Package: hdMTD (via r-universe)

October 31, 2026

Type Package

Title Inference for High-Dimensional Mixture Transition Distribution Models

Version 0.1.4

Description Estimates parameters in Mixture Transition Distribution (MTD) models, a class of high-order Markov chains. The set of relevant pasts (lags) is selected using either the Bayesian Information Criterion or the Forward Stepwise and Cut algorithms. Other model parameters (e.g. transition probabilities and oscillations) can be estimated via maximum likelihood estimation or the Expectation-Maximization algorithm. Additionally, 'hdMTD' includes a perfect sampling algorithm that generates samples of an MTD model from its invariant distribution. For theory, see Ost & Takahashi (2023) <http://jmlr.org/papers/v24/22-0266.html>.

URL https://github.com/MaiaraGripp/hdMTD

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

Roxygen list(markdown = TRUE)

Depends R (>= 4.1.0)

Imports methods, dplyr, purrr, igraph

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

Config/pak/sysreqs libglpk-dev libxml2-dev

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

Date/Publication 2026-04-25 20:02:42 UTC

RemoteUrl https://github.com/maiaragripp/hdmtd

RemoteRef HEAD

RemoteSha e238820abb03fc16d8d7d539ecd0ce038d4b617e

Contents

as.MTD . . . 2
countsTab . . . 3
dTV_sample . . . 4
empirical_probs . . . 6
freqTab . . . 7
hdMTD . . . 8
hdMTD-methods . . . 10
hdMTD_BIC . . . 12
hdMTD_CUT . . . 14
hdMTD_FS . . . 16
hdMTD_FSC . . . 17
MTD-accessors . . . 19
MTD-methods . . . 20
MTDest . . . 22
MTDest-methods . . . 25
MTDmodel . . . 26
oscillation . . . 28
perfectSample . . . 30
plot.MTD . . . 30
plot.MTDest . . . 32
probs . . . 34
tempdata . . . 35

Index **37**

`as.MTD` *Coerce an EM fit to an MTD model*

Description

Convenience coercion to rebuild an object of class `"MTD"` from an EM fit (i.e., an output from `MTDest()` with class `c("MTDest","MTD")`). This simply feeds the estimated parameters back into `MTDmodel`. Note that most methods for `"MTD"` also work directly on `"MTDest"` objects via inheritance and explicit coercion is therefore optional.

Usage

```
as.MTD(x, ...)
```

Arguments

`x`	An object of class `"MTDest"`.
`...`	Further arguments passed to or from other methods (ignored).

Value

An object of class `"MTD"` as returned by `MTDmodel`.

See Also

MTDest, MTDmodel

Examples

```
## Not run:
  set.seed(1)
  MTD <- MTDmodel(Lambda = c(1, 3), A = c(0, 1), lam0 = 0.05) # generates MTD model
  X <- perfectSample(MTD, N = 400) # generates MTD sample
  init <- list(
    p0 = c(0.4, 0.6),
    lambdas = c(0.05, 0.45, 0.5),
    pj = list(
      matrix(c(0.2, 0.8, 0.45, 0.55), byrow = TRUE, ncol = 2),
      matrix(c(0.25, 0.75, 0.3, 0.7),  byrow = TRUE, ncol = 2)
    )
  )
  fit <- MTDest(X, S = c(1, 3), init = init) # estimates parameters from sample
  m <- as.MTD(fit) # generates an MTD model from estimated parameters
  str(m, max.level = 1)

## End(Not run)
```

---

countsTab *Counts sequences of length d+1 in a sample*

Description

Creates a tibble containing all unique sequences of length d+1 found in the sample, along with their absolute frequencies.

Usage

```
countsTab(X, d)
```

Arguments

X	A numeric vector, a single-column data frame, or a list with a sample from a Markov chain. The first element must be the most recent observation.
d	A positive integer specifying the number of elements in each sequence, which will be d+1. Typically, d represents the chain order or serves as an upper limit for it.

Details

The function generates a tibble with d+2 columns. In the first d+1 columns, each row displays a unique sequence of size d+1 observed in the sample. The last column, called Nxa, contains the number of times each of these sequences appeared in the sample.

The number of rows in the output varies between 1 and $|A|^{d+1}$, where $|A|$ is the number of unique states in X, since it depends on the number of unique sequences that appear in the sample.

Value

A tibble with all observed sequences of length d+1 and their absolute frequencies.

Examples

```
countsTab(c(1,2,2,1,2,1,1,2,1,2), d = 3)

# Using simulated data.
set.seed(1)
M <- MTDmodel(Lambda = c(1, 3), A = c(1, 2), lam0 = 0.05)
X <- perfectSample(M, N = 400)
countsTab(X, d = 2)
```

---

dTV_sample *The total variation distance between distributions*

Description

Calculates the total variation distance between distributions conditioned in a given past sequence.

Usage

```
dTV_sample(S, j, A = NULL, base, lenA = NULL, A_pairs = NULL, x_S)
```

Arguments

S	A numeric vector of positive integers (or NULL) representing a set of past lags. The distributions from which this function will calculate the total variation distance are conditioned on a fixed sequence indexed by S (the user must also input the sequence through the argument x_S).
j	A positive integer representing a lag in the *complement* of S. The symbols indexed by j vary along the state space A, altering the distribution through this single lag, and the size of this change is what this function seeks to measure.
A	A vector of unique nonnegative integers (state space) with at least two elements. A represents the state space. You may leave A=NULL (default) if you provide the function with the arguments lenA and A_pairs (see *Details* below).

base	A data frame with sequences of elements from A and their transition probabilities. base is meant to be an output from function freqTab(), and must be structured as such. The data frame must contain all required transitions conditioned on x_S (i.e. length(A)^2 rows with sequence x_S). See *Details* section for further information.
lenA	An integer >= 2, representing length(A). Required if A is not provided.
A_pairs	A two-column matrix with all unique pairs of elements from A. Required if A is not provided.
x_S	A vector of length length(S) or NULL. If S==NULL, x_S will be set to NULL. x_S represents a sequence of symbols from A indexed by S. This sequence remains constant across the conditional distributions to be compared, representing the fixed configuration of the past.

Details

This function computes the total variation distance between distributions found in base, which is expected to be the output of the function freqTab(). Therefore, base must follow a specific structure (e.g., column names must match, and a column named qax_Sj, containing transition distributions, must be present). For more details on the output structure of freqTab(), refer to its documentation.

The total-variation distance is computed as

$$\tfrac{1}{2} \sum_{a \in \mathcal{A}} \left| \hat{p}(a | x_{-S}, x_{-j} = b) - \hat{p}(a | x_{-S}, x_{-j} = c) \right|,$$

for each pair of symbols b, c in A using the empirical conditional probabilities obtained from freqTab for the supplied S and j.

If you provide the state space A, the function calculates: lenA <- length(A) and A_pairs <- t(utils::combn(A, 2)). Alternatively, you can input lenA and A_pairs directly and let A <- NULL, which is useful in loops to improve efficiency.

Value

A single-row matrix with one column per pair of distinct elements from the state space A (so choose(length(A), 2) columns). Each entry corresponds to the total variation distance between a pair of distributions, conditioned on the same fixed past x_S (when S is not NULL), differing only in the symbol indexed by j, which varies across all distinct pairs of elements in A. When S is NULL, the row name is the empty string "".

Examples

```
set.seed(1)
M <- MTDmodel(Lambda = c(1, 4), A = c(1, 2, 3), lam0 = 0.1)
X <- perfectSample(M, N = 400)
ct <- countsTab(X, d = 5)

# --- Case 1: S non-empty
pbase <- freqTab(S = c(1, 4), j = 2, A = c(1, 2, 3), countsTab = ct)
dTV_sample(S = c(1, 2), j = 4, A = c(1, 2, 3), base = pbase, x_S = c(2, 3))
```

```
# --- Case 2: S = NULL
pbase2 <- freqTab(S = NULL, j = 1, A = c(1, 2, 3), countsTab = ct)
dTV_sample(S = NULL, j = 1, A = c(1, 2, 3), base = pbase2)
```

`empirical_probs` *Estimated transition probabilities*

Description

Computes the Maximum Likelihood estimators (MLE) for an MTD Markov chain with relevant lag set `S`.

Usage

```
empirical_probs(X, S, matrixform = FALSE, A = NULL, warn = FALSE)
```

Arguments

`X`	A vector or single-column data frame containing a sample of a Markov chain (`X[1]` is the most recent).
`S`	A numeric vector of unique positive integers. Typically, `S` represents a set of relevant lags.
`matrixform`	Logical. If `TRUE`, the output is formatted as a stochastic transition matrix.
`A`	A numeric vector of distinct integers representing the state space. If not provided, this function will set `A <- sort(unique(X))`.
`warn`	Logical. If `TRUE`, the function warns the user when the state space is automatically set as `A <- sort(unique(X))`.

Details

The probabilities are estimated as:

$$\hat{p}(a|x_S) = \frac{N(x_S a)}{N(x_S)}$$

where $N(x_S a)$ is the number of times the sequence x_S appeared in the sample followed by a, and $N(x_S)$ is the number of times x_S appeared (followed by any state). If $N(x_S) = 0$, the probability is set to $1/|A|$ (assuming a uniform distribution over `A`).

Value

A data frame or a matrix containing estimated transition probabilities:

- If `matrixform = FALSE`, the function returns a data frame with three columns:
  - The past sequence x_S (a concatenation of past states).
  - The current state a.
  - The estimated probability $\hat{p}(a|x_S)$.
- If `matrixform = TRUE`, the function returns a stochastic transition matrix, where rows correspond to past sequences x_S and columns correspond to states in `A`.

Examples

```
# Simulate a chain from an MTD model
set.seed(1)
M <- MTDmodel(Lambda = c(1, 15, 30), A = c(1, 2, 3), lam0 = 0.05)
X <- perfectSample(M, N = 400)

# Estimate probabilities for different subsets S
empirical_probs(X, S = c(1, 30))
empirical_probs(X, S = c(1, 15, 30), matrixform = TRUE)
```

`freqTab` *A tibble containing sample sequence frequencies and estimated probabilities*

Description

This function returns a tibble containing the sample sequences, their frequencies and the estimated transition probabilities.

Usage

```
freqTab(S, j = NULL, A, countsTab, complete = TRUE)
```

Arguments

`S`	A numeric vector of positive integers or `NULL`. Represents a set of past lags that must be present within the columns of the `countsTab` argument and are to be considered while estimating the transition probabilities. Both `S` and `j` cannot be `NULL` at the same time.
`j`	An integer or `NULL`. Typically represents a lag `j` in the *complement* of `S`. Both `S` and `j` cannot be `NULL` at the same time. See *Details* for further information.
`A`	A vector with nonnegative integers. Must have at least two different entries. `A` represents the state space.
`countsTab`	A tibble or a data frame with all sequences of length d+1 that appear in the sample, and their absolute frequency. This tibble is typically generated by the function `countsTab()`. If using a custom data frame not generated by `countsTab()`, make sure its format and column names match the expected structure; otherwise, errors may occur in `freqTab()`.
`complete`	Logical. If `TRUE` all sequences that did not appear in the sample will be included in the output with frequency equal to 0.

Details

The parameters `S` and `j` determine which columns of `countsTab` are retained in the output. Specifying a lag `j` is optional. All lags can be specified via `S`, while leaving `j = NULL` (default). The output remains the same as when specifying `S` and `j` separately. The inclusion of `j` as a parameter improves clarity within the package's algorithms. Note that `j` cannot be an element of `S`.

Value

A tibble where each row represents a sequence of elements from A. The initial columns display each sequence symbol separated into columns corresponding to their time indexes. The remaining columns show the sample frequencies of the sequences and the MLE (Maximum Likelihood Estimator) of the transition probabilities.

Examples

```
# Reproducible simulated data
set.seed(1)
M <- MTDmodel(Lambda = c(1, 4), A = c(1, 2, 3), lam0 = 0.1)
X <- perfectSample(M, N = 400)
ct <- countsTab(X, d = 5)

# Example with S non-empty and j specified
freqTab(S = c(1, 4), j = 2, A = c(1, 2, 3), countsTab = ct)

# Equivalent to calling with S = c(1,2,4) and j = NULL
freqTab(S = c(1, 2, 4), j = NULL, A = c(1, 2, 3), countsTab = ct)
```

---

hdMTD *Inference in MTD models*

Description

This function estimates the relevant lag set in a Mixture Transition Distribution (MTD) model using one of the available methods. By default, it applies the Forward Stepwise ("FS") method, which is particularly useful in high-dimensional settings.

Usage

```
hdMTD(X, d, method = "FS", ...)
```

Arguments

X	A vector or single-column data frame containing a chain sample.
d	A positive integer representing an upper bound for the chain order.
method	A character string indicating the method for estimating the relevant lag set. The available methods are: "FS" (default), "FSC", "CUT", and "BIC". See the *Details* section and the documentation of the corresponding method functions for more information.
...	Additional arguments for the selected method. If not specified, default values will be used (see *Details*).

Details

The available methods are:

- "FS" (Forward Stepwise): selects the lags by a criterion that depends on their oscillations.
- "CUT": a method that selects the relevant lag set based on a predefined threshold.
- "FSC" (Forward Stepwise and Cut): applies the "FS" method followed by the "CUT" method.
- "BIC": selects the lag set using the Bayesian Information Criterion.

The function dynamically calls the corresponding method function (e.g., `hdMTD_FSC()` for `method = "FSC"`). Additional parameters specific to each method can be provided via `...`, and default values are used for unspecified parameters.

This function serves as a wrapper for the method-specific functions:

- `hdMTD_FS()`, for `method = "FS"`
- `hdMTD_FSC()`, for `method = "FSC"`
- `hdMTD_CUT()`, for `method = "CUT"`
- `hdMTD_BIC()`, for `method = "BIC"`

Any additional parameters (`...`) must match those accepted by the corresponding method function. If a parameter value is not explicitly provided, a default value is used. The main default parameters are:

- `S = seq_len(d)`: Used in "BIC" or "CUT" methods.
- `alpha = 0.05`, `mu = 1`. Used in "CUT" or "FSC" methods.
- `xi = 0.5`. Used in "CUT", "FSC" or "BIC" methods.
- `minl = 1`, `maxl = length(S)`, `byl = FALSE`. Used in "BIC" method. All default values are specified in the documentation of the method-specific functions.

BIC-specific notes. When `method = "BIC"`, the object may carry additional BIC diagnostics:

- `attr(x, "extras")$BIC_out`: exactly the object returned by `hdMTD_BIC()` (its type depends on `BICvalue` and `byl`: named numeric vector when `BICvalue = TRUE`; otherwise a character vector of labels, possibly including `"smallest: <set>"`).
- `attr(x, "BIC_selected_value")`: when `BICvalue = TRUE`, the numeric BIC at the selected lag set (shown by `summary()`).

Value

An integer vector of class `"hdMTD"` (the selected lag set $S \subset \mathbb{N}^+$), with attributes:

- `method`, `d`, `call`, `settings`
- `A`: the state space actually used (either provided `A`, or `sort(unique(X))` if `A = NULL`)
- (for `method="BIC"`) `extras`: a list with `BIC_out` (exactly the object returned by `hdMTD_BIC`)
- (for `method="BIC"` and `BICvalue = TRUE`) `BIC_selected_value`: numeric BIC at the selected set

Methods (S3)

The returned object has class "hdMTD" and supports:

- print and summary: methods for lag-selection outputs (see hdMTD-methods).

Accessors

The selected lag set can be retrieved via S and in the package convention as negative lags via lags.

See Also

hdMTD_FS, hdMTD_FSC, hdMTD_CUT, hdMTD_BIC for the method-specific implementations, and MTD-accessors for accessors.

Examples

```
# Simulate a chain from an MTD model
set.seed(1)
M <- MTDmodel(Lambda = c(1, 4), A = c(1, 3), lam0 = 0.05)
X <- perfectSample(M, N = 400)

# Fit using Forward Stepwise (FS)
S_fs <- hdMTD(X = X, d = 5, method = "FS", l = 2)
print(S_fs)
summary(S_fs)  # shows A used if A = NULL in the call

# Fit using Bayesian Information Criterion (BIC)
hdMTD(X = X, d = 5, method = "BIC", xi = 0.6, minl = 3, maxl = 3)
```

---

hdMTD-methods *Methods for objects of class* "hdMTD"

Description

Printing and summarizing methods for lag-selection results returned by hdMTD.

Arguments

x	An object of class "hdMTD" used in print.hdMTD().
object	An object of class "hdMTD" used in summary.hdMTD().
settings	Logical (summary.hdMTD() only). If TRUE, the printed summary includes the method-specific settings list. Default FALSE.
...	Further arguments passed to or from other methods (ignored).

Details

An object of class "hdMTD" is an integer vector S (selected lags, elements of $\mathbb{N}^+$) with attributes:

- method: one of "FS", "FSC", "CUT", "BIC".
- d: upper bound for the order used in the call.
- call: the matched call that produced the object.
- settings: a (method-specific) list of the arguments actually used.
- A: the state space actually used (provided or inferred).
- (optional) BIC_selected_value for method="BIC" with BICvalue=TRUE.
- (optional) extras$BIC_out for method="BIC" (exactly the output of hdMTD_BIC()).

print() shows the method, d, and the selected set of lags in $\mathbb{N}^+$. summary() prints the call, the estimated lag set, optional BIC diagnostics and (optionally) the method-specific settings when settings = TRUE.

Value

print.hdMTD Invisibly returns the "hdMTD" object.

summary.hdMTD Invisibly returns a named list with fields: call, S, lags, A, method, d, settings, BIC_selected and BIC_out. Relevant information is printed to the console in a readable format.

See Also

hdMTD, hdMTD_FS, hdMTD_FSC, hdMTD_CUT, hdMTD_BIC, S, lags

Examples

```
## Not run:
set.seed(1)
M <- MTDmodel(Lambda = c(1, 4), A = c(1, 3), lam0 = 0.05)
X <- perfectSample(M, N = 400)
S_hat <- hdMTD(X, d = 5, method = "FS", l = 2)
print(S_hat)
summary(S_hat)
S(S_hat); lags(S_hat)

## End(Not run)
```

hdMTD_BIC *The Bayesian Information Criterion (BIC) method for inference in MTD models*

Description

A function for estimating the relevant lag set Λ of a Markov chain using Bayesian Information Criterion (BIC). This means that this method selects the set of lags that minimizes a penalized log likelihood for a given sample, see *References* below for details on the method.

Usage

```
hdMTD_BIC(
  X,
  d,
  S = seq_len(d),
  minl = 1,
  maxl = length(S),
  xi = 1/2,
  A = NULL,
  byl = FALSE,
  BICvalue = FALSE,
  single_matrix = FALSE,
  indep_part = TRUE,
  zeta = maxl,
  warn = FALSE,
  ...
)
```

Arguments

`X`	A vector or single-column data frame containing a chain sample (`X[1]` is the most recent).
`d`	A positive integer representing an upper bound for the chain order.
`S`	A numeric vector of positive integers from which this function will select a set of relevant lags. Typically, `S` is a subset of `1:d`. If `S` is not provided, by default `S=1:d`.
`minl`	A positive integer. `minl` represents the smallest length of any relevant lag set this function might return. If `minl == maxl`, this function will return the subset of `S` of length `minl` with the lowest BIC. If `minl < maxl`, the function will consider subsets ranging from length `minl` to length `maxl` when searching for the subset of `S` with the smallest BIC.
`maxl`	A positive integer equal to or greater than `minl` but less than the number of elements in `S` (`maxl = length(S)` is accepted but in this case the output will always be `S`). `maxl` represents the largest length of any relevant lag set this function might return.

`xi`	The BIC penalization term constant. Defaulted to 1/2. A smaller `xi (near 0)` reduces the impact of overparameterization.
`A`	A vector with positive integers representing the state space. If not informed, this function will set `A=sort(unique(X))`.
`byl`	Logical. If `TRUE`, the function will look for the set with smallest BIC by each length (from `minl` to `maxl`), and return the set with smallest BIC for each length. If `minl==maxl` setting `byl=TRUE` or `FALSE` makes no difference, since the function will only calculate the BIC for sets with `maxl` elements in the relevant lag set.
`BICvalue`	Logical. If `TRUE`, the function will also return the calculated values of the BIC for the estimated relevant lag sets.
`single_matrix`	Logical. If `TRUE`, the chain sample is thought to come from an MTD model where the stochastic matrices p_j are constant across all lags $j \in \Lambda$. In practice, this means the user believes the stochastic matrices for every lag in `S` are the same, which reduces the number of parameters in the penalization term.
`indep_part`	Logical. If `FALSE` there is no independent distribution and $\lambda_0 = 0$ which reduces the number of parameters in the penalization term.
`zeta`	A positive integer representing the number of distinct matrices p_j in the MTD, which affects the number of parameters in the penalization term. Defaulted to `maxl`. See more in *Details*.
`warn`	Logical. If `TRUE`, the function warns the user when `A` is set automatically.
`...`	Additional arguments (not used in this function, but maintained for compatibility with `hdMTD()`.

Details

Criterion. For each candidate lag set T contained in S with size $l = |T|$ where `minl <= l <= maxl`, `hdMTD_BIC()` evaluates

$$BIC(T) = -L_T + xi * df(T) * log(N),$$

where $N = length(X)$ and

$$L_T = \sum_{x_T \in A^T} \sum_{a \in A} N(ax_T) * log(\hat{p}(a|x_T)).$$

The empirical conditionals are

$$\hat{p}(a|x_T) = N(ax_T)/N(x_T),$$

computed from the sample counts (same quantities returned by `freqTab` and `empirical_probs`).

Degrees of freedom. The parameter count $df(T)$ is the number of free parameters of an MTD model with lag set T and state space A, honoring the constraints `single_matrix`, `indep_part`, and `zeta`:

$$df(T) = w_{df} + p0_{df} + |A| * (|A| - 1) * zeta.$$

Here $zeta$ is the number of distinct p_j matrices allowed across lags (by default $zeta = l$; setting `single_matrix = TRUE` forces $zeta = 1$). The weight and independent-part contributions are:

$w_{df} = l$ if `indep_part` is `TRUE`, otherwise $w_{df} = l - 1$; $p0_{df} = |A| - 1$ if `indep_part` is `TRUE`, otherwise $p0_{df} = 0$.

Scale. With `xi = 1/2` (the default), BIC equals one half of the classical Schwarz BIC $-2 * L_T + df(T) * log(N)$; minimizing either criterion selects the same lag set.

Note. The likelihood term L_T sums over the $N - max(T)$ effective transitions, while the penalty uses $log(N)$ (this matches the implementation).

Note that the upper bound for the order of the chain (`d`) affects the estimation of the transition probabilities. If we run the function with a certain order parameter `d`, only the sequences of length `d` that appeared in the sample will be counted. Therefore, all transition probabilities, and hence all BIC values, will be calculated with respect to that `d`. If we use another value for `d` to run the function, even if the output agrees with that of the previous run, its BIC value might change a little.

The parameter `zeta` indicates the the number of distinct matrices pj in the MTD. If `zeta = 1`, all matrices p_j are identical; if `zeta = 2` there exists two groups of distinct matrices and so on. The largest value for `zeta` is `maxl` since this is the largest number of matrices p_j. When `minl<maxl`, for each `minl` $\leq$ `l` $\leq$ `maxl`, `zeta = min(zeta,l)`. If `single_matrix = TRUE` then `zeta` is set to 1.

Value

Returns a vector with the estimated relevant lag set using BIC. It might return more than one set if `minl < maxl` and `byl = TRUE`. Additionally, it can return the value of the penalized likelihood for the outputted lag sets if `BICvalue = TRUE`.

References

Imre Csiszár, Paul C. Shields. The consistency of the BIC Markov order estimator. *The Annals of Statistics*, *28*(6), 1601-1619. doi:10.1214/aos/1015957472

Examples

```
# Simulate a chain from an MTD model
set.seed(1)
M <- MTDmodel(Lambda = c(1, 3), A = c(1, 2), lam0 = 0.05)
X <- perfectSample(M, N = 400)

# Fit using BIC with a single lag
hdMTD_BIC(X, d = 6, minl = 1, maxl = 1)

# Fit using BIC with lag selection and extract BIC value
hdMTD_BIC(X, d = 3, minl = 1, maxl = 2, BICvalue = TRUE)
```

`hdMTD_CUT` *The CUT method for inference in MTD models*

Description

A function that estimates the set of relevant lags of an MTD model using the CUT method.

Usage

```
hdMTD_CUT(
  X,
  d,
  S = seq_len(d),
  alpha = 0.05,
  mu = 1,
  xi = 0.5,
  A = NULL,
  warn = FALSE,
  ...
)
```

Arguments

`X`	A vector or single-column data frame containing a chain sample (`X[1]` is the most recent).
`d`	A positive integer representing an upper bound for the chain order.
`S`	A numeric vector of distinct positive integers from which this function will select a set of relevant lags. Should be a subset of `1:d`. Default is `1:d`.
`alpha`	A positive real number used in the CUT threshold (which determines if two distributions can be considered different). The larger the `alpha`, the greater the distance required to consider that there is a difference between a set of distributions.
`mu`	A positive real number such that `mu` $> (e^{\texttt{mu}} - 1)/2$. `mu` is also a component of the same threshold as `alpha`.
`xi`	A positive real number, `xi` is also a component of the same threshold as `alpha`.
`A`	A vector with positive integers representing the state space. If not informed, this function will set `A <- sort(unique(X))`.
`warn`	Logical. If `TRUE`, the function warns the user when `A` is set automatically.
`...`	Additional arguments (not used in this function, but maintained for compatibility with `hdMTD()`.

Details

The "Forward Stepwise and Cut" (FSC) is an algorithm for inference in Mixture Transition Distribution (MTD) models. It consists in the application of the "Forward Stepwise" (FS) step followed by the CUT algorithm. This method and its steps where developed by Ost and Takahashi and are specially useful for inference in high-order MTD Markov chains. This specific function will only apply the CUT step of the algorithm and return an estimated relevant lag set.

Value

Returns a set of relevant lags estimated using the CUT algorithm.

References

Ost, G. & Takahashi, D. Y. (2023). Sparse Markov models for high-dimensional inference. *Journal of Machine Learning Research*, *24*(279), 1-54. http://jmlr.org/papers/v24/22-0266.html

Examples

```
# Simulate a chain from an MTD model
set.seed(1)
M <- MTDmodel(Lambda = c(1, 4), A = c(1, 3), lam0 = 0.05)
X <- perfectSample(M, N = 400)

# Apply CUT with custom alpha, mu, and xi
hdMTD_CUT(X, d = 4, alpha = 0.02, mu = 1, xi = 0.4)

# Apply CUT with selected lags and smaller alpha
hdMTD_CUT(X, d = 6, S = c(1, 4, 6), alpha = 0.08)
```

---

hdMTD_FS *The Forward Stepwise (FS) method for inference in MTD models*

Description

A function that estimates the set of relevant lags of an MTD model using the FS method.

Usage

```
hdMTD_FS(X, d, l, A = NULL, elbowTest = FALSE, warn = FALSE, ...)
```

Arguments

`X`	A vector or single-column data frame containing a chain sample (`X[1]` is the most recent).
`d`	A positive integer representing an upper bound for the chain order.
`l`	A positive integer specifying the number of lags to be selected as relevant.
`A`	A vector with positive integers representing the state space. If not informed, this function will set `A <- sort(unique(X))`.
`elbowTest`	Logical. If TRUE, the function applies an alternative stopping criterion to determine the length of the set of relevant lags. See *Details* for more information.
`warn`	Logical. If `TRUE`, the function warns the user when `A` is set automatically.
`...`	Additional arguments (not used in this function, but maintained for compatibility with `hdMTD()`.

Details

The "Forward Stepwise" (FS) algorithm is the first step of the "Forward Stepwise and Cut" (FSC) algorithm for inference in Mixture Transition Distribution (MTD) models. This method was developed by Ost and Takahashi. This specific function will only apply the FS step of the algorithm and return an estimated relevant lag set of length `l`.

This method iteratively selects the most relevant lags based on a certain quantity ν. In the first step, the lag in `1:d` with the greatest ν is deemed important. This lag is included in the output, and using this knowledge, the function proceeds to seek the next important lag (the one with the highest ν among the remaining ones). The process stops when the output vector reaches length `l` if `elbowTest=FALSE`.

If `elbowTest = TRUE`, the function will store these maximum ν values at each iteration, and output only the lags that appear before the one with smallest ν among them.

Value

A numeric vector containing the estimated relevant lag set using FS algorithm.

Note

Tie-breaking: if multiple lags share the maximum ν, FS picks the most recent lag (smallest j). Up to version 0.1.2 ties were broken at random, which could cause run-to-run differences.

References

Ost, G. & Takahashi, D. Y. (2023). Sparse Markov models for high-dimensional inference. *Journal of Machine Learning Research*, *24*(279), 1-54. http://jmlr.org/papers/v24/22-0266.html

Examples

```
# Simulate a chain from an MTD model
set.seed(1)
M <- MTDmodel(Lambda = c(1, 3), A = c(1, 2), lam0 = 0.05)
X <- perfectSample(M, N = 400)

# Forward Stepwise with l = 2
hdMTD_FS(X, d = 5, l = 2)

# Forward Stepwise with l = 3
hdMTD_FS(X, d = 4, l = 3)
```

---

hdMTD_FSC *Forward Stepwise and Cut method for inference in MTD models*

Description

A function for inference in MTD Markov chains with FSC method. This function estimates the relevant lag set Λ of an MTD model through the FSC algorithm.

Usage

```
hdMTD_FSC(X, d, l, alpha = 0.05, mu = 1, xi = 0.5, A = NULL, ...)
```

Arguments

X	A vector or single-column data frame containing a chain sample (X[1] is the most recent).
d	A positive integer representing an upper bound for the chain order.
l	A positive integer that sets the number of elements in the output vector.
alpha	A positive real number used in the CUT threshold (which determines if two distributions can be considered different). The larger the alpha, the greater the distance required to consider that there is a difference between a set of distributions. Defaulted to 0.05.
mu	A positive real number such that mu $> (e^{\text{mu}} - 1)/2$. mu is also a component of the same threshold as alpha.
xi	A positive real number, xi is also a component of the same threshold as alpha.
A	A vector with positive integers representing the state space. If not informed, this function will set A <- sort(unique(X)).
...	Additional arguments (not used in this function, but maintained for compatibility with hdMTD().

Details

The "Forward Stepwise and Cut" (FSC) is an algorithm for inference in Mixture Transition Distribution (MTD) models. It consists in the application of the "Forward Stepwise" (FS) step followed by the CUT algorithm. This method and its steps where developed by Ost and Takahashi and are specially useful for inference in high-order MTD Markov chains.

Value

Returns a vector with the estimated relevant lag set using FSC algorithm.

References

Ost, G. & Takahashi, D. Y. (2023). Sparse Markov models for high-dimensional inference. *Journal of Machine Learning Research*, *24*(279), 1-54. http://jmlr.org/papers/v24/22-0266.html

Examples

```
# Simulate a chain from an MTD model
set.seed(1)
M <- MTDmodel(Lambda = c(1, 3), A = c(1, 2), lam0 = 0.05)
X <- perfectSample(M, N = 400)

# Forward Stepwise and Cut with different parameters
hdMTD_FSC(X, d = 4, l = 2)
hdMTD_FSC(X, d = 4, l = 3, alpha = 0.1)
```

MTD-accessors *Accessors for objects of classes* "MTD"*,* "MTDest"*, and/or* "hdMTD"

Description

Public accessors that expose object components without relying on the internal list structure. These accessors are available for "MTD" (model objects), "MTDest" (EM fits), and/or "hdMTD" (lag selection).

Usage

```
pj(object)

p0(object)

lambdas(object)

lags(object)

Lambda(object)

S(object)

states(object)

transitP(object)
```

Arguments

object	An object of class "MTD", "MTDest" or "hdMTD" (as supported by each accessor).

Details

Returned lag sets follow the package convention and are shown as negative integers via lags() (elements of $\mathbb{Z}^-$). For convenience, positive-index accessors are also provided: Lambda() for "MTD" objects and "MTDest" fits, and S() for "MTDest" and "hdMTD" objects (elements of $\mathbb{N}^+$).

The function transitP() returns the global transition matrix, i.e., a convex combination of the independent distribution p0 and the lag-specific transition matrices pj, weighted by lambdas.

Value

pj(object) A list of stochastic matrices (one per lag).

p0(object) A numeric probability vector for the independent component.

lambdas(object) A numeric vector of mixture weights that sums to 1.

lags(object) The lag set (elements of $\mathbb{Z}^-$).

Lambda(object) The set of relevant lags as positive integers (elements of $\mathbb{N}^+$).

`S(object)` For `"MTDest"` and `"hdMTD"`, the set of candidate/estimated lags as positive integers (elements of $\mathbb{N}^+$).

`states(object)` The state space.

`transitP(object)` The global transition matrix P.

Note

Naming conventions reflect the conceptual distinction:

- `Lambda`: The true (known) relevant lag set in `"MTD"` models
- `S`: An estimated or candidate lag set in `"MTDest"` and `"hdMTD"` objects

Most accessors for `"MTD"` also work for `"MTDest"` via inheritance. When needed (e.g., `transitP()`), a specific `"MTDest"` method is provided.

See Also

`MTDmodel`, `MTDest`, `hdMTD`.

Examples

```
## Not run:
## For generating an MTD model
set.seed(1)
m <- MTDmodel(Lambda = c(1, 3), A = c(0, 1))
pj(m); p0(m); lambdas(m); lags(m); Lambda(m); states(m)
transitP(m)
## For an EM fit (using coef(m) as init for simplicity):
X <- perfectSample(m, N = 800)
fit <- MTDest(X, S = c(1, 3), init = coef(m))
pj(fit); p0(fit); lambdas(fit); lags(fit); S(fit); states(fit)
transitP(fit)
## For lag selection:
S_hat <- hdMTD(X, d = 5, method = "FS", l = 2)
S(S_hat); lags(S_hat)

## End(Not run)
```

`MTD-methods` *Methods for objects of class* `"MTD"`

Description

Printing, summarizing, and coefficient-extraction methods for Mixture Transition Distribution (MTD) model objects.

Arguments

x	An object of class "MTD" (for print.MTD(x, ...)).
object	An object of class "MTD".
X	A vector or single-column data frame containing an MTD chain sample (required for logLik.MTD(object, X, ...)). Values must be in the model's state space.
...	Further arguments passed to or from other methods (ignored).

Details

- print.MTD() displays a compact summary of the model: the relevant lag set (shown as negative integers) and the state space.
- summary.MTD() computes and prints a detailed summary of the model, including order, relevant lags, state space, mixture weights, the independent distribution (if present), and a compact preview of the global transition matrix P.
- coef.MTD() extracts parameters as a list with lambdas, pj, and p0 (works for c("MTDest","MTD") objects by inheritance).
- logLik.MTD() computes the log-likelihood of a sample under the model. Since an object of class "MTD" carries only the model parameters, a sample X must be supplied. The method honors constraints such as single_matrix and an independent component (indep_part), and returns an object of class "logLik" with appropriate attributes.

Value

print.MTD Invisibly returns the "MTD" object, after displaying its relevant lag set and state space.

summary.MTD Invisibly returns a named list with fields: call, order, Lambda, states, lags, indep, lambdas, pj, p0 (or NULL), P_dim, and P. The same information is printed to the console in a readable format.

coef.MTD A list with parameters: lambdas, pj, and p0.

logLik.MTD An object of class "logLik" with attributes nobs (number of transitions) and df (free parameters), honoring model constraints such as single_matrix and the independent component (indep_part).

See Also

MTDmodel, MTDest for fitted models (note that "MTDest" objects inherit from "MTD"), transitP, lambdas, pj, p0, lags, Lambda, states, oscillation, perfectSample, probs, logLik

Examples

```
## Not run:
set.seed(1)
m <- MTDmodel(Lambda = c(1, 3), A = c(0, 1), lam0 = 0.05)

print(m)       # compact display: lags (Z^-) and state space
s <- summary(m)
str(s)
```

```
coef(m)        # list(lambdas = ..., pj = ..., p0 = ...)
transitP(m)    # global transition matrix P
pj(m); p0(m); lambdas(m); lags(m); Lambda(m); states(m)

X <- perfectSample(m, N = 400)
logLik(m, X)

## End(Not run)
```

`MTDest` *EM estimation of MTD parameters*

Description

Estimation of MTD parameters through the Expectation Maximization (EM) algorithm.

Usage

```
MTDest(
  X,
  S,
  M = 0.01,
  init,
  iter = FALSE,
  nIter = 100,
  A = NULL,
  oscillations = FALSE
)
```

Arguments

`X`	A vector or single-column data frame containing an MTD chain sample (`X[1]` is the most recent).
`S`	A numeric vector of distinct positive integers, sorted increasingly. Typically, `S` represents a set of relevant lags.
`M`	A stopping point for the EM algorithm. If `M=NULL` the algorithm will run for a total of `nIter` iterations (i.e., no convergence check).
`init`	A list with initial parameters: `p0` (optional), `lambdas` (required), `pj` (required). The entries in `lambdas` are weights for the distribution `p0` and the distributions present in the list `pj`. Therefore, the order in which the elements appear in the vector `lambdas` is important for correct assignment. See *Details*.
`iter`	Logical. If `TRUE` include iteration diagnostics in the output (number of updates `iterations`; vector of absolute log-likelihood changes `deltaLogLik`; and `lastComputedDelta`, the last delta log-likelihood compared against `M`).

nIter	An integer positive number with the maximum number of EM iterations.
A	Optional integer vector giving the state space. If omitted, defaults to sort(unique(X)).
oscillations	Logical. If TRUE, also compute oscillations for the fitted model (see oscillation).

Details

Regarding the M parameter: it functions as a stopping criterion within the EM algorithm. When the difference between the log-likelihood computed with the newly estimated parameters and that computed with the previous parameters falls below M, the algorithm halts. Nevertheless, if the value of nIter (which represents the maximum number of iterations) is smaller than the number of iterations required to meet the M criterion, the algorithm will conclude its execution when nIter is reached. To ensure that the M criterion is effectively utilized, we recommend using a higher value for nIter, which is set to a default of 100.

Concerning the init parameter, it is expected to be a list with 2 or 3 entries. These entries consist of: an optional vector named p0, representing an independent distribution (the probability in the first entry of p0 must be that of the smallest element in A and so on), a required list of matrices pj, containing a stochastic matrix for each element of S (the first matrix corresponds to the smallest lag in S and so on), and a vector named lambdas representing the weights, the first entry must be the weight for p0, and then one entry for each element in pj list. If your MTD model does not have an independent distribution p0, set init$lambdas[1]=0.

Value

An S3 object of class c("MTDest", "MTD") (a list) with at least the following elements:

- lambdas: estimated mixture weights (independent part first, if any).
- pj: list of estimated transition matrices p_j.
- p0: estimated independent distribution (if applicable).
- logLik: log-likelihood of the final fitted model.
- iterations, deltaLogLik, lastComputedDelta (optional): returned if iter = TRUE. Here, iterations is the number of parameter updates performed; deltaLogLik stores the successive absolute log-likelihood changes for the accepted updates; and lastComputedDelta is the last computed change (which may be below M when the loop stops by convergence).
- oscillations (optional): returned if oscillations = TRUE.
- call: the matched call.
- S: the lag set used for estimation.
- A: the state space used for estimation.
- n: the sample size.
- n_eff: the effective sample size used for estimation.

Methods (S3)

Objects returned by MTDest() have class c("MTDest","MTD") and support:

- print and summary: methods for fitted objects (see MTDest-methods).
- coef: extracts fitted parameters lambdas, pj, and p0 (inherited from "MTD"; see MTD-methods).

- logLik: returns the final log-likelihood stored in the fit (see MTDest-methods).
- plot: diagnostic plots for fitted objects (see plot.MTDest).
- probs, oscillation, and perfectSample: additional utilities available by inheritance from "MTD".
- as.MTD: coerces an "MTDest" fit to an "MTD" model.

Accessors

Stable access to fitted components is provided by MTD-accessors, including S (or Lambda), lags, lambdas, pj, p0, states, and transitP.

References

Lebre, Sophie and Bourguignon, Pierre-Yves. (2008). An EM algorithm for estimation in the Mixture Transition Distribution model. *Journal of Statistical Computation and Simulation*, *78*(1), 1-15. doi:10.1080/00949650701266666

See Also

MTDmodel for constructing an MTD model, hdMTD for lag selection procedures, MTDest-methods for methods specific to fitted objects, MTD-methods for methods inherited from "MTD", MTD-accessors for stable access to components, and as.MTD for coercion of fits to model objects.

Examples

```
# Simulating data.
set.seed(1)
MTD <- MTDmodel(Lambda = c(1, 10), A = c(0, 1), lam0 = 0.01)
X <- perfectSample(MTD, N = 400)

# Initial Parameters:
init <- list(
  'p0' = c(0.4, 0.6),
  'lambdas' = c(0.05, 0.45, 0.5),
  'pj' = list(
      matrix(c(0.2, 0.8, 0.45, 0.55), byrow = TRUE, ncol = 2),
      matrix(c(0.25, 0.75, 0.3, 0.7), byrow = TRUE, ncol = 2)
    )
 )

fit <- MTDest(X, S = c(1, 10), init = init, iter = TRUE)
str(fit, max.level = 1)
fit$logLik
class(fit)

fit2 <- MTDest(X, S = c(1, 10), init = init, oscillations = TRUE)
fit2$oscillations
```

MTDest-methods *Methods for objects of class* "MTDest"

Description

Methods for objects returned by MTDest() – EM fits of Mixture Transition Distribution models. Note that "MTDest" objects inherit from class "MTD" (they have class c("MTDest", "MTD")), and several methods for "MTD" also work on "MTDest" objects by inheritance. The methods documented here are specific to EM fits, providing diagnostics and summaries of the estimation.

Arguments

x	An object of class "MTDest" (for print.MTDest(x, ...)).
object	An object of class "MTDest".
...	Further arguments passed to or from other methods (ignored).

Details

These methods handle objects returned by MTDest (class c("MTDest","MTD")):

- print.MTDest() displays a compact summary of the fitted model: the lag set (S), the state space (A), the final log-likelihood, and, if available, the number of EM updates performed.
- summary.MTDest() computes and prints a detailed summary of the key components of the object, including lambdas, transition matrices, independent distribution (if present), log-likelihood, and (if available) oscillations and iteration diagnostics.
- logLik.MTDest() returns the log-likelihood as an object of class "logLik", with attributes df (number of free parameters under the multimatrix model) and nobs (effective sample size).

Value

print.MTDest Invisibly returns the "MTDest" object, after displaying its lag set, state space, final log-likelihood, and iteration count (if available).

summary.MTDest Invisibly returns a named list with fields: call, S, A, lambdas, pj, p0 (or NULL),logLik, oscillations, iterations, lastComputedDelta and deltaLogLik. The same information is printed to the console in a readable format.

logLik.MTDest An object of class "logLik" with attributes df (number of free parameters) and nobs (effective sample size).

See Also

MTDest, as.MTD, MTD-methods, oscillation, perfectSample, probs

Examples

```
## Not run:
set.seed(1)
MTD <- MTDmodel(Lambda = c(1, 3), A = c(0, 1), lam0 = 0.01)
X <- perfectSample(MTD, N = 200)  # small N to keep examples fast
init <- list(
  p0 = c(0.4, 0.6),
  lambdas = c(0.05, 0.45, 0.5),
  pj = list(
    matrix(c(0.2, 0.8, 0