survey.
- `timeRef`: the time associated to the projected ecological state onto the reference trajectory.
- `shift`: the time difference between the time of the target survey and the time of projected ecological state onto the reference trajectory. Positive values mean faster trajectories and negative values mean slower trajectories.

Author(s)

Miquel De Cáceres, CREAF

Nicolas Djeghri, UBO

References

Besse, P., Guillouet, B., Loubes, J.-M. & François, R. (2016). Review and perspective for distance based trajectory clustering. *IEEE Trans. Intell. Transp. Syst.*, 17, 3306–3317.

De Cáceres M, Coll L, Legendre P, Allen RB, Wiser SK, Fortin MJ, Condit R & Hubbell S. (2019). Trajectory analysis in community ecology. *Ecological Monographs* 89, e01350.

See Also

[trajectoryMetrics](#), [trajectoryPlot](#), [transformTrajectories](#), [trajectoryProjection](#), [MannKendall](#)

Examples

```
#Description of entities (sites) and surveys
entities <- c("1","1","1","1","2","2","2","2","3","3","3","3")
surveys <- c(1,2,3,4,1,2,3,4,1,2,3,4)

#Raw data table
xy<-matrix(0, nrow=12, ncol=2)
xy[2,2]<-1
xy[3,2]<-2
xy[4,2]<-3
xy[5:6,2] <- xy[1:2,2]
xy[7,2]<-1.5
xy[8,2]<-2.0
xy[5:6,1] <- 0.25
xy[7,1]<-0.5
xy[8,1]<-1.0
xy[9:10,1] <- xy[5:6,1]+0.25
xy[11,1] <- 1.0
xy[12,1] <-1.5
xy[9:10,2] <- xy[5:6,2]
xy[11:12,2]<-c(1.25,1.0)

#Draw trajectories
trajectoryPlot(xy, entities, surveys,
              traj.colors = c("black","red", "blue"), lwd = 2)

#Distance matrix
d <- dist(xy)
d

#Trajectory data
x <- defineTrajectories(d, entities, surveys)

#Distances between trajectory segments
segmentDistances(x, distance.type = "Hausdorff")
```

```

segmentDistances(x, distance.type = "directed-segment")

#Distances between trajectories
trajectoryDistances(x, distance.type = "Hausdorff")
trajectoryDistances(x, distance.type = "DSPD")

#Trajectory convergence/divergence
trajectoryConvergence(x)

#### Example of trajectory shifts
#Description of entities (sites) and surveys
entities2 <- c("1","1","1","1","2","2","2","2","3","3","3","3")
times2 <- c(1,2,3,4,1,2,3,4,1,2,3,4)

#Raw data table
xy2<-matrix(0, nrow=12, ncol=2)
xy2[2,2]<-1
xy2[3,2]<-2
xy2[4,2]<-3
xy2[5:8,1] <- 0.25
xy2[5:8,2] <- xy2[1:4,2] + 0.5 # States are all shifted with respect to site "1"
xy2[9:12,1] <- 0.5
xy2[9:12,2] <- xy2[1:4,2]*1.25 # 1.25 times faster than site "1"

#Draw trajectories
trajectoryPlot(xy2, entities2,
              traj.colors = c("black","red", "blue"), lwd = 2)

#Trajectory data
x2 <- defineTrajectories(dist(xy2), entities2, times = times2)

#Check that the third trajectory is faster
trajectorySpeeds(x2)

#Trajectory shifts
trajectoryShifts(x2)

```

trajectoryCyclical *Functions for Cyclical Ecological Trajectory Analysis*

Description

The Cyclical extension of Ecological Trajectory Analysis (CETA) aims at allowing ETA to describe ecological trajectories presenting cyclical dynamics such as seasonal or day/night cycles. We call such trajectories "cyclical". CETA operates by subdividing cyclical trajectories into two types of sub-trajectories of interest: cycles and fixed-date trajectories.

- Cycles are sub-trajectories joining the ecological states belonging to the same cycle.
- Fixed-date trajectories are sub-trajectories joining the ecological states of the same date in different cycles (e.g. in a multi-annual cyclical trajectory with seasonality, a fixed-date trajectory might join all the ecological states associated with the January months of the different years).

We recommend reading the vignette on CETA prior to use it. The CETA functions provided here achieve one of two goals:

1. Reformatting data to analyze either cycles or fixed-date trajectories. The reformatted data can then be fed into existing ETA functions to obtain desired metrics (although special care need to be taken with cycles, see details).
2. Providing new metrics relevant to cycles complementing other ETA functions.

Usage

```
extractCycles(  
  x,  
  cycleDuration,  
  dates = NULL,  
  startdate = NA,  
  externalBoundary = "end",  
  minEcolStates = 3  
)  
  
extractFixedDateTrajectories(  
  x,  
  cycleDuration,  
  dates = NULL,  
  fixedDate = NULL,  
  namesFixedDate = NULL,  
  minEcolStates = 2  
)  
  
cycleConvexity(  
  x,  
  cycleDuration,  
  dates = NULL,  
  startdate = NA,  
  externalBoundary = "end",  
  minEcolStates = 3,  
  add = TRUE  
)  
  
cycleShifts(  
  x,  
  cycleDuration,  
  dates = NULL,  
  datesCS = NULL,  
  centering = TRUE,  
  minEcolStates = 3,  
  add = TRUE  
)  
  
cycleMetrics(  
  x,
```

```

x,
cycleDuration,
dates = NULL,
startdate = NA,
externalBoundary = "end",
minEcolStates = 3,
add = TRUE
)

