Package: ecoregime (via r-universe)

October 4, 2026

Title Analysis of Ecological Dynamic Regimes

Version 0.4.1

Description A toolbox for implementing the Ecological Dynamic Regime framework, including functions to characterize and compare groups of ecological trajectories (Sánchez-Pinillos et al., 2023 <doi:10.1002/ecm.1589>); assess the ecological resilience of a disturbed system using a reference dynamic regime (Sánchez-Pinillos et al., 2024 <doi:10.1016/j.biocon.2023.110409>); and forecast ecological trajectories from a dynamic regime (Sánchez-Pinillos et al. 2026, <doi:10.1111/2041-210x.70372>). Additional functions are also available for visualizing ecological dynamic regimes, their representative trajectories, as well as predicted trajectories in a multidimensional state space.

License GPL (>= 3)

Encoding UTF-8

Roxygen list(markdown = TRUE)

URL https://mspinillos.github.io/ecoregime/, https://github.com/MSPinillos/ecoregime

BugReports https://github.com/MSPinillos/ecoregime/issues

Depends R (>= 4.4.0)

LazyData true

Imports ape, data.table, ecotraj (>= 1.1.1), graphics, grDevices, methods, shape, smacof, stats, stringr

Suggests knitr, rmarkdown, testthat (>= 3.0.0), vegan

Config/testthat/edition 3

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

RoxygenNote 8.0.0

Config/pak/sysreqs cmake make libicu-dev libuv1-dev libx11-dev zlib1g-dev

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

Date/Publication 2026-08-04 17:49:51 UTC

RemoteUrl https://github.com/mspinillos/ecoregime

RemoteRef HEAD

RemoteSha 718cb58c544b817b55ddc57de44f62b3d8d01a76

Contents

define_retra 2
dist_edr 6
EDR_data 8
EDR_metrics 10
MPD 13
petra_edr 14
plot.PETRA 19
plot.RETRA 23
plot_edr 25
resilience_metrics 28
retra_edr 33
state_to_trajectory 36
summary.RETRA 38

Index **40**

`define_retra` *Define representative trajectories from trajectory features*

Description

Generate an object of class `RETRA` from a data frame containing trajectory states to define representative trajectories in Ecological Dynamic Regimes (EDR).

Usage

```
define_retra(data, d = NULL, trajectories = NULL, states = NULL, retra = NULL)
```

Arguments

`data` A data frame of four columns indicating identifiers for the new representative trajectories, the individual trajectories or sites to which the states belong, the order of the states in the individual trajectories, and the identifier of the representative trajectory to which the states belong (only if `!is.null(retra)`). Alternatively, 'data' can be a vector or a list of character vectors including the sequence of segments forming the new representative trajectory. See Details for further clarifications to define `data`.

d	Either a symmetric matrix or an object of class dist containing the dissimilarities between each pair of states of all trajectories in the EDR. If NULL (default), the length (Length) of the new representative trajectories and the distances between states of different trajectories or sites (Link_distance) are not calculated.
trajectories	Only needed if !is.null(d). Vector indicating the trajectory or site to which each state in d belongs.
states	Only needed if !is.null(d). Vector of integers indicating the order of the states in d for each trajectory.
retra	Object of class RETRA returned from retra_edr(). If NULL (default), minSegs and Seg_density are not provided for the new representative trajectories.

Details

Each representative trajectory returned by the function retra_edr() corresponds to the longest sequence of representative segments that can be linked according to the criteria defined in the RETRA-EDR algorithm (Sánchez-Pinillos et al., 2023). One could be interested in splitting the obtained trajectories, considering only a fraction of the returned trajectories, or defining representative trajectories following different criteria than those in RETRA-EDR. The function define_retra() allows generating an object of class RETRA that can be used in other functions of ecoregime (e.g., plot()).

For that, it is necessary to provide information about the set of segments or trajectory states that form the new representative trajectory through the argument data:

- data can be defined as a **data frame** with as many rows as the number of states in all representative trajectories and the following columns:

  RT A string indicating the identifier of the new representative trajectories. Each identifier needs to appear as many times as the number of states forming each representative trajectory.

  RT_traj A vector indicating the individual trajectories in the EDR to which each state of the new representative trajectory belongs.

  RT_states A vector of integers indicating the identifier of the states forming the new representative trajectories. Each integer must refer to the order of the states in the individual trajectories of the EDR to which they belong.

  RT_retra Only if the new trajectories are defined from representative trajectories returned by retra_edr() (when !is.null(retra)). A vector of strings indicating the representative trajectory in retra to which each state belongs.

- Alternatively, data can be defined as either a **vector** (if there is one representative trajectory) or a **list of character vectors** (with as many elements as the number of representative trajectories desired) containing the sequence of segments of the representative trajectories. In any case, each segment needs to be specified in the form traj[st1-st2], where traj is the identifier of the original trajectory to which the segment belongs and st1 and st2 are identifiers of the initial and final states defining the segment. If only one state of an individual trajectory is considered to form the representative trajectory, the corresponding segment needs to be defined as traj[st-st].

Value

An object of class RETRA, which is a list of length equal to the number of representative trajectories defined. For each trajectory, the following information is returned:

minSegs Value of the minSegs parameter used in retra_edr(). If retra is NULL, minSegs = NA.

Segments Vector of strings including the sequence of segments forming the representative trajectory. Each segment is identified by a string of the form traj[st1-st2], where traj is the identifier of the original trajectory to which the segment belongs and st1 and st2 are identifiers of the initial and final states defining the segment. The same format traj[st1-st2] is maintained when only one state of an individual trajectory is considered (st1 = st2). traj, st1, and st2 are recycled from data.

Size Integer indicating the number of states forming the representative trajectory.

Length Numeric value indicating the length of the representative trajectory, calculated as the sum of the dissimilarities in d between every pair of consecutive states. If d is NULL, Length = NA.

Link_distance Data frame of two columns indicating artificial links between two segments (Link) and the dissimilarity between the connected states (Distance). When two representative segments are linked by a common state or by two consecutive states of the same trajectory, the link distance is zero or equal to the length of a real segment, respectively. In both cases, the link is not considered in the returned data frame. If d is NULL, Link_distance = NA.

Seg_density Data frame of two columns and one row for each representative segment. Density contains the number of segments in the EDR that is represented by each segment of the representative trajectory. kdTree_depth contains the depth of the k-d tree for each leaf represented by the corresponding segment. That is, the number of partitions of the ordination space until finding a region with minSegs segments or less. If retra is NULL, Seg_density = NA.

Author(s)

Martina Sánchez-Pinillos

See Also

retra_edr() for identifying representative trajectories in EDRs through RETRA-EDR.

summary() for summarizing the characteristics of the representative trajectories.

plot() for plotting representative trajectories in an ordination space representing the state space of the EDR.

Examples

```
# Example 1 ----------------------------------------------------------------
# Define representative trajectories from the outputs of retra_edr().

# Identify representative trajectories using retra_edr()
d <- EDR_data$EDR1$state_dissim
trajectories <- EDR_data$EDR1$abundance$traj
states <- EDR_data$EDR1$abundance$state
old_retra <- retra_edr(d = d, trajectories = trajectories, states = states,
                       minSegs = 5)
```

```
# retra_edr() returns three representative trajectories
old_retra

# Keep the last five segments of trajectories "T2" and "T3"
selected_segs <- old_retra$T2$Segments[4:length(old_retra$T2$Segments)]

# Identify the individual trajectories for each state...
selected_segs
selected_traj <- rep(c(15, 4, 4, 1, 14), each = 2)

# ...and the states (in the same order as the representative trajectory).
selected_states <- c(1, 2, 2, 3, 3, 4, 1, 2, 2, 3)

# Generate the data frame with the format indicated in the documentation
df <- data.frame(RT = rep("A", length(selected_states)),
                 RT_traj = selected_traj,
                 RT_states = as.integer(selected_states),
                 RT_retra = rep("T2", length(selected_states)))

# Remove duplicates (trajectory 4, state 3)
df <- unique(df)

# Generate a RETRA object using define_retra()
new_retra <- define_retra(data = df,
                          d = d,
                          trajectories = trajectories,
                          states = states,
                          retra = old_retra)

# Example 2 ------------------------------------------------------------------
# Define representative trajectories from sequences of segments

# Select all segments in T1, split T2 into two new trajectories, and include
# a trajectory composed of states belonging to trajectories "5", "6", and "7"
data <- list(old_retra$T1$Segments,
             old_retra$T2$Segments[1:3],
             old_retra$T2$Segments[4:8],
             c("5[1-2]", "5[2-3]", "7[4-4]", "6[4-5]"))

# Generate a RETRA object using define_retra()
new_retra <- define_retra(data = data,
                          d = d,
                          trajectories = trajectories,
                          states = states,
                          retra = old_retra)

# Example 3 ------------------------------------------------------------------
# Define two representative trajectories from individual trajectories in EDR1.

# Define trajectory "A" from states in trajectories 3 and 4
data_A <- data.frame(RT = rep("A", 4),
                     RT_traj = c(3, 3, 4, 4),
                     RT_states = c(1:2, 4:5))
```

```
# Define trajectory "B" from states in trajectories 5, 6, and 7
data_B <- data.frame(RT = rep("B", 5),
                     RT_traj = c(5, 5, 7, 6, 6),
                     RT_states = c(1, 2, 4, 4, 5))

# Compile data for both trajectories in a data frame
df <- rbind(data_A, data_B)
df$RT_states <- as.integer(df$RT_states)

# Generate a RETRA object using define_retra()
new_retra <- define_retra(data = df, d = EDR_data$EDR1$state_dissim,
                          trajectories = EDR_data$EDR1$abundance$traj,
                          states = EDR_data$EDR1$abundance$state)
```

`dist_edr` *Dissimilarities between Ecological Dynamic Regimes*

Description

Generate a matrix containing dissimilarities between one or more pairs of Ecological Dynamic Regimes (EDR). `dist_edr()` computes different dissimilarity indices, all of them based on the dissimilarities between the trajectories of two EDRs.

Usage

```
dist_edr(
  d,
  d.type,
  trajectories = NULL,
  states = NULL,
  edr,
  metric = "dDR",
  symmetrize = NULL,
  ...
)
```

Arguments

`d`	Symmetric matrix or object of class `dist` containing the dissimilarities between each pair of states of all trajectories in the EDR or the dissimilarities between each pair of trajectories.
`d.type`	One of `"dStates"` (if `d` contains state dissimilarities) or `"dTraj"` (if `d` contains trajectory dissimilarities).
`trajectories`	Only if `d.type = "dStates"`. Vector indicating the trajectory or site corresponding to each entry in `d`.

`states`	Only if `d.type = "dStates"`. Vector of integers indicating the order of the states in `d` for each trajectory.
`edr`	Vector indicating the EDR to which each trajectory/state in `d` belongs.
`metric`	A string indicating the dissimilarity index to be used: `"dDR"` (default), `"minDist"`, `"maxDist"`.
`symmetrize`	String naming the function to be called to symmetrize the resulting dissimilarity matrix (`"mean"`, `"min"`, `"max"`, `"lower"`, `"upper"`). If `NULL` (default), the matrix is not symmetrized.
`...`	Only if `d.type = "dStates"`. Further arguments to calculate trajectory dissimilarities. See `ecotraj::trajectoryDistances()`.

Details

The implemented metrics are:

`"dDR"` $d_{DR}(R_1, R_2) = \frac{1}{n}\sum_{i=1}^{n} d_{TR}(T_{1i}, R_2)$

`"minDist"` $d_{DRmin}(R_1, R_2) = \min_{i=1}^{n}\{d_{TR}(T_{1i}, R_2)\}$

`"maxDist"` $d_{DRmax}(R_1, R_2) = \max_{i=1}^{n}\{d_{TR}(T_{1i}, R_2)\}$

where R_1 and R_2 are two EDRs composed of n and m ecological trajectories, respectively, and $d_{TR}(T_{1i}, R_2)$ is the dissimilarity between the trajectory T_{1i} of R_1 and the closest trajectory of R_2:

$d_{TR}(T_{1i}, R_2) = \min\{d_T(T_{1i}, T_{21}), ..., d_T(T_{1i}, T_{2m})\}$

The metrics calculated are not necessarily symmetric. That is, $d_{DR}(R_1, R_2)$ is not necessarily equal to $d_{DR}(R_2, R_1)$. It is possible to symmetrize the returned matrix by indicating the name of the function to be used in `symmetrize`:

`"mean"` $d_{DRsym} = \frac{d_{DR}(R_1,R_2)+d_{DR}(R_2,R_1)}{2}$

`"min"` $d_{DRsym} = \min\{d_{DR}(R_1, R_2), d_{DR}(R_2, R_1)\}$

`"max"` $d_{DRsym} = \max\{d_{DR}(R_1, R_2), d_{DR}(R_2, R_1)\}$

`"lower"` The lower triangular part of the dissimilarity matrix is used.

`"upper"` The upper triangular part of the dissimilarity matrix is used.

Value

Matrix including the dissimilarities between every pair of EDRs.

Author(s)

Martina Sánchez-Pinillos

References

Sánchez-Pinillos, M., Kéfi, S., De Cáceres, M., Dakos, V. 2023. Ecological Dynamic Regimes: Identification, characterization, and comparison. *Ecological Monographs*. doi:10.1002/ecm.1589

Examples

```
if (requireNamespace("vegan", quietly = TRUE)) {
    # Load species abundances and compile in a data frame
    abun1 <- EDR_data$EDR1$abundance
    abun2 <- EDR_data$EDR2$abundance
    abun3 <- EDR_data$EDR3$abundance
    abun <- data.frame(rbind(abun1, abun2, abun3))

    # Define row names in abun to keep the reference of the EDR, trajectory, and
    # state
    row.names(abun) <- paste0(abun$EDR, "_", abun$traj, "_", abun$state)

    # Calculate dissimilarities between every pair of states
    # For example, Bray-Curtis index
    dStates <- vegan::vegdist(abun[, -c(1, 2, 3)], method = "bray")

    # Use the labels in dStates to define the trajectories to which each state
    # belongs
    id_traj <- vapply(strsplit(labels(dStates), "_"), function(x){
      paste0(x[1], "_", x[2])
    }, character(1))
    id_state <- vapply(strsplit(labels(dStates), "_"), function(x){
      as.integer(x[3])
    }, integer(1))
    id_edr <- vapply(strsplit(labels(dStates), "_"), function(x){
      paste0("EDR", x[1])
    }, character(1))

    # Compute dissimilarities between EDRs without symmetrizing the matrix
    # and using state dissimilarities
    dEDR <- dist_edr(d = dStates, d.type = "dStates",
                     trajectories = id_traj,
                     states = id_state,
                     edr = id_edr,
                     metric = "dDR",
                     symmetrize = NULL)

}
```

EDR_data *Ecological Dynamic Regime data*

Description

Example datasets to characterize and compare EDRs, including abundance data, state, segment, and trajectory dissimilarity matrices for 93 artificial communities belonging to three different EDRs.

Usage

```
EDR_data
```

Format

List of four nested sublists. Each element of "EDR1", "EDR2", and "EDR3" is associated with one EDR and includes the following elements:

- abundance: Data table with 15 columns and one row for each community state:
  - EDR: Integer indicating the identifier of the EDR.
  - traj: Integer containing the identifier of the trajectory for each artificial community in the corresponding EDR. Each trajectory represents a different sampling unit.
  - state: Integer indicating the observations or states of each community. The sequence of states of a given community forms a trajectory.
  - sp1, ..., sp12: Vectors containing species abundances for each community state.
- state_dissim: Object of class dist containing Bray-Curtis dissimilarities between every pair of states in abundance.
- segment_dissim: Object of class dist containing the dissimilarities between every pair of trajectory segments in abundance.
- traj_dissim: Object of class dist containing the dissimilarities between every pair of community trajectories in abundance.

The element EDR3_disturbed represents the dynamics of three disturbed communities originally associated with EDR3. It includes an abundance matrix with 16 columns and one row for each community state. The column disturbed_states is a numeric vector indicating whether the corresponding state represents a state before the disturbance (0), during or immediately after the release of the disturbance (1), or a post-disturbance state (> 1).

Details

Artificial data was generated following the procedure explained in Box 1 in Sánchez-Pinillos et al. (2023). The initial state of each community was defined using a hypothetical environmental space with optimal locations for 12 species. Community dynamics were simulated using a general Lotka-Volterra model.

Abundances for EDR3_disturbed were generated following the procedure explained in Sánchez-Pinillos et al. (2024) for ecological systems affected by pulse disturbances.

State dissimilarities were calculated using the Bray-Curtis metric. Segment and trajectory dissimilarities were calculated using the package 'ecotraj'.

References

Sánchez-Pinillos, M., Kéfi, S., De Cáceres, M., Dakos, V. 2023. Ecological Dynamic Regimes: Identification, characterization, and comparison. *Ecological Monographs*. doi:10.1002/ecm.1589

Sánchez-Pinillos, M., Dakos, V., Kéfi, S. 2024. Ecological Dynamic Regimes: A key concept for assessing ecological resilience. *Biological Conservation*. doi:10.1016/j.biocon.2023.110409

EDR_metrics *Metrics of trajectory distribution in Ecological Dynamic Regimes*

Description

Set of metrics to analyze the distribution and variability of trajectories in Ecological Dynamic Regimes (EDR), including dynamic dispersion (dDis), dynamic beta diversity (dBD), and dynamic evenness (dEve).

Usage

```
dDis(
  d,
  d.type,
  trajectories,
  states = NULL,
  reference,
  w.type = "none",
  w.values,
  ...
)

dBD(d, d.type, trajectories, states = NULL, ...)

dEve(d, d.type, trajectories, states = NULL, w.type = "none", w.values, ...)
```

Arguments

`d`	Symmetric matrix or object of class `dist` containing the dissimilarities between each pair of states of all trajectories in the EDR or the dissimilarities between each pair of trajectories. To compute dDis, `d` needs to include the dissimilarities between all states/trajectories and the states/trajectory of reference.
`d.type`	One of `"dStates"` (if `d` contains state dissimilarities) or `"dTraj"` (if `d` contains trajectory dissimilarities).
`trajectories`	Vector indicating the trajectory or site corresponding to each entry in `d`.
`states`	Only if `d.type = "dStates"`. Vector of integers indicating the order of the states in `d` for each trajectory.
`reference`	Vector of the same class as `trajectories` and length equal to one, indicating the reference trajectory to compute dDis.
`w.type`	Method used to weight individual trajectories: • `"none"`: All trajectories are considered equally relevant (default). • `"length"`: Trajectories are weighted by their length, calculated as the sum of the dissimilarities between every pair of consecutive states. `d` must contain dissimilarities between trajectory states and `d.type = "dStates"`.

- "size": Trajectories are weighted by their size, calculated as the number of states forming the trajectory. d must contain dissimilarities between trajectory states and d.type = "dStates".
- "precomputed": Trajectories weighted according to different criteria.

w.values — Only if w.type = "precomputed". Numeric vector of length equal to the number of trajectories containing the weight of each trajectory.

... — Only if d.type = "dStates". Further arguments to calculate trajectory dissimilarities. See ecotraj::trajectoryDistances().

Details

Dynamic dispersion (dDis())

dDis is calculated as the average dissimilarity between each trajectory in an EDR and a target trajectory taken as reference (Sánchez-Pinillos et al., 2023).

$$dDis = \frac{\sum_{i=1}^{m} d_{i\alpha}}{m}$$

where $d_{i\alpha}$ is the dissimilarity between trajectory i and the trajectory of reference α, and m is the number of trajectories.

Alternatively, it is possible to calculate a weighted mean of the dissimilarities by assigning a weight to each trajectory.

$$dDis = \frac{\sum_{i=1}^{m} w_i d_{i\alpha}}{\sum_{i=1}^{m} w_i}$$

where w_i is the weight assigned to trajectory i.

Dynamic beta diversity (dBD())

dBD quantifies the overall variation of the trajectories in an EDR and is equivalent to the average distance to the centroid of the EDR (De Cáceres et al., 2019).

$$dBD = \frac{\sum_{i=1}^{m-1} \sum_{j=i+1}^{m} d_{ij}^2}{m(m-1)}$$

Dynamic evenness (dEve())

dEve quantifies the regularity with which an EDR is filled by the individual trajectories (Sánchez-Pinillos et al., 2023).

$$dEve = \frac{\sum_{l=1}^{m-1} \min(\frac{d_{ij}}{\sum_{l=1}^{m-1} d_{ij}}, \frac{1}{m-1}) - \frac{1}{m-1}}{1 - \frac{1}{1-1}}$$

where d_{ij} is the dissimilarity between trajectories i and j linked in a minimum spanning tree by the link l.

Optionally, it is possible to weight the trajectories of the EDR. In that case, *dEve* becomes analogous to the functional evenness index proposed by Villéger et al. (2008).

$$dEve_w = \frac{\sum_{l=1}^{m-1} \min(\frac{EW_{ij}}{\sum_{l=1}^{m-1} EW_{ij}}, \frac{1}{m-1}) - \frac{1}{m-1}}{1 - \frac{1}{1-1}}$$

where EW_{ij} is the weighted evenness:

$$EW_{ij} = \frac{d_{ij}}{w_i + w_j}$$

Value

- `dDis()` returns the value of dynamic dispersion for a given trajectory taken as a reference.
- `dBD()` returns the value of dynamic beta diversity.
- `dEve()` returns the value of dynamic evenness.

Author(s)

Martina Sánchez-Pinillos

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.

Sánchez-Pinillos, M., Kéfi, S., De Cáceres, M., Dakos, V. 2023. Ecological Dynamic Regimes: Identification, characterization, and comparison. *Ecological Monographs*. doi:10.1002/ecm.1589

Villéger, S., Mason, N.W.H., Mouillot, D. (2008) New multidimensional functional diversity indices for a multifaced framework in functional ecology. Ecology.

Examples

```
# Data to compute dDis, dBD, and dEve
dStates <- EDR_data$EDR1$state_dissim
dTraj <- EDR_data$EDR1$traj_dissim
trajectories <- paste0("T", EDR_data$EDR1$abundance$traj)
states <- EDR_data$EDR1$abundance$state

# Dynamic dispersion taking the first trajectory as reference
dDis(d = dTraj, d.type = "dTraj", trajectories = unique(trajectories),
         reference = "T1")

# Dynamic dispersion weighting trajectories by their length
dDis(d = dStates, d.type = "dStates", trajectories = trajectories, states = states,
         reference = "T1", w.type = "length")

# Dynamic beta diversity using trajectory dissimilarities
dBD(d = dTraj, d.type = "dTraj", trajectories = unique(trajectories))

# Dynamic evenness
dEve(d = dStates, d.type = "dStates", trajectories = trajectories, states = states)

# Dynamic evenness considering that the 10 first trajectories are three times
# more relevant than the rest
w.values <- c(rep(3, 10), rep(1, length(unique(trajectories))-10))
dEve(d = dTraj, d.type = "dTraj", trajectories = unique(trajectories),
         w.type = "precomputed", w.values = w.values)
```

MPD *Mean predicted deviation (MPD)*

Description

Metric to quantify the accuracy of predicted trajectories using PETRA-EDR.

Usage

```
MPD(x, pc_predicted = 1)
```

Arguments

x	Object of class PETRA returned by petra_edr() using return_args = T.
pc_predicted	Numeric value in the range 0-1, indicating the percentage of predicted states used to compute MPD.

Details

MDP estimates the accuracy of predicted trajectories (x) returned by petra_edr(). MDP is based on the assumption that only one trajectory passes by each target, and therefore, subsequent trajectories predicted from any point of x should contain the original states in the target.

$MPD = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} d(x_i,\hat{Z}_j)}{nm}$

where $d(x_i, \hat{Z}_i)$ is the dissimilarity between the initial or final state x_i of the target and the predicted trajectory $\hat{Z}_j$, defined as the minimum dissimilarity between x_i and the predicted states of $\hat{Z}_j$; n is the number of states of the target; and m is the number of states of the trajectory predicted from the target ($\hat{Z}_i$).

pc_predicted values lower than 1 may reduce the computation time.

Value

MPD() returns a numeric value quantifying the average dissimilarity between the target used to compute x and the trajectories predicted from the precedent and following states included in x. Note that the larger the value of MPD, the lower the accuracy of x.

Author(s)

Martina Sánchez-Pinillos

References

Sánchez-Pinillos M., Fortin, M-J., Messier, C., Kneeshaw, D. 2026. Forecasting ecological trajectories from ecological dynamic regimes to improve resilience analysis. *Methods in Ecology and Evolution*. doi:10.1111/2041-210x.70372

See Also

petra_edr() for predicting ecological trajectories from EDRs.

plot.PETRA() for plotting predicted trajectories in an ordination space representing the state space of the EDR.

Examples

```
if (requireNamespace("vegan", quietly = TRUE)) {
  # Define state_var including the state variables for the states in the EDR and
  # the target
  EDR_var <- EDR_data$EDR1$abundance
  target_var <- data.frame(sp1 = 30, sp2 = 10, sp3 = 13, sp4 = 12, sp5 = 5, sp6 = 2,
                           sp7 = 6, sp8 = 3, sp9 = 7, sp10 = 8, sp11 = 2, sp12 = 3)
  state_var <- data.frame(rbind(EDR_var[, -c("EDR", "traj", "state")], target_var))

  # Compute PETRA-EDR
  petra <- petra_edr(state_var = state_var,
                     trajectories = c(EDR_var$traj, "target"),
                     states = as.integer(c(EDR_var$state, 0)),
                     targets = "target",
                     k = 5L,
                     minPts = 2L,
                     d_function = "vegan::vegdist",
                     d_args = list(x = state_var, method = "bray"),
                     return_args = TRUE,
                     direction = 2)

  # Calculate MPD
  MPD(x = petra)
}
```

---

petra_edr *Predicted Ecological Trajectories in Ecological Dynamic Regimes (PETRA-EDR)*

Description

Predicts ecological trajectories from a target state or a sequence of states based on their k-nearest states in the trajectories forming a reference ecological dynamic regime (EDR) and the previous and following states in their respective trajectories.

Usage

```
petra_edr(
  state_var,
  trajectories,
```

```
  states,
  targets,
  d_function,
  d_args,
  d = NULL,
  k,
  eps = NULL,
  minPts = k,
  w_function = NULL,
  alpha = 1,
  w = NULL,
  method = "mean",
  direction = 2,
  return_args = FALSE
)
```

Arguments

state_var	Object containing the state variables for each trajectory state in both the reference dynamic regime and the targets. The class must match the one required in d_function. It can be a list if that is required in d_function.
trajectories	Vector indicating the trajectory or site to which each state in state_var belongs.
states	Vector of integers indicating the order of the states in state_var for each trajectory.
targets	Vector indicating the trajectory or site of the target or targets for which longer trajectories must be predicted.
d_function	Either a function or a non-empty character string naming the function to be called to compute state dissimilarities (see do.call). The output of d_function must be an object of class dist.
d_args	A list of arguments to the d_function call (see do.call).
d	Either a symmetric matrix or an object of class dist containing the dissimilarities between each pair of states in state_var. The elements need to follow the order indicated by trajectories and states. If NULL, d is calculated by calling d_function.
k	Vector of integers indicating the number of near states to the target in the trajectories composing the reference ecological dynamic regime.
eps	Numeric vector indicating the dissimilarity threshold for each target beyond which trajectories in the dynamic regime are not considered.
minPts	Vector of integers including the minimum number of states required to predict preceding and following states for each target.
w_function	String indicating the weighting function to estimate the state variables of the predicted states for each target: "linear", "power", "exponential", "Gaussian", "hyperbolic", "spherical".
alpha	Numeric value of the shape parameter used by w_function (required for "power", "exponential", and "hyperbolic").

`w`	Numeric vector of the same length as `trajectories` giving the weights to estimate the state variables in the predicted trajectory. If `targets` is a vector of two or more elements, list of the same length than `targets` containing a numeric vector with trajectory weights for each target.
`method`	String naming the method used to compute the states forming the predicted trajectory: `"mean"` (default) or `"medoid"`.
`direction`	String or integer indicating whether the prediction is done backward (`"backward"` or `-1`), forward (`"forward"` or `1`), or in both directions (`"both"` or `2`, default).
`return_args`	Logic value indicating whether a list containing the arguments provided in the function call must be returned.

Details

The PETRA-EDR algorithm (`petra_edr()`) is based on the search for the `k` nearest states of the trajectories forming a reference EDR to the target and a moving window towards both directions of their corresponding trajectories.

First, PETRA-EDR looks for the target's `k` nearest states in the trajectories forming the reference EDR. Alternatively, PETRA-EDR may consider all states in a pre-defined radius `eps` by assigning `k` the total number of states in the trajectories of the reference EDR.

Then, consecutive states forming the predicted trajectory are defined based on the previous and following states of the `k` nearest states in their respective trajectories.

Predicted states are defined by averaging the values of the state variables of at least `minPts` states in the trajectories to which the k-nearest states belong (`method = "mean"`) or using the state variables of the medoid of those states (`method = "medoid"`).

To estimate a weighted mean of the state variables, it is necessary to indicate the weight (`w`) that must be used in each trajectory of the EDR. Different weights can be used for each target by providing a list of numeric vectors with the weights assigned to each trajectory state. Alternatively, one can specify the weighting function to be applied depending on the dissimilarity between the target and the states in the trajectories forming the EDR:

`"linear"`

$$w(d_i) = 1 - \frac{d_i}{d_{max}}$$

`"power"`

$$w(d_i) = 1 - (\frac{d_i}{d_{max}})^{\alpha}$$

`"exponential"`

$$w(d_i) = e^{\frac{-\alpha d_i}{d_{max}}}$$

`"Gaussian"`

$$w(d_i) = e^{-(\frac{d_i}{d_{max}})^2}$$

"hyperbolic"

$$w(d_i) = \frac{(1 + \frac{d_i}{d_{max}})^{-\alpha} - 2^{-\alpha}}{1 - 2^{-\alpha}}$$

"spherical"

$$w(d_i) = 1 - 1.5\frac{d_i}{d_{max}} + 0.5(\frac{d_i}{d_{max}})^3$$

where d_i is the dissimilarity between the target and the trajectory i, d_{max} is the maximum dissimilarity between the target and all trajectories, and α is a shape parameter.

Value

The function `petra_edr()` returns an object of class `PETRA`, which is a list with elements containing the following information:

`arguments` List of arguments as specified when the function is called (only if `return_args = TRUE`).

`k_dist` Data frame of five columns including `target`: identifier of the target provided in the argument `targets`; `target_state`: indices of the states in the extremes of the `targets`; `k_trajectories`: vector indicating the trajectory or site to which each of the k-nearest states belongs; `k_states`: vector indicating the order of the k-nearest states in their trajectory (`k_trajectories`); `k_dist`: dissimilarity between the target and each of the k-nearest states.

`predicted_dist` Data frame of six columns including, for each predicted state (`predicted_state`) of the target (`target`) the number of states in the EDR used in the averaging process to calculate the predicted states (`N`; $N \geq MinPts$) and the mean (`mean_dist`), standard deviation (`sd_dist`), minimum (`min_dist`), and maximum (`max_dist`) dissimilarities to the predicted states.

`state_var` Updated `state_var` object containing the state variables of the states forming the predicted trajectory/ies.

`trajectories` Vector indicating the target to which each state in the predicted `state_var` object belongs.

`states` Vector of integers indicating the order of the states in the predicted `state_var` object for each target. In order to keep the original state value of the targets, there can be values lower or equal to zero.

Author(s)

Martina Sánchez-Pinillos

References

Sánchez-Pinillos M., Fortin, M-J., Messier, C., Kneeshaw, D. 2026. Forecasting ecological trajectories from ecological dynamic regimes to improve resilience analysis. *Methods in Ecology and Evolution*. doi:10.1111/2041-210x.70372

See Also

MPD() for estimating the prediction accuracy of petra-edr() outputs.

plot.PETRA() for plotting predicted trajectories in an ordination space representing the state space of the EDR.

Examples

```
if (requireNamespace("vegan", quietly = TRUE)) {
  # Example 1 ----------------------------------------------------------------
  # Compute the predicted trajectory of a target composed of one state

  # State variables for the states in the trajectories forming the EDR
  EDR_var <- EDR_data$EDR1$abundance

  # State variables of the target
  target_var <- data.table::data.table(sp1 = 30, sp2 = 10, sp3 = 13, sp4 = 12,
                                       sp5 = 5, sp6 = 2, sp7 = 6, sp8 = 3,
                                       sp9 = 7, sp10 = 8, sp11 = 2, sp12 = 3)

  # Define state_var including the state variables for the states in the EDR and
  # the target
  state_var <- data.frame(rbind(EDR_var[, -c("EDR", "traj", "state")], target_var))

  # Define the function used to calculate state dissimilarities and its arguments
  # For example, the Canberra dissimilarity.
  d_function = "vegan::vegdist"
  d_args = list(x = state_var, method = "canberra")

  # Compute PETRA-EDR
  petra <- petra_edr(state_var = state_var,
                     trajectories = c(EDR_var$traj, "target"),
                     states = as.integer(c(EDR_var$state, 1)),
                     targets = "target",
                     k = 5L,
                     minPts = 2L,
                     d_function = d_function,
                     d_args = d_args)

  # Example 2 ----------------------------------------------------------------
  # Compute the predicted trajectory of two targets using different parameter
  # values

  # State variables for the states in the trajectories forming the EDR
  EDR_var <- EDR_data$EDR1$abundance

  # State variables of the target states
  target_var <- data.table::data.table(
    traj = c("target1", "target1", "target2"), state = c(1, 2, 1),
    sp1 = c(30, 31, 3), sp2 = c(10, 9, 70), sp3 = c(13, 8, 3), sp4 = c(12, 9, 4),
    sp5 = c(5, 4, 4), sp6 = c(2, 3, 3), sp7 = c(6, 7, 2), sp8 = c(3, 5, 2),
    sp9 = c(7, 6, 3), sp10 = c(8, 7, 3), sp11 = c(2, 3, 2), sp12 = c(3, 3, 2))
```

```
  # Define state_var including the state variables for the states in the EDR and
  # the targets
  state_var <- data.frame(rbind(EDR_var[, -c("EDR", "traj", "state")],
                                target_var[, -c("traj", "state")]))

  # Define trajectories and states in state_var and the ID of the targets
  trajectories <- c(EDR_var$traj, target_var$traj)
  states <- as.integer(c(EDR_var$state, target_var$state))
  targets <- c("target1", "target2")

  # Define the function used to calculate state dissimilarities and its arguments
  d_function = "vegan::vegdist"
  d_args <- list(x = state_var, method = "bray")

  # Compute PETRA-EDR
  petra <- petra_edr(state_var = state_var,
                     trajectories = trajectories,
                     states = states,
                     targets = targets,
                     k = c(5L, 3L),
                     minPts = c(2L, 3L),
                     eps = c(NA, 0.6),
                     d_function = d_function,
                     d_args = d_args,
                     method = "mean",
                     w_function = c("exponential", NA),
                     alpha = c(3, NA))
}
```

`plot.PETRA` *Plot predicted trajectories in Ecological Dynamic Regimes*

Description

Represent predicted trajectories in the state space of an Ecological Dynamic Regime (EDR) distinguishing between the observed states belonging to the target and the forecasted states using the function `petra_edr()`.

Usage

```
## S3 method for class 'PETRA'
plot(
  x,
  petra.colors = "red",
  target.colors = NULL,
  traj.colors = "grey",
  uncert.metric = NULL,
```

```
  uncert.colors = NULL,
  uncert.range = NULL,
  coord = NULL,
  trajectories = NULL,
  states = NULL,
  axes = c(1, 2),
  ...
)
```

Arguments

x	Object of class PETRA returned by petra_edr() using return_args = T.
petra.colors	Specification for the color of all predicted trajectories (defaults red) or a vector with length equal to the number of predicted trajectories in x indicating the color for each predicted trajectory.
target.colors	Specification for the color of the observed states (target) within a predicted trajectory in x or a vector with length equal to the number of predicted trajectories in x indicating a color for each target.
traj.colors	Specification for the color of the observed trajectories in the EDR (defaults grey).
uncert.metric	Column name of x$predicted_dist used to display predicted states with a gradient color depending on that metric. One of "N", "mean_dist", "min_dist", "max_dist".
uncert.colors	Vector of colors used to generate a gradient depending on the values of uncert.metric.
uncert.range	Numeric vector including the minimum and maximum values to scale uncert.metric.
coord	Data frame containing the coordinates of all trajectory states in an ordination space (including the predicted states).
trajectories	Vector indicating the trajectory or site to which each state in coord belongs.
states	Vector of integers indicating the order of the states in coord for each trajectory.
axes	An integer vector indicating the pair of axes in the ordination space to be plotted.
...	Arguments for plot_edr().

Value

Graphical representation of a set of individual trajectories and the predicted trajectories in an ordination space defined by coord or calculated by applying metric multidimensional scaling (mMDS; Borg and Groenen, 2005) to a dissimilarity matrix computed from the arguments used in petra_edr() to generate x.

Author(s)

Martina Sánchez-Pinillos

References

Borg, I., & Groenen, P. J. F. (2005). Modern Multidimensional Scaling (2nd ed.). Springer.

Sánchez-Pinillos M., Fortin, M-J., Messier, C., Kneeshaw, D. 2026. Forecasting ecological trajectories from ecological dynamic regimes to improve resilience analysis. *Methods in Ecology and Evolution*. doi:10.1111/2041-210x.70372

See Also

petra_edr() for predicting ecological trajectories from EDRs.

MPD() for estimating the prediction accuracy of petra-edr() outputs.

plot_edr() for representing EDRs in a multidimensional state space.

Examples

```
if (requireNamespace("vegan", quietly = TRUE)) {

  # Compute PETRA-EDR -------------------------------------------------------

  # State variables of the states in the EDR and the target
  EDR_var <- EDR_data$EDR3$abundance
  target_var <- EDR_data$EDR3_disturbed$abundance[disturbed_states == 0]

  # Define state_var including the state variables for the states in the EDR and
  # the targets
  state_var <- data.frame(rbind(EDR_var[, paste0("sp", 1:12)],
                                target_var[, paste0("sp", 1:12)]))

  # Compute PETRA-EDR for each target. Set return_args = TRUE
  petra <- petra_edr(state_var = state_var,
                     trajectories = c(EDR_var$traj, target_var$traj),
                     states = c(EDR_var$state, target_var$state),
                     targets = unique(target_var$traj),
                     k = 5L, minPts = 2L,
                     d_function = "vegan::vegdist",
                     d_args = list(x = state_var, method = "bray"),
                     return_args = TRUE)

  # Recalculate state_var and d, including the data for the predicted trajectories
  state_var <- rbind(EDR_var[, paste0("sp", 1:12)],
                     petra$state_var)

  d <- vegan::vegdist(state_var, method = "bray")

  # Compute PCoA (optional)
  pcoa <- cmdscale(d = d)#'

  # Example 1 ---------------------------------------------------------------

  # Display each predicted trajectories with different colors
  plot(x = petra,
```

```
     petra.colors = c("red", "royalblue", "seagreen"),
     traj.colors = "grey",
     coord = pcoa,
     trajectories = c(EDR_var$traj, petra$trajectories),
     states = as.integer(c(EDR_var$state, petra$states)),
     xlab = "MDS D1", ylab = "MDS D2",
     main = "Predicted trajectories in EDR")
legend("bottomleft", legend = unique(target_var$traj),
       lwd = 2,
       col = c("red", "royalblue", "seagreen"),
       title = "Target")

# Example 2 ----------------------------------------------------------------

# Identify the observed states with a different color
plot(x = petra,
     petra.colors = "black",
     target.colors = c("red", "royalblue", "seagreen"),
     traj.colors = "grey",
     coord = pcoa,
     trajectories = c(EDR_var$traj, petra$trajectories),
     states = as.integer(c(EDR_var$state, petra$states)),
     xlab = "MDS D1", ylab = "MDS D2",
     main = "Observed and predicted states")
legend("bottomleft",
       legend = c(paste0("Target ", unique(target_var$traj)), "Predicted states"),
       lwd = 2,
       col = c("red", "royalblue", "seagreen", "black"))

# Example 3 ----------------------------------------------------------------

# Display predicted states based on an uncertainty metric
plot(x = petra,
     petra.colors = "black",
     target.colors = "black",
     traj.colors = "grey",
     uncert.metric = "N",
     uncert.colors = hcl.colors(4, rev = TRUE),
     coord = pcoa,
     trajectories = c(EDR_var$traj, petra$trajectories),
     states = as.integer(c(EDR_var$state, petra$states)),
     xlab = "MDS D1", ylab = "MDS D2",
     main = "Uncertainty of the predicted states")
legend("bottomleft", legend = c(paste0("N = ", min(petra$predicted_dist$N)),
                                rep(NA, 18),
                                paste0("N = ", max(petra$predicted_dist$N))),
       fill = hcl.colors(20, rev = TRUE), border = NA, y.intersp = 0.2)
}
```

plot.RETRA *Plot representative trajectories of Ecological Dynamic Regimes*

Description

Plot representative trajectories of an Ecological Dynamic Regime (EDR) in the state space distinguishing between the segments belonging to real trajectories of the EDR and the artificial links between segments.

Usage

```
## S3 method for class 'RETRA'
plot(
  x,
  d,
  trajectories,
  states,
  select_RT = NULL,
  traj.colors = NULL,
  RT.colors = NULL,
  sel.color = NULL,
  link.color = NULL,
  link.lty = 2,
  axes = c(1, 2),
  ...
)
```

Arguments

x	Object of class RETRA.
d	Symmetric matrix or dist object containing the dissimilarities between each pair of states of all trajectories in the EDR or data frame containing the coordinates of all trajectory states in an ordination space.
trajectories	Vector indicating the trajectory or site to which each state in d belongs.
states	Vector of integers indicating the order of the states in d for each trajectory.
select_RT	Optional string indicating the name of a representative trajectory that must be highlighted in the plot. By default (select_RT = NULL), all representative trajectories are represented with the same color.
traj.colors	Specification for the color of all individual trajectories (defaults "grey") or a vector with length equal to the number of trajectories indicating the color for each individual trajectory.
RT.colors	Specification for the color of representative trajectories (defaults "black").
sel.color	Specification for the color of the selected representative trajectory (defaults "red"). Only if !is.null(select_RT).

link.color	Specification for the color of the links between trajectory segments forming representative trajectories. By default, the same color than RT.colors is used.
link.lty	The line type of the links between trajectory segments forming representative trajectories. Defaults 2 = "dashed" (See graphics::par).
axes	An integer vector indicating the pair of axes in the ordination space to be plotted.
...	Arguments for generic plot().

Value

The function plot() plots a set of individual trajectories and the representative trajectories in an ordination space defined through d or calculated by applying metric multidimensional scaling (mMDS; Borg and Groenen, 2005) to d.

Author(s)

Martina Sánchez-Pinillos

References

Borg, I., & Groenen, P. J. F. (2005). Modern Multidimensional Scaling (2nd ed.). Springer.

Sánchez-Pinillos, M., Kéfi, S., De Cáceres, M., Dakos, V. 2023. Ecological Dynamic Regimes: Identification, characterization, and comparison. *Ecological Monographs*. doi:10.1002/ecm.1589

See Also

retra_edr() for identifying representative trajectories in EDRs applying RETRA-EDR.

define_retra() for defining representative trajectories from a subset of segments or trajectory features.

summary() for summarizing representative trajectories in EDRs.

Examples

```
# Example 1 ------------------------------------------------------------------

# d contains the dissimilarities between trajectory states
d <- EDR_data$EDR1$state_dissim

# trajectories and states are defined according to `d` entries.
trajectories <- EDR_data$EDR1$abundance$traj
states <- EDR_data$EDR1$abundance$state

# x defined from retra_edr(). We obtain three representative trajectories.
RT <- retra_edr(d = d, trajectories = trajectories, states = states, minSegs = 5)
summary(RT)

# Plot individual trajectories in blue and representative trajectories in orange,
# "T2" will be displayed in green. Artificial links will be displayed with a
# dotted line.
```

```
plot(x = RT, d = d, trajectories = trajectories, states = states, select_RT = "T2",
     traj.colors = "lightblue", RT.colors = "orange", sel.color = "darkgreen",
     link.lty = 3, xlab = "MDS D1", ylab = "MDS D2",
     main = "Representative trajectories in EDR1")

# Example 2 ----------------------------------------------------------------

# d contains the coordinates in an ordination space. For example, we use
# the coordinates of the trajectory states after applying a principal component
# analysis (PCA) to an abundance matrix.
abun <- EDR_data$EDR1$abundance
pca <- prcomp(abun[, -c(1:3)])
coord <- data.frame(pca$x)

# trajectories and states are defined according to the abundance matrix
# used in the PCA
trajectories <- EDR_data$EDR1$abundance$traj
states <- EDR_data$EDR1$abundance$state

# Instead of using the representative trajectories obtained from `retra_edr()`,
# we will define the set of trajectories that we want to highlight. For example,
# we can select the trajectories whose initial and final states are in the
# extremes of the first axis.
T1 <- trajectories[which.max(coord[, 1])]
T2 <- trajectories[which.min(coord[, 1])]
RT_traj <- c(trajectories[trajectories %in% T1],
             trajectories[trajectories %in% T2])
RT_states <- c(states[which(trajectories %in% T1)],
               states[which(trajectories %in% T2)])

# Create a data frame to generate a RETRA object using define_retra
RT_df <- data.frame(RT = c(rep("T1", sum(trajectories %in% T1)),
                           rep("T2", sum(trajectories %in% T2))),
                 RT_traj = RT_traj,
                 RT_states = as.integer(RT_states))
RT_retra <- define_retra(data = RT_df)

# Plot the defined trajectories with the default graphic values
plot(x = RT_retra, d = coord, trajectories = trajectories, states = states,
     main = "Extreme trajectories in EDR1")
```

`plot_edr` *Plot Ecological Dynamic Regimes*

Description

Represents EDR trajectories in the state space. Trajectories and/or states can be displayed in different colors based in a predefined classification or variable.

Usage

```
plot_edr(
  x,
  trajectories,
  states,
  traj.colors = NULL,
  state.colors = NULL,
  variable = NULL,
  type = "trajectories",
  axes = c(1, 2),
  initial = FALSE,
  ...
)
```

Arguments

`x`	Symmetric matrix or `dist` object containing the dissimilarities between each pair of states of all trajectories in the EDR. Alternatively, data frame containing the coordinates of all trajectory states in an ordination space.
`trajectories`	Vector indicating the trajectory or site to which each state in `x` belongs.
`states`	Vector of integers indicating the order of the states in `x` for each trajectory.
`traj.colors`	Specification for the color of all individual trajectories (defaults "grey") or a vector with length equal to the number of different trajectories indicating the color for each individual trajectory.
`state.colors`	Specification for the color of all trajectory states (defaults equal to `traj.colors`), vector with length equal to the number of states indicating the color for each trajectory state, or vector of colors used to generate a gradient depending on the values of `variable` (if `type = "gradient"`).
`variable`	Numeric vector with equal length to the number of states to be represented using a gradient of state colors (if `type = "gradient"`).
`type`	One of the following `"trajectories"`, `"states"`, or `"gradient"`.
`axes`	An integer vector indicating the pair of axes in the ordination space to be plotted.
`initial`	Flag indicating if the initial state must be plotted (only if `type = "states"` or `type = "gradient"`)
`...`	Arguments for generic `plot()`.

Value

`plot_edr()` permits representing the trajectories of an Ecological Dynamic Regime using different colors for each trajectory or state.

Author(s)

Martina Sánchez-Pinillos

See Also

plot.RETRA() for plotting representative trajectories in an ordination space representing the state space of the EDR.

Examples

```
# Data
state_variables <- EDR_data$EDR1$abundance
d <- EDR_data$EDR1$state_dissim

# Coordinates in classic multidimensional scaling
x <- cmdscale(d, k = 3)

# Plot trajectories 1-10 in "coral", 11-20 in "blue" and 21-30 in "gold"
plot_edr(x = x, trajectories = state_variables$traj,
         states = as.integer(state_variables$state),
         traj.colors = c(rep("coral", 10), rep("royalblue", 10), rep("gold", 10)),
         xlab = "PCoA 1", ylab = "PCoA 2",
         main = "type = 'trajectories'")
legend("bottomleft", legend = paste0("Trajectories ", c("1-10", "11-20", "21-30")),
       lty = 1, col = c("coral", "royalblue", "gold"), bty = "n", cex = 0.9)

# Plot states with different colors depending on the state value
plot_edr(x = x, trajectories = state_variables$traj,
         states = as.integer(state_variables$state),
         traj.colors = NULL, initial = TRUE,
         state.colors = rep(c("coral2", "azure3", "azure3", "azure3", "royalblue4"),
                            length(unique(state_variables$traj))),
         xlab = "PCoA 1", ylab = "PCoA 2",
         type = "states", main = "type = 'states'")
legend("bottomleft", legend = c("State 1", "States 2-4", "State 5"),
       pch = 15, col = c("coral2", "azure3", "royalblue4"), bty = "n", cex = 0.9)

# Plot states with different colors depending on the abundance of sp1
plot_edr(x = x, trajectories = state_variables$traj,
         states = as.integer(state_variables$state),
         traj.colors = NULL,
         state.colors = c("yellow", "orange2", "purple4"),
         variable = state_variables$sp1,
         xlab = "PCoA 1", ylab = "PCoA 2",
         type = "gradient", main = "type = 'gradient'", initial = TRUE)
legend("bottomleft",
       legend = c(paste0("abun sp1 = ", min(state_variables$sp1)),
                  rep(NA, 28),
                  paste0("abun sp1 = ", max(state_variables$sp1))),
       fill = colorRampPalette(c("yellow", "orange2", "purple4"))(30),
       border = NA, y.intersp = 0.2, cex = 0.9, bty = "n")
```

resilience_metrics *Metrics of trajectory deviation with respect to a reference trajectory*

Description

Set of metrics to analyze the deviation of disturbed trajectories from an ecological dynamic regime (EDR) considering a representative trajectory or a predicted trajectory as the reference. These metrics include the resistance to the disturbance, amplitude, recovery, and net change. Altogether, these indices inform on the resilience of a system.

Usage

```
resistance(
  d,
  trajectories,
  states,
  disturbed_trajectories,
  disturbed_states,
  predisturbed_states = disturbed_states - 1
)

amplitude(
  d,
  trajectories,
  states,
  disturbed_trajectories,
  disturbed_states,
  predisturbed_states = disturbed_states - 1,
  reference,
  state_var = NULL,
  index = c("absolute", "relative"),
  method = "nearest_state"
)

recovery(
  d,
  trajectories,
  states,
  disturbed_trajectories,
  disturbed_states,
  reference,
  state_var = NULL,
  index = c("absolute", "relative"),
  method = "nearest_state"
)

net_change(
```

```
  d,
  trajectories,
  states,
  disturbed_trajectories,
  disturbed_states,
  predisturbed_states = disturbed_states - 1,
  reference,
  state_var = NULL,
  index = c("absolute", "relative"),
  method = "nearest_state"
)
```

Arguments

- `d` — Either a symmetric matrix or an object of class `dist` containing the dissimilarities between each pair of states.
- `trajectories` — Vector indicating the trajectory or site to which each state in `d` and `state_var` belongs.
- `states` — Vector of integers indicating the order of the states in `d` and `state_var` for each trajectory.
- `disturbed_trajectories` — Vector of the same class as `trajectories` indicating the identifier of the disturbed trajectories.
- `disturbed_states` — Vector of integers included in `states` indicating the first state after the release of the disturbance for each value in `disturbed_trajectories`.
- `predisturbed_states` — Vector of integers included in `states` indicating the last undisturbed state of each `disturbed_trajectories`. The previous states to `disturbed_states` are considered by default.
- `reference` — Object of class `RETRA` or `PETRA` indicating the trajectory taken as the reference to compute amplitude, recovery, and net_change of the disturbed trajectories (see Details). `PETRA` objects must include the arguments (`return_args = TRUE`).
- `state_var` — Object containing the state variables for each trajectory state (required if `reference` is of class `PETRA`).
- `index` — Method to calculate amplitude, recovery, or net change (`"absolute"`, `"relative"`; see Details).
- `method` — Method to calculate the distance between the `disturbed_states` or `predisturbed_states` and the `reference` trajectory. One of `"nearest_state"`, of `"projection"` (see Details).

Details

Resistance (`resistance()`)

Resistance captures the immediate impact of the disturbance as a function of the changes in the state variables (Sánchez-Pinillos et al., 2019).

$Rt = 1 - d_{pre,dist}$

Amplitude (`amplitude()`)

Amplitude indicates the direction in which the system is deviated during the disturbance in relation to the `reference` (Sánchez-Pinillos et al., 2024; Sánchez-Pinillos et al., 2026).

If `reference` is of class `RETRA`: Positive values indicate that the disturbance deviates the system towards the boundaries of the dynamic regime. Negative values indicate that the disturbance deviates the system towards the representative trajectory.

If `reference` is of class `PETRA`: Only values equal or greater than zero are possible. Positive values indicate that the system deviates from its predicted trajectory, whereas amplitude equal to zero indicates that the system remains on its predicted trajectory.

Two indices can be calculated:

If `index = "absolute"`,

$A = d_{dist,RT} - d_{pre,RT}$

If `index = "relative"`,

$A = \frac{d_{dist,RT} - d_{pre,RT}}{d_{pre,dist}}$

Recovery (`recovery()`)

Recovery quantifies the ability of the system to evolve towards the `reference` following the relief of the disturbance (if positive) or move in the direction of the boundaries of the dynamic regime (if negative and `reference` is of class `RETRA`) or elsewhere (if negative and `reference` is of class `PETRA`) (Sánchez-Pinillos et al., 2024; Sánchez-Pinillos et al., 2026).

Two indices can be calculated:

If `index = "absolute"`,

$Rc = d_{dist,RT} - d_{post,RT}$

If `index = "relative"`,

$Rc = \frac{d_{dist,RT} - d_{post,RT}}{d_{dist,post}}$

Net change (`net_change()`)

Net change quantifies the proximity of the system to the `reference` relative to the pre-disturbed state after the disturbance (Sánchez-Pinillos et al., 2024; Sánchez-Pinillos et al., 2026).

If reference is of class `RETRA`: Positive values indicate that the system eventually evolves towards the boundaries of the dynamic regime. Negative values indicate that the system eventually evolves towards the `reference`.

If `reference` is of class `PETRA`: Only values equal or greater than zero are possible. Positive values indicate that the system eventually deviates from its predicted trajectory, whereas net change equal to zero indicates that the system reaches its predicted trajectory some time after the disturbance.

Two indices can be calculated:

If `index = "absolute"`,

$NC = d_{post,RT} - d_{pre,RT}$

If `index = "relative"`,

$NC = \frac{d_{post,RT} - d_{pre,RT}}{d_{pre,post}}$

In all cases:

- $d_{pre,RT}$ is the dissimilarity between the `predisturbed_states` and the `reference`.
- $d_{dist,RT}$ is the dissimilarity between the `disturbed_states` and the `reference`.
- $d_{post,RT}$ is the dissimilarity between the states after `disturbed_states` and the `reference`.
- $d_{pre,dist}$ is the dissimilarity contained in `d` between the `predisturbed_states` and the `disturbed_states`.
- $d_{dist,post}$ is the dissimilarity contained in `d` between the `disturbed_states` and the post-disturbed states.
- $d_{pre,post}$ is the dissimilarity contained in `d` between the `predisturbed_states` and the post-disturbed states.

$d_{pre,RT}$, $d_{dist,RT}$, and $d_{post,RT}$ are calculated using the function `state_to_trajectory()` by two different methods:

- If `method = "nearest_state"`, $d_{pre,RT}$, $d_{dist,RT}$, and $d_{post,RT}$ are calculated as the dissimilarity between the pre-disturbance, disturbed, or post-disturbance states and their nearest state in the `reference`.
- If `method = "projection"`, $d_{pre,RT}$, $d_{dist,RT}$, and $d_{post,RT}$ are calculated as the dissimilarity between the pre-disturbance, disturbed, or post-disturbance states and their projection onto the `reference` or using the nearest state of the `reference` if the dissimilarity is smaller.

Value

- `resistance()` returns a data frame of two columns indicating the resistance value (`Rt`) for each `disturbed_trajectory`.
- `amplitude()` returns a data frame of three columns indicating the amplitude value (`A_abs`; `A_rel`) for each `disturbed_trajectory` and `reference`. If `index = c("absolute", "relative")`, both values are included in a data frame of four columns.
- `recovery()` returns a data frame of four columns indicating the recovery value (`Rc_abs`; `Rc_rel`) for each `disturbed_trajectory`, post-disturbance state (`state`) and `reference`. If `index = c("absolute", "relative")`, both values are included in a data frame of five columns.
- `net_change` returns a data frame of four columns indicating the net change value (`NC_abs`; `NC_rel`) for each `disturbed_trajectory`, post-disturbance state (`state`), and `reference`. If `index = c("absolute", "relative")`, both values are included in a data frame of five columns.

Author(s)

Martina Sánchez-Pinillos

References

Sánchez-Pinillos, M., Leduc, A., Ameztegui, A., Kneeshaw, D., Lloret, F., & Coll, L. (2019). Resistance, resilience or change: Post-disturbance dynamics of boreal forests after insect outbreaks. *Ecosystems* 22, 1886-1901 https://doi.org/10.1007/s10021-019-00378-6

Sánchez-Pinillos, M., Dakos, V., & Kéfi, S. (2024). Ecological dynamic regimes: A key concept for assessing ecological resilience. *Biological Conservation* 289, 110409 https://doi.org/10.1016/j.biocon.2023.110409

Sánchez-Pinillos M., Fortin, M-J., Messier, C., Kneeshaw, D. 2026. Forecasting ecological trajectories from ecological dynamic regimes to improve resilience analysis. *Methods in Ecology and Evolution*. doi:10.1111/2041-210x.70372

See Also

`retra_edr()` to identify representative trajectories in an ecological dynamic regime.

`define_retra()` to generate an object of classRETRA.

`petra_edr()` to predict ecological trajectories in an ecological dynamic regime.

`MPD()` for estimating the prediction accuracy of `petra-edr()` outputs.

`state_to_trajectory()` to calculate the position of a state with respect to a trajectory.

Examples

```
if (requireNamespace("vegan", quietly = TRUE)) {
  # Calculate the resistance of disturbed systems

  # Abundance matrix including disturbed and undisturbed trajectories
  abundance <- rbind(EDR_data$EDR3$abundance,
                     EDR_data$EDR3_disturbed$abundance, fill = TRUE)

  # State dissimilarities (Bray-Curtis) for disturbed and undisturbed trajectories
  d <- vegan::vegdist(abundance[, paste0("sp", 1:12)], method = "bray")

  # Resistance
  Rt <- resistance(d = d, trajectories = abundance$traj, states = abundance$state,
              disturbed_trajectories = unique(abundance[!is.na(disturbed_states)]$traj),
                   disturbed_states = abundance[disturbed_states == 1]$state)

  # Calculate the amplitude, recovery, and net change of disturbed systems taking
  # a representative trajectory as the reference. Analogous metrics can be
  # calculated using a PETRA object as the reference.

  # Identify the representative trajectories of the EDR from undisturbed trajectories
  RT <- retra_edr(d = EDR_data$EDR3$state_dissim,
                  trajectories = EDR_data$EDR3$abundance$traj,
                  states = as.integer(EDR_data$EDR3$abundance$state),
                  minSegs = 5)

  # Amplitude
  A <- amplitude(d = d, trajectories = abundance$traj, states = abundance$state,
              disturbed_trajectories = unique(abundance[!is.na(disturbed_states)]$traj),
               disturbed_states = abundance[disturbed_states == 1]$state, reference = RT)

  # Recovery
  Rc <- recovery(d = d, trajectories = abundance$traj, states = abundance$state,
              disturbed_trajectories = unique(abundance[!is.na(disturbed_states)]$traj),
               disturbed_states = abundance[disturbed_states == 1]$state, reference = RT)

  # Net change
  NC <- net_change(d = d, trajectories = abundance$traj, states = abundance$state,
```

```
                disturbed_trajectories = unique(abundance[!is.na(disturbed_states)]$traj),
                 disturbed_states = abundance[disturbed_states == 1]$state, reference = RT)

}
```

`retra_edr` *Representative trajectories in Ecological Dynamic Regimes (RETRA-EDR)*

Description

`retra_edr()` applies the algorithm RETRA-EDR (Sánchez-Pinillos et al., 2023) to identify representative trajectories summarizing the main dynamical patterns of an Ecological Dynamic Regime (EDR).

Usage

```
retra_edr(
  d,
  trajectories,
  states,
  minSegs,
  dSegs = NULL,
  coordSegs = NULL,
  traj_Segs = NULL,
  state1_Segs = NULL,
  state2_Segs = NULL,
  Dim = NULL,
  eps = 0
)
```

Arguments

`d`	Either a symmetric matrix or an object of class `dist` containing the dissimilarities between each pair of states of all trajectories in the EDR.
`trajectories`	Vector indicating the trajectory or site to which each state in `d` belongs.
`states`	Vector of integers indicating the order of the states in `d` for each trajectory.
`minSegs`	Integer indicating the minimum number of segments in a region of the EDR represented by a segment of the representative trajectory.
`dSegs`	Either a symmetric matrix or an object of class `dist` containing the dissimilarities between every pair of trajectory segments (see Details).
`coordSegs`	Matrix containing the coordinates of trajectory segments (rows) in each axis (columns) of an ordination space (see Details).
`traj_Segs`	Vector indicating the trajectory to which each segment in `dSeg` and/or `coordSegs` belongs. Only required if `dSegs` or `coordSegs` are not `NULL`.

`state1_Segs` Vector indicating the initial state of each segment in `dSegs` and/or `coordSegs` according to the values given in `states`. Only required if `dSegs` or `coordSegs` are not `NULL`.

`state2_Segs` Vector indicating the final state of each segment in `dSegs` and/or `coordSegs` according to the values given in `states`. Only required if `dSegs` or `coordSegs` are not `NULL`.

`Dim` Optional integer indicating the number of axes considered to partition the segment space and generate a k-d tree. By default (`Dim = NULL`), all axes are considered.

`eps` Numeric value indicating the minimum length in the axes of the segment space to be partitioned when the k-d tree is generated. If `eps = 0` (default), partitions are made regardless of the size.

Details

The algorithm RETRA-EDR is based on a partition-and-group approach by which it identifies regions densely crossed by ecological trajectories in an EDR, selects a representative segment in each dense region, and joins the representative segments by a set of artificial `Links` to generate a network of representative trajectories. For that, RETRA-EDR splits the trajectories of the EDR into segments and uses an ordination space generated from a matrix containing the dissimilarities between trajectory segments. Dense regions are identified by applying a k-d tree to the ordination space.

By default, RETRA-EDR calculates segment dissimilarities following the approach by De Cáceres et al. (2019) and applies metric multidimensional scaling (mMDS, Borg and Groenen, 2005) to generate the ordination space. It is possible to use other dissimilarity metrics and/or ordination methods and reduce the computational time by indicating the dissimilarity matrix and the coordinates of the segments in the ordination space through the arguments `dSegs` and `coordSegs`, respectively.

- If `!is.null(dSegs)` and `is.null(coordSegs)`, RETRA-EDR is computed by applying mMDS to `dSegs`.
- If `!is.null(dSegs)` and `!is.null(coordSegs)`, RETRA-EDR is directly computed from the coordinates provided in `coordSegs` and representative segments are identified using `dSegs`. `coordSegs` should be calculated by the user from `dSegs`.
- If `is.null(dSegs)` and `!is.null(coordSegs)` (not recommended), RETRA-EDR is directly computed from the coordinates provided in `coordSegs`. As `dSegs` is not provided, `retra_edr()` assumes that the ordination space is metric and identifies representative segments using the Euclidean distance.

Value

The function `retra_edr()` returns an object of class `RETRA`, which is a list of length equal to the number of representative trajectories identified. For each trajectory, the following information is returned:

`minSegs` Value of the `minSegs` parameter.

`Segments` Vector of strings including the sequence of segments forming the representative trajectory. Each segment is identified by a string of the form `traj[st1-st2]`, where `traj` is the

identifier of the original trajectory to which the segment belongs and `st1` and `st2` are identifiers of the initial and final states defining the segment.

`Size` Numeric value indicating the number of states forming the representative trajectory.

`Length` Numeric value indicating the length of the representative trajectory, calculated as the sum of the dissimilarities in `d` between every pair of consecutive states.

`Link_distance` Data frame of two columns indicating artificial links between representative segments (`Link`) and the dissimilarity between the connected states (`Distance`). When two representative segments are linked by a common state or by two consecutive states of the same trajectory, the link distance is zero or equal to the length of a real segment, respectively. In both cases, the link is not considered in the returned data frame.

`Seg_density` Data frame of two columns and one row for each representative segment. `Density` contains the number of segments in the EDR that is represented by each segment of the representative trajectory. `kdTree_depth` contains the depth of the k-d tree for each leaf represented by the corresponding segment. That is, the number of partitions of the ordination space until finding a region with `minSegs` segments or less.

Author(s)

Martina Sánchez-Pinillos

References

Borg, I., & Groenen, P. J. F. (2005). Modern Multidimensional Scaling (2nd ed.). Springer.

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.

Sánchez-Pinillos, M., Kéfi, S., De Cáceres, M., Dakos, V. 2023. Ecological Dynamic Regimes: Identification, characterization, and comparison. *Ecological Monographs*. doi:10.1002/ecm.1589

See Also

`summary()` for summarizing the characteristics of the representative trajectories.

`plot()` for plotting representative trajectories in an ordination space representing the state space of the EDR.

`define_retra()` for defining representative trajectories from a subset of segments or trajectory features.

Examples

```
# Example 1 ------------------------------------------------------------------
# Identify representative trajectories from state dissimilarities

# State dissimilarities (Bray-Curtis) from species abundances
abundance <- data.frame(EDR_data$EDR1$abundance)
d <- EDR_data$EDR1$state_dissim
# (d is equivalent to vegan::vegdist(abundance[, -c(1:3)]))

# Identify the trajectory (or site) and states in d
```

```
trajectories <- abundance$traj
states <- as.integer(abundance$state)

# Compute RETRA-EDR
RT1 <- retra_edr(d = d, trajectories = trajectories, states = states,
                 minSegs = 5)

# Example 2 ----------------------------------------------------------------
# Identify representative trajectories from segment dissimilarities

# Calculate segment dissimilarities using the Hausdorff distance
dSegs <- ecotraj::segmentDistances(ecotraj::defineTrajectories(d = d, sites = trajectories,
                                                               surveys = states),
                                   distance.type = "Hausdorff")
dSegs <- dSegs$Dseg

# Identify the trajectory (or site) and states in dSegs:
# Split the labels of dSegs (traj[st1-st2]) into traj, st1, and st2
seg_components <- strsplit(gsub("\\]", "", gsub("\\[", "-", labels(dSegs))), "-")
traj_Segs <- sapply(seg_components, "[", 1)
state1_Segs <- as.integer(sapply(seg_components, "[", 2))
state2_Segs <- as.integer(sapply(seg_components, "[", 3))

# Compute RETRA-EDR
RT2 <- retra_edr(d = d, trajectories = trajectories, states = states, minSegs = 5,
                dSegs = dSegs, traj_Segs = traj_Segs,
                state1_Segs = state1_Segs, state2_Segs = state2_Segs)
```

`state_to_trajectory` *Position of a state with respect to a trajectory*

Description

Define the position of a state with respect to a reference trajectory based on its distance from the trajectory and the length and direction of the trajectory.

Usage

```
state_to_trajectory(
  d,
  trajectories,
  states,
  target_states,
  reference,
  method,
  coordStates = NULL
)
```

Arguments

d	Either a symmetric matrix or an object of class dist containing the dissimilarities between each pair of states.
trajectories	Vector indicating the trajectory or site to which each state in d belongs.
states	Vector of integers indicating the order of the states in d for each trajectory (assign 1 if the state does not belong to any trajectory).
target_states	Vector of integers indicating the indices in trajectories and states of the ecological states for which their relative position will be calculated.
reference	Vector of the same class of trajectories or object of class RETRA indicating the reference trajectory to calculate the relative position of the target_states.
method	Method to calculate the distance and relative position of the target_states and the reference. One of "nearest_state" or "projection" (see Details).
coordStates	Matrix containing the coordinates of each state (rows) and axis (columns) of a metric ordination space (see Details).

Details

state_to_trajectory() can calculate the distance and relative position of one or more target_states relative to a reference trajectory by two different methods:

- "nearest_state" returns the dissimilarity of the target_states to the nearest state of the reference trajectory (distance) and calculates the relative position of the nearest state within the reference.
- "projection" returns the dissimilarity of the target_states to their projection onto the reference trajectory and calculates the relative position of the projected state within the reference. When the target_states cannot be projected onto any of the segments forming the reference and in cases in which the dissimilarity to nearest state of the reference is smaller than the dissimilarity to the projected state, state_to_trajectory() uses the nearest state in the reference to compute distance and relative_position. This method requires d to be metric (i.e. to satisfy the triangle inequality). If d is not metric, state_to_trajectory() calculates the Euclidean distance within a transformed space generated through multidimensional scaling (Borg and Groenen, 2005). To use the state coordinates in a different metric space, use the coordStates argument.

Value

The function state_to_trajectory() returns a data frame of four columns including the distance and relative_position between the target_state and the reference.

- Depending on the method, distance is calculated as the dissimilarity between the target_states and their respective nearest state in the reference or the dissimilarity to their projections onto the reference.
- The relative_position is a value that ranges between 0 (if the nearest state or projected point coincides with the first reference state) and 1 (if the nearest state or projected point coincides with the last reference state).

Author(s)

Martina Sánchez-Pinillos

Examples

```
# State dissimilarities, trajectories, and states
d <- EDR_data$EDR3$state_dissim
# (d is Equivalent to vegan::vegdist(EDR_data$EDR3$abundance[, -c(1:3)]))
trajectories <- EDR_data$EDR3$abundance$traj
states <- EDR_data$EDR3$abundance$state

# Calculate the representative trajectories of an EDR to be used as reference
RT <- retra_edr(d = d,
               trajectories = trajectories,
               states = states,
               minSegs = 10)

# Define the target states
target_states <- as.integer(c(1, 16, 55))

# Calculate the position of the target states with respect to  the representative
# trajectories of an EDR
state_to_trajectory(d = d, trajectories = trajectories,
                    states = states,
                    target_states = target_states,
                    reference = RT,
                    method = "nearest_state")
```

---

summary.RETRA *Summarize representative trajectories*

Description

Summarize the properties of representative trajectories returned by `retra_edr()` or `define_retra()`

Usage

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

Arguments

`object`	An object of class `RETRA`.
`...`	(not used)

Value

Data frame with nine columns and one row for each representative trajectory in `object`. The columns in the returned data frame contain the following information:

`ID` Identifier of the representative trajectories.

`Size` Number of states forming each representative trajectory.

`Length` Sum of the dissimilarities in `d` between every pair of consecutive states forming the representative trajectories.

`Avg_link` Mean value of the dissimilarities between consecutive states of the representative trajectories that do not belong to the same ecological trajectory or site (i.e., artificial links).

`Sum_link` Sum of the dissimilarities between consecutive states of the representative trajectories that do not belong to the same ecological trajectory or site (i.e., artificial links).

`Avg_density` Mean value of the number of segments represented by each segment of the representative trajectory (excluding artificial links).

`Max_density` Maximum number of segments represented by at least one of the segments of the representative trajectory (excluding artificial links).

`Avg_depth` Mean value of the k-d tree depths, that is, the number of partitions of the ordination space until finding a region with `minSegs` segments or less.

`Max_depth` Maximum depth in the k-d tree, that is, the number of partitions of the ordination space until finding a region with `minSegs` segments or less.

See Also

`retra_edr()` for identifying representative trajectories in EDRs applying RETRA-EDR.

`define_retra()` for generating an object of class `RETRA` from trajectory features.

Examples

```
# Apply RETRA-EDR to identify representative trajectories
d = EDR_data$EDR1$state_dissim
trajectories = EDR_data$EDR1$abundance$traj
states = EDR_data$EDR1$abundance$state
RT <- retra_edr(d = d, trajectories = trajectories, states = states, minSegs = 5)

# Summarize the properties of the representative trajectories in a data frame
summary(RT)
```

Index

∗ **datasets**
EDR_data, 8

amplitude *(*resilience_metrics*)*, 28

dBD *(*EDR_metrics*)*, 10
dDis *(*EDR_metrics*)*, 10
define_retra, 2
define_retra(), *24*, *32*, *35*, *38, 39*
dEve *(*EDR_metrics*)*, 10
dist, *3*, *6*, *10*, *15*, *29*, *33*, *37*
dist_edr, 6
do.call, *15*

ecotraj::trajectoryDistances(), *7*, *11*
EDR_data, 8
EDR_metrics, 10

graphics::par, *24*

MPD, 13
MPD(), *18*, *21*, *32*

net_change *(*resilience_metrics*)*, 28

petra_edr, 14
petra_edr(), *14*, *21*, *32*
plot(), *3, 4*, *24*, *26*, *35*
plot.PETRA, 19
plot.PETRA(), *14*, *18*
plot.RETRA, 23
plot.RETRA(), *27*
plot_edr, 25
plot_edr(), *20, 21*

recovery *(*resilience_metrics*)*, 28
resilience_metrics, 28
resistance *(*resilience_metrics*)*, 28
retra_edr, 33
retra_edr(), *3, 4*, *24*, *32*, *38, 39*

state_to_trajectory, 36
state_to_trajectory(), *31, 32*
summary(), *4*, *24*, *35*
summary.RETRA, 38