```

Arguments

x	An object of class <code>trajectories</code> describing a cyclical trajectory.
cycleDuration	A value indicating the duration of a cycle. Must be in the same units as times.
dates	An optional vector indicating the dates ($< \text{cycleDuration}$) corresponding to each ecosystem state. Must be in the same units as times. Defaults to times modulo cycleDuration (see details).
startdate	An optional value indicating at which date the cycles must begin. Must be in the same units as times. Defaults to $\min(\text{dates})$.
externalBoundary	An optional string, either "end" or "start", indicating whether the start or end of the cycles must be considered "external". Defaults to "end".
minEcolStates	An optional integer indicating the minimum number of ecological states to return a fixed-date trajectory. Fixed-date trajectories comprising less ecological states than minEcolStates are discarded and do not appear in the output of the function. Defaults to 2.
fixedDate	An optional vector of dates for which fixed-date trajectories must be computed. Defaults to $\text{unique}(\text{dates})$, resulting in returning all possible fixed-date trajectories.
namesFixedDate	An optional vector of names associated to each fixedDate. Defaults to $\text{round}(\text{fixedDate}, 2)$.
add	Flag to indicate that constant values should be added (local transformation) to correct triplets of distance values that do not fulfill the triangle inequality.
datesCS	An optional vector indicating the dates for which a cyclical shift must be computed. Default to $\text{unique}(\text{dates})$ resulting in the computation of all possible cyclical shifts.
centering	An optional boolean. Should the cycles be centered before computing cyclical shifts? Defaults to TRUE.

Details

CETA functions:

- Function `extractCycles` reformats an object of class `trajectories` describing one or more cyclical trajectories into a new object of class `trajectories` designed for the analysis cycles.
- Function `extractFixedDateTrajectories` reformats an object of class `trajectories` describing one or more cyclical trajectories into a new object of class `trajectories` designed for the analysis fixed-date trajectories.

- Function `cycleConvexity` computes the "convexity" of the cycles embedded in one or more cyclical trajectories.
- Function `cycleShifts` computes the cyclical shifts (i.e. advances and delays) that can be obtain from one or more cyclical trajectories.

CETA is a little more time-explicit than the rest of ETA. Hence the parameter `times` is needed to initiate the CETA approach (classical ETA functions can work from surveys which is only ordinal). CETA also distinguishes between times and dates. Times represent linear time whereas dates represent circular time (e.g. the month of year). Dates are circular variables, coming back to zero when reaching their maximum value `cycleDuration` corresponding to the duration of a cycle. In CETA, dates are by default assumed to be times modulo `cycleDuration`. This should fit many applications but if this is not the case (i.e. if there is an offset between times and dates), dates can be specified. dates however need to remain compatible with times and `cycleDuration` (i.e. $(\text{times modulo cycleDuration}) - (\text{dates modulo cycleDuration})$ needs to be a constant).

IMPORTANT: Cycles within CETA comprises both "internal" and "external" ecological states (see the output of function `extractCycles`). This distinction is a solution to what we call the "December-to-January segment problem". Taking the example of a monthly resolved multi-annual time series, a way to make cycles would be to take the set of ecological states representing months from January to December of each year. However, this omits the segment linking December of year Y to January of year Y+1. However, including this segments means having two January months in the same cycle. The proposed solution in CETA (in the case of this specific example) is to set the January month of year Y+1 as "external". "external" ecological states need a specific handling for some operation in ETA, namely:

- Centering where external ecological states must be excluded from computation but included nonetheless in the procedure. This is handled automatically by the function `centerTrajectories`.
- Trajectory internal variability, where external ecological states must be excluded. This handled directly by the `trajectoryInternalVariation` function.
- Visualization through principal coordinate analysis of the cycles. The dedicated function `cyclePCoA` must be preferred over `trajectoryPCoA`.

As a general rule the outputs of `extractCycles` should be used as inputs in other, non-CETA function (e.g. `trajectoryDistances`). There is three important exceptions to that rule: the functions `cycleConvexity`, `cycleShifts` and `cycleMetrics`. Instead, the inputs of these three functions should parallel the inputs of `extractCycles` in a given analysis. For `cycleConvexity`, this is because convexity uses angles obtained from the whole cyclical trajectory, and not only the cycles. For `cycleShifts`, this is because cyclical shifts are not obtained with respect to a particular set of cycles. For `cycleMetrics`, this is because it calls `cycleConvexity`. The function instead compute the most adapted set of cycles to obtain the metric.

Note: Function `cycleShifts` is computation intensive for large data sets, it may not execute immediately.

Further information and detailed examples of the use of CETA functions can be found in the associated vignette.

Value

Function `extractCycles` returns the base information needed to describe cycles. Its outputs are meant to be used as input for other ETA functions. Importantly, within cycles, ecological states can

be considered "internal" or "external". Some operations and metrics within ETA use all ecological states whereas others use only "internal" ones (see details). Function `extractCycles` returns an object of class `cycles` containing:

- `d`: an object of class `dist`, the new distance matrix describing the cycles. To take in account ecological states that are both the end of a cycle and the start of another, `d` contains duplications. As compared to the input matrix, `d` may present deletions of ecological states that do not belong to any cycles (e.g. due to `minEcolStates`))
- `metadata`: an object of class `data.frame` describing the ecological states in `d` with columns:
 - `sites`: the sites associated to each ecological states.
 - `Cycles`: the names of the cycle each ecological states belongs to. The cycle name is built by combining the site name with C1, C2, C3... in chronological order.
 - `surveys`: renumbering of the surveys to describe individual Cycles.
 - `times`: the times associated to each ecological states.
 - `internal`: a boolean vector with TRUE indicating "internal" ecological states whereas FALSE indicates "external" ecological states. This has implications for how the outputs of `extractCycles` are treated by other ETA functions (see details).
 - `dates`: the dates associated to each ecological states.
- `interpolationInfo`: an output that only appear if ecological states have been interpolated. It is used internally by plotting functions (see `cyclePCoA`) but is not intended to be of interest to the end user.

Function `extractFixedDateTrajectories` returns the base information needed to describe fixed-date trajectories. Its outputs are meant to be used as inputs for other ETA functions in order to obtain desired metrics. Function `extractFixedDateTrajectories` returns an object of class `fd.trajectories` containing:

- `d`: an object of class `dist`, the new distance matrix describing the fixed-date trajectories. As compared to the input matrix, `d` may present deletions of ecological states that do not belong to any fixed-date trajectories (e.g. due to `minEcolStates`))
- `metadata`: an object of class `data.frame` describing the ecological states in `d` with columns:
 - `sites`: the sites to each ecological states.
 - `fdT`: the names of the fixed-date trajectory each ecological states belongs to. The fixed-date trajectory name is built by combining the site name with "fdT" and the name of the fixed date (from `namesFixedDate`).
 - `surveys`: renumbering of the surveys to describe individual fixed date trajectories.
 - `times`: the times associated to each ecological states.
 - `dates`: the dates associated to each ecological states.

Function `cycleConvexity` returns the a vector containing values between 0 and 1 describing the convexity of cycles. Importantly, outputs of `extractCycles` should not be used as inputs for `cycleConvexity` (see details).

Function `cycleShifts` returns an object of class `data.frame` describing cyclical shifts (i.e. advances and delays). Importantly, outputs of `extractCycles` should not be used as inputs for `cycleShifts` (see details). The columns of the `data.frame` are:

- `site`: the site for which each cycle shift has been computed.

- `dateCS`: the date for which a cycle shift has been computed.
- `timeCS`: the time of the ecological state for which a cycle shift has been computed (i.e. the time associated to the projected ecological state).
- `timeRef`: the time associated to the reference ecological state.
- `timeScale`: the time difference between the reference and the projected ecological state.
- `cyclicalShift`: the cyclical shift computed (an advance if positive, a delay if negative) in the same units as the times input.

Function `cycleMetrics` returns a data frame where rows are cycles and columns are different cycle metrics.

Author(s)

Nicolas Djeghri, UBO
Miquel De Cáceres, CREAF

References

Djeghri et al. (in preparation) Going round in cycles, but going somewhere: Ecological Trajectory Analysis as a tool to decipher seasonality and other cyclical dynamics.

See Also

[trajectoryCyclicalPlots](#), [trajectoryMetrics](#), [trajectoryComparison](#)

Examples

```
#First build a toy dataset with:
#The sampling times of the time series
timesToy <- 0:30

#The duration of the cycles (i.e. the periodicity of the time series)
cycleDurationToy <- 10

#The sites sampled (only one named "A")
sitesToy <- rep(c("A"),length(timesToy))

#And prepare a trend term
trend <- 0.05

#Build cyclical data (note that we apply the trend only to x):
x <- sin((timesToy*2*pi)/cycleDurationToy)+trend*timesToy
y <- cos((timesToy*2*pi)/cycleDurationToy)
matToy <- cbind(x,y)

#And express it as distances:
dToy <- dist(matToy)

#Make it an object of class trajectory:
cyclicalTrajToy <- defineTrajectories(d = dToy,
```

```

sites = sitesToy,
times = timesToy)

#At this stage, cycles and / or fixed date trajectories are not isolated.
#This done with the two CETA "extract" functions:
cyclesToy <- extractCycles(x = cyclicalTrajToy,
                           cycleDuration = cycleDurationToy)
fdTrajToy <- extractFixedDateTrajectories(x = cyclicalTrajToy,
                                          cycleDuration = cycleDurationToy)

#The output of these functions can be used as input
#for other ETA functions to get metrics of interest
#such as trajectory length:
trajectoryLengths(x = cyclesToy)
trajectoryLengths(x = fdTrajToy)

#or distances between trajectories:
trajectoryDistances(x = cyclesToy)
trajectoryDistances(x = fdTrajToy)

#In addition CETA adds two additional specific metrics.
#that require the same inputs as function extractCycles():
cycleConvexity(x = cyclicalTrajToy,
               cycleDuration = cycleDurationToy)
#The NA with the first cycle, is expected:
#Cycle convexity cannot be computed right at the boundary of the time series
cycleShifts(x = cyclicalTrajToy,
            cycleDuration = cycleDurationToy)
#Note that because our cycles are perfectly regular here, the cyclicalShift
#computed are all 0 (or close because of R's computing approximations)

#Subsetting cycles and fixed date trajectories:
subsetTrajectories(cyclesToy,
                  subtrajectory_selection = "A_C1")
subsetTrajectories(fdTrajToy,
                  subtrajectory_selection = c("A_fdT_2", "A_fdT_4"))

#General metrics describing the geometry of cycles:
cycleMetrics(x = cyclicalTrajToy,
            cycleDuration = cycleDurationToy)

```

trajectoryCyclicalPlots

Cyclical trajectory plots

Description

Plotting functions for Cyclical Ecological Trajectory Analysis:

- Function `cyclePCoA` removes unwanted points (see details) and performs principal coordinates analysis (`cmdscale`) and draws cycles in the ordination scatterplot.
- Function `fixedDateTrajectoryPCoA` performs principal coordinates analysis (`cmdscale`) and draws fixed date trajectories in the ordination scatterplot.

Usage

```
cyclePCoA(
  x,
  centered = FALSE,
  sites.colors = NULL,
  cycles.colors = NULL,
  print.names = FALSE,
  print.init.points = FALSE,
  cex.init.points = 1,
  axes = c(1, 2),
  ...
)

fixedDateTrajectoryPCoA(
  x,
  fixedDates.colors = NULL,
  sites.lty = NULL,
  print.names = FALSE,
  add.cyclicalTrajectory = TRUE,
  axes = c(1, 2),
  ...
)
```

Arguments

<code>x</code>	The full output of function <code>extractCycles</code> or <code>extractFixedDateTrajectories</code> as appropriate, an object of class <code>cycles</code> or <code>fd.trajectories</code> .
<code>centered</code>	Boolean. Have the cycles been centered? Default to <code>FALSE</code> .
<code>sites.colors</code>	The colors applied to the different sites. The cycles will be distinguished (old to recent) by increasingly lighter tones of the provided colors.
<code>cycles.colors</code>	The colors applied to the different cycles. Not compatible with <code>sites.colors</code> .
<code>print.names</code>	A boolean flag to indicate whether the names of cycles or fixed-date trajectories should be printed.
<code>print.init.points</code>	A boolean flag to indicate whether an initial point at the start of cycles should be printed (useful to spot the start of cycles in graphs containing many trajectories).
<code>cex.init.points</code>	The size of initial points.
<code>axes</code>	The pair of principal coordinates to be plotted.
<code>...</code>	Additional parameters for function <code>arrows</code> .

<code>fixedDates.colors</code>	The colors applied to the different fixed dates trajectories. Defaults to a simple RGB circular color palette.
<code>sites.lty</code>	The line type for the different sites (see par , "lty").
<code>add.cyclicalTrajectory</code>	A boolean flag to indicate whether the original cyclical trajectory should also be drawn as background.

Details

The functions `cyclePCoA` and `fixedDateTrajectoryPCoA` give adapted graphical representation of cycles and fixed-date trajectories using principal coordinate analysis (PCoA, see [cmdscale](#)). Function `cyclePCoA` handles external and potential interpolated ecological states so that they are correctly taken in account in PCoA (i.e. avoiding duplication, and reducing the influence of interpolated ecological states as much as possible). In case of centered cycles, the influence of these ecological states will grow as they will not correspond to duplications anymore. In case of centered cycles, the intended use is to set the parameter `centered` to `TRUE`.

Value

Functions `cyclePCoA` and `fixedDateTrajectoryPCoA` return the results of calling of [cmdscale](#).

Author(s)

Nicolas Djeghri, UBO
Miquel De Cáceres, CREAM

References

Djeghri et al. (in preparation) Going round in cycles, but going somewhere: Ecological Trajectory Analysis as a tool to decipher seasonality and other cyclical dynamics.

See Also

[trajectoryCyclical](#), [cmdscale](#)

Examples

```
#First build a toy dataset with:
#The sampling times of the time series
timesToy <- 0:30

#The duration of the cycles (i.e. the periodicity of the time series)
cycleDurationToy <- 10

#The sites sampled (only one named "A")
sitesToy <- rep(c("A"),length(timesToy))

#And prepare a trend term
trend <- 0.05
```

```

#Build cyclical data (note that we apply the trend only to x):
x <- sin((timesToy*2*pi)/cycleDurationToy)+trend*timesToy
y <- cos((timesToy*2*pi)/cycleDurationToy)
matToy <- cbind(x,y)

#And express it as distances:
dToy <- dist(matToy)

#Make it an object of class trajectory:
cyclicalTrajToy <- defineTrajectories(d = dToy,
                                     sites = sitesToy,
                                     times = timesToy)

#And extract the cycles and fixed date trajectories:
cyclesToy <- extractCycles(x = cyclicalTrajToy,
                          cycleDuration = cycleDurationToy)
fdTrajToy <- extractFixedDateTrajectories(x = cyclicalTrajToy,
                                         cycleDuration = cycleDurationToy)

#CETA plotting functions:
cyclePCoA(cyclesToy)
fixedDateTrajectoryPCoA(fdTrajToy)

#After centering of cycles, set parameter centered to TRUE in cyclePCoA():
cent_cyclesToy <- centerTrajectories(cyclesToy)
cyclePCoA(cent_cyclesToy, centered = TRUE)

```

trajectoryMetrics	<i>Trajectory metrics</i>
-------------------	---------------------------

Description

Set of functions to estimate metrics describing individual trajectories. Given input trajectory data, the set of functions that provide ETA metrics are:

- Function `trajectoryLengths` calculates lengths of directed segments and total path lengths of trajectories.
- Function `trajectoryLengths2D` calculates lengths of directed segments and total path lengths of trajectories from 2D coordinates given as input.
- Function `trajectorySpeeds` calculates speeds of directed segments and total path speed of trajectories.
- Function `trajectorySpeeds2D` calculates speeds of directed segments and total path speed of trajectories from 2D coordinates given as input.
- Function `trajectoryAngles` calculates the angle between consecutive pairs of directed segments or between segments of ordered triplets of points.

- Function `trajectoryAngles2D` calculates the angle between consecutive pairs of directed segments or between segments of ordered triplets of points.
- Function `trajectoryDirectionality` calculates (for each trajectory) a statistic that measures directionality of the whole trajectory.
- Function `trajectoryInternalVariation` calculates (for each trajectory) a statistic that measures the variability between the states included in the trajectory.
- Function `trajectoryMetrics` evaluates several trajectory metrics at once.
- Function `trajectoryWindowMetrics` evaluates several trajectory metrics on subtrajectories defined using moving windows.

Usage

```
trajectoryLengths(x, relativeToInitial = FALSE, all = FALSE)
```

```
trajectoryLengths2D(
  xy,
  sites,
  surveys = NULL,
  relativeToInitial = FALSE,
  all = FALSE
)
```

```
trajectorySpeeds(x)
```

```
trajectorySpeeds2D(xy, sites, surveys = NULL, times = NULL)
```

```
trajectoryAngles(
  x,
  all = FALSE,
  relativeToInitial = FALSE,
  stats = TRUE,
  add = TRUE
)
```

```
trajectoryAngles2D(
  xy,
  sites,
  surveys,
  relativeToInitial = FALSE,
  betweenSegments = TRUE
)
```

```
trajectoryDirectionality(x, add = TRUE, nperm = NA)
```

```
trajectoryInternalVariation(x, relativeContributions = FALSE)
```

```
trajectoryMetrics(x, add = TRUE)
```



```
trajectoryWindowMetrics(x, bandwidth, type = "surveys", add = TRUE)
```

Arguments

<code>x</code>	An object of class <code>trajectories</code> .
<code>relativeToInitial</code>	Flag to indicate that lengths or angles should be calculated with respect to initial survey.
<code>all</code>	A flag to indicate that angles are desired for all triangles (i.e. all pairs of segments) in the trajectory. If <code>FALSE</code> , angles are calculated for consecutive segments only.
<code>xy</code>	Matrix with 2D coordinates in a Cartesian space (typically an ordination of ecological states).
<code>sites</code>	A vector indicating the site corresponding to each ecological state.
<code>surveys</code>	A vector indicating the survey corresponding to each ecological state (only necessary when surveys are not in order).
<code>times</code>	A numeric vector indicating the time corresponding to each ecosystem state.
<code>stats</code>	A flag to indicate that circular statistics are desired (mean, standard deviation and mean resultant length, i.e. ρ)
<code>add</code>	Flag to indicate that constant values should be added (local transformation) to correct triplets of distance values that do not fulfill the triangle inequality.
<code>betweenSegments</code>	Flag to indicate that angles should be calculated between trajectory segments or with respect to X axis.
<code>nperm</code>	The number of permutations to be used in the directionality test.
<code>relativeContributions</code>	A logical flag to indicate that contributions of individual observations to temporal variability should be expressed in relative terms, i.e. as the ratio of the sum of squares of the observation divided by the overall sum of squares (otherwise, absolute sum of squares are returned).
<code>bandwidth</code>	Bandwidth of the moving windows (in units of surveys or times, depending on type)
<code>type</code>	A string, either "surveys" or "times", indicating how windows are defined.

Details

Ecological Trajectory Analysis (ETA) is a framework to analyze dynamics of ecological entities described as trajectories in a chosen space of multivariate resemblance (De Cáceres et al. 2019). ETA takes trajectories as objects to be analyzed and compared geometrically.

The input distance matrix `d` should ideally be metric. That is, all subsets of distance triplets should fulfill the triangle inequality (see utility function `is.metric`). All ETA functions that require metricity include a parameter `'add'`, which by default is `TRUE`, meaning that whenever the triangle inequality is broken the minimum constant required to fulfill it is added to the three distances. If such local (and hence, inconsistent across triplets) corrections are not desired, users should find another way modify `d` to achieve metricity, such as PCoA, metric MDS or non-metric MDS (see

vignette 'Introduction to Ecological Trajectory Analysis'). If parameter 'add' is set to FALSE and problems of triangle inequality exist, ETA functions may provide missing values in some cases where they should not.

Function `trajectoryAngles` calculates angles between consecutive segments in degrees. For each pair of segments, the angle between the two is defined on the plane that contains the two segments, and measures the change in direction (in degrees) from one segment to the other. Angles are always positive, with zero values indicating segments that are in a straight line, and values equal to 180 degrees for segments that are in opposite directions. If `all = TRUE` angles are calculated between the segments corresponding to all ordered triplets. Alternatively, if `relativeToInitial = TRUE` angles are calculated for each segment with respect to the initial survey.

Function `trajectoryAngles2D` calculates angles between consecutive segments in degrees from 2D coordinates given as input. For each pair of segments, the angle between the two is defined on the plane that contains the two segments, and measures the change in direction (in degrees) from one segment to the other. Angles are always positive (0 to 360), with zero values indicating segments that are in a straight line, and values equal to 180 degrees for segments that are in opposite directions. If `all = TRUE` angles are calculated between the segments corresponding to all ordered triplets. Alternatively, if `relativeToInitial = TRUE` angles are calculated for each segment with respect to the initial survey. If `betweenSegments = TRUE` angles are calculated between segments of trajectory, otherwise, If `betweenSegments = FALSE`, angles are calculated considering Y axis as the North (0°).

Function `trajectoryDirectionality` evaluates the directionality metric proposed in De Cáceres et al (2019). If `nperm` is supplied, then the function performs a permutational test to evaluate the significance of directionality, where the null hypothesis entails a random order of surveys within each trajectory. The p-value corresponds to the proportion of permutations with a directional value equal or larger than the observed.

Value

Functions `trajectoryLengths` and `trajectoryLengths2D` return a data frame with the length of each segment on each trajectory and the total length of all trajectories. If `relativeToInitial = TRUE` lengths are calculated between the initial survey and all the other surveys. If `all = TRUE` lengths are calculated for all segments.

Functions `trajectorySpeeds` and `trajectorySpeeds2D` return a data frame with the speed of each segment on each trajectory and the total speeds of all trajectories. Units depend on the units of distance matrix and the units of times of the input trajectory data.

Function `trajectoryAngles` returns a data frame with angle values on each trajectory. If `stats=TRUE`, then the mean, standard deviation and mean resultant length of those angles are also returned.

Function `trajectoryAngles2D` returns a data frame with angle values on each trajectory. If `betweenSegments=TRUE`, then angles are calculated between trajectory segments, alternatively, If `betweenSegments=FALSE`, angles are calculated considering Y axis as the North (0°).

Function `trajectoryDirectionality` returns a vector with directionality values (one per trajectory). If `nperm` is not missing, the function returns a data frame with a column of directional values and a column of p-values corresponding to the result of the permutational test.

Function `trajectoryInternalVariation` returns data.frame with as many rows as trajectories, and different columns: (1) the contribution of each individual state to the internal sum of squares

(in absolute or relative terms); (2) the overall sum of squares of internal variability; (3) an unbiased estimator of overall internal variance.

Function `trajectoryMetrics` returns a data frame where rows are trajectories and columns are different trajectory metrics.

Function `trajectoryWindowMetrics` returns a data frame where rows are midpoints over trajectories and columns correspond to different trajectory metrics.

Author(s)

Miquel De Cáceres, CREAF

Anthony Sturbois, Vivarmor nature, Réserve Naturelle nationale de la Baie de Saint-Brieuc

Nicolas Djeghri, UBO

References

De Cáceres M, Coll L, Legendre P, Allen RB, Wiser SK, Fortin MJ, Condit R & Hubbell S. (2019). Trajectory analysis in community ecology. *Ecological Monographs* 89, e01350.

See Also

[trajectoryComparison](#), [trajectoryPlot](#), [transformTrajectories](#), [cycleMetrics](#)

Examples

```
#Description of entities (sites) and surveys
entities <- c("1","1","1","2","2","2")
surveys <- c(1, 2, 3, 1, 2, 3)
times <- c(0, 1.5, 3, 0, 1.5, 3)

#Raw data table
xy <- matrix(0, nrow=6, ncol=2)
xy[2,2]<-1
xy[3,2]<-2
xy[4:6,1] <- 0.5
xy[4:6,2] <- xy[1:3,2]
xy[6,1]<-1

#Draw trajectories
trajectoryPlot(xy, entities, surveys,
              traj.colors = c("black","red"), lwd = 2)

#Distance matrix
d <- dist(xy)
d

#Trajectory data
x <- defineTrajectories(d, entities, surveys, times)

#Trajectory lengths
trajectoryLengths(x)
```

```

trajectoryLengths2D(xy, entities, surveys)

#Trajectory speeds
trajectorySpeeds(x)
trajectorySpeeds2D(xy, entities, surveys, times)

#Trajectory angles
trajectoryAngles(x)
trajectoryAngles2D(xy, entities, surveys, betweenSegments = TRUE)
trajectoryAngles2D(xy, entities, surveys, betweenSegments = FALSE)

#Several metrics at once
trajectoryMetrics(x)

```

trajectoryPlot	<i>Trajectory plots</i>
----------------	-------------------------

Description

Set of plotting functions for Ecological Trajectory Analysis:

Usage

```

trajectoryPCoA(
  x,
  traj.colors = NULL,
  axes = c(1, 2),
  survey.labels = FALSE,
  time.labels = FALSE,
  ...
)

trajectoryPlot(
  coords,
  sites,
  surveys = NULL,
  times = NULL,
  traj.colors = NULL,
  axes = c(1, 2),
  survey.labels = FALSE,
  time.labels = FALSE,
  ...
)

```

Arguments

x An object of class `trajectories`.

<code>traj.colors</code>	A vector of colors (one per site). If <code>selection != NULL</code> the length of the color vector should be equal to the number of sites selected.
<code>axes</code>	The pair of principal coordinates to be plotted.
<code>survey.labels</code>	A boolean flag to indicate whether surveys should be added as text next to arrow endpoints
<code>time.labels</code>	A boolean flag to indicate whether times should be added as text next to arrow endpoints
<code>...</code>	Additional parameters for function arrows .
<code>coords</code>	A data.frame or matrix where rows are ecological states and columns are coordinates in an arbitrary space
<code>sites</code>	A vector indicating the site corresponding to each ecological state.
<code>surveys</code>	A vector indicating the survey corresponding to each ecological state (only necessary when surveys are not in order).
<code>times</code>	A numeric vector indicating survey times.

Details

- Function `trajectoryPCoA` performs principal coordinates analysis ([cmdscale](#)) and draws trajectories in the ordination scatterplot.
- Function `trajectoryPlot` draws trajectories in a scatter plot corresponding to the input coordinates.

Value

Function `trajectoryPCoA` returns the result of calling [cmdscale](#).

Author(s)

Miquel De Cáceres, CREAF

Anthony Sturbois, Vivarmor nature, Réserve Naturelle nationale de la Baie de Saint-Brieuc

References

De Cáceres M, Coll L, Legendre P, Allen RB, Wiser SK, Fortin MJ, Condit R & Hubbell S. (2019). Trajectory analysis in community ecology. *Ecological Monographs* 89, e01350.

See Also

[trajectoryMetrics](#), [transformTrajectories](#), [cmdscale](#), [cyclePCoA](#)

Examples

```
#Description of sites and surveys
sites <- c("1","1","1","2","2","2")
surveys <- c(1,2,3,1,2,3)

#Raw data table
```

```

xy<-matrix(0, nrow=6, ncol=2)
xy[2,2]<-1
xy[3,2]<-2
xy[4:6,1] <- 0.5
xy[4:6,2] <- xy[1:3,2]
xy[6,1]<-1

#Define trajectory data
x <- defineTrajectories(dist(xy), sites, surveys)

#Draw trajectories using original coordinates
trajectoryPlot(xy, sites, surveys,
               traj.colors = c("black","red"), lwd = 2)

#Draw trajectories in a PCoA
trajectoryPCoA(x,
               traj.colors = c("black","red"), lwd = 2)

#Should give the same results if surveys are not in order
#(here we switch surveys for site 2)
temp <- xy[5,]
xy[5,] <- xy[6,]
xy[6,] <- temp
surveys[5] <- 3
surveys[6] <- 2

trajectoryPlot(xy, sites, surveys,
               traj.colors = c("black","red"), lwd = 2)

x <- defineTrajectories(dist(xy), sites, surveys)
trajectoryPCoA(x,
               traj.colors = c("black","red"), lwd = 2)

```

trajectoryProjection *Trajectory projection*

Description

Performs an projection of a set of target points onto a specified trajectory and returns the distance to the trajectory (i.e. rejection) and the relative position of the projection point within the trajectory.

Usage

```

trajectoryProjection(
  d,
  target,
  trajectory,
  tol = 1e-06,
  add = TRUE,
  force = TRUE
)

```

Arguments

d	A symmetric matrix or an object of class dist containing the distance values between pairs of ecological states (see details).
target	An integer vector of the ecological states to be projected.
trajectory	An integer vector of the ecological states conforming the trajectory onto which target states are to be projected.
tol	Numerical tolerance value to determine that projection of a point lies within the trajectory.
add	Flag to indicate that constant values should be added (local transformation) to correct triplets of distance values that do not fulfill the triangle inequality.
force	Flag to indicate that when projection falls out of the reference trajectory for a given, the closest point in the trajectory will be used.

Value

A data frame with the following columns:

- **distanceToTrajectory**: Distances to the trajectory, i.e. rejection. If there is no orthogonal projection the distance corresponds to the minimum distance to the trajectory.
- **segment**: Segment that includes the projected point or the closest state.
- **relativeSegmentPosition**: Relative position of the projected point within the segment, i.e. values from 0 to 1 with 0 representing the start of the segment and 1 representing its end.
- **relativeTrajectoryPosition**: Relative position of the projected point within the trajectory, i.e. values from 0 to 1 with 0 representing the start of the trajectory and 1 representing its end.

Author(s)

Miquel De Cáceres, CREAF

transformTrajectories *Transform trajectories*

Description

The following functions are provided to transform trajectories:

- Function **smoothTrajectories** performs multivariate smoothing on trajectory data using a Gaussian kernel.
- Function **centerTrajectories** shifts all trajectories to the center of the multivariate space and returns a modified distance matrix.
- Function **interpolateTrajectories** relocates trajectory ecological states to those corresponding to input times, via interpolation.

Usage

```
smoothTrajectories(
  x,
  survey_times = NULL,
  kernel_scale = 1,
  fixed_endpoints = TRUE
)

centerTrajectories(x, exclude = integer(0))

interpolateTrajectories(x, times)
```

Arguments

<code>x</code>	An object of class <code>trajectories</code> .
<code>survey_times</code>	A vector indicating the survey time for all surveys (if <code>NULL</code> , time between consecutive surveys is considered to be one)
<code>kernel_scale</code>	Scale of the Gaussian kernel, related to survey times
<code>fixed_endpoints</code>	A logical flag to force keeping the location of trajectory endpoints unmodified
<code>exclude</code>	An integer vector indicating sites that are excluded from trajectory centroid computation. Note: for objects of class <code>cycles</code> , external are excluded by default.
<code>times</code>	A numeric vector indicating new observation times for trajectories. Values should be comprised between time limits of the original trajectories.

Details

Details of calculations are given in De Cáceres et al (2019). Function `centerTrajectories` performs centering of trajectories using matrix algebra as explained in Anderson (2017).

Value

A modified object of class `trajectories`, where distance matrix has been transformed. When calling `interpolateTrajectories`, also the number of observations and metadata is likely to be affected.

Author(s)

Miquel De Cáceres, CREAF
Nicolas Djeghri, UBO

References

De Cáceres M, Coll L, Legendre P, Allen RB, Wiser SK, Fortin MJ, Condit R & Hubbell S. (2019). Trajectory analysis in community ecology. *Ecological Monographs* 89, e01350.

Anderson (2017). *Permutational Multivariate Analysis of Variance (PERMANOVA)*. Wiley Stat-sRef: Statistics Reference Online. 1-15. Article ID: stat07841.

See Also

[trajectoryPlot](#) [trajectoryMetrics](#)

Index

- * **data**
 - avoca, [2](#)
 - furseals, [6](#)
 - glenan, [7](#)
 - glomel, [8](#)
 - heatmapdata, [8](#)
 - isoscape, [11](#)
 - northseaZoo, [12](#)
 - pike, [13](#)
- arrows, [29](#), [37](#)
- avoca, [2](#)
- avoca_sites (avoca), [2](#)
- avoca_strat (avoca), [2](#)
- avoca_surveys (avoca), [2](#)
- centerTrajectories, [25](#)
- centerTrajectories
 - (transformTrajectories), [39](#)
- cmdscale, [29](#), [30](#), [37](#)
- compareToStateEnvelope
 - (referenceEnvelopes), [14](#)
- compareToTrajectoryEnvelope
 - (referenceEnvelopes), [14](#)
- cycleConvexity (trajectoryCyclical), [22](#)
- cycleMetrics, [35](#)
- cycleMetrics (trajectoryCyclical), [22](#)
- cyclePCoA, [25](#), [26](#), [37](#)
- cyclePCoA (trajectoryCyclicalPlots), [28](#)
- cycles, [29](#), [40](#)
- cycles (trajectoryCyclical), [22](#)
- cycleShifts (trajectoryCyclical), [22](#)
- data.frame, [12](#), [20](#), [26](#)
- defineTrajectories, [3](#), [5](#), [10](#), [17](#)
- dist, [3](#), [4](#), [10](#), [15](#), [20](#), [26](#), [39](#)
- dynamicVariation, [4](#)
- extractCycles, [29](#)
- extractCycles (trajectoryCyclical), [22](#)
- extractFixedDateTrajectories, [29](#)
- extractFixedDateTrajectories
 - (trajectoryCyclical), [22](#)
- fd.trajectories, [29](#)
- fd.trajectories (trajectoryCyclical), [22](#)
- fixedDateTrajectoryPCoA
 - (trajectoryCyclicalPlots), [28](#)
- furseals, [6](#)
- glenan, [7](#)
- glomel, [8](#), [16](#)
- heatmapdata, [8](#)
- interpolateTrajectories
 - (transformTrajectories), [39](#)
- is.metric, [9](#), [19](#), [33](#)
- is.synchronous, [5](#), [10](#)
- isoscape, [11](#)
- list, [12](#)
- MannKendall, [20](#), [21](#)
- matrix, [3](#), [10](#), [15](#), [39](#)
- northseaZoo, [12](#)
- par, [30](#)
- pike, [13](#)
- referenceEnvelopes, [7](#), [8](#), [14](#)
- segmentDistances
 - (trajectoryComparison), [18](#)
- smoothTrajectories
 - (transformTrajectories), [39](#)
- stateEnvelopeVariability
 - (referenceEnvelopes), [14](#)
- subsetTrajectories, [4](#), [16](#)
- trajectories, [18](#), [24](#), [33](#), [36](#), [40](#)

- trajectories (defineTrajectories), 3
- trajectoryAngles (trajectoryMetrics), 31
- trajectoryAngles2D (trajectoryMetrics), 31
- trajectoryComparison, 18, 27, 35
- trajectoryConvergence
 - (trajectoryComparison), 18
- trajectoryCyclical, 12, 17, 22, 30
- trajectoryCyclicalPlots, 27, 28
- trajectoryDirectionality
 - (trajectoryMetrics), 31
- trajectoryDistances, 5, 15
- trajectoryDistances
 - (trajectoryComparison), 18
- trajectoryEnvelopeVariability
 - (referenceEnvelopes), 14
- trajectoryInternalVariation, 5, 25
- trajectoryInternalVariation
 - (trajectoryMetrics), 31
- trajectoryLengths (trajectoryMetrics), 31
- trajectoryLengths2D
 - (trajectoryMetrics), 31
- trajectoryMetrics, 16, 21, 27, 31, 37, 41
- trajectoryPCoA, 25
- trajectoryPCoA (trajectoryPlot), 36
- trajectoryPlot, 21, 35, 36, 41
- trajectoryProjection, 20, 21, 38
- trajectoryShifts
 - (trajectoryComparison), 18
- trajectorySpeeds (trajectoryMetrics), 31
- trajectorySpeeds2D (trajectoryMetrics), 31
- trajectoryWindowMetrics
 - (trajectoryMetrics), 31
- transformTrajectories, 21, 35, 37, 39
- variationDecomposition
 - (dynamicVariation), 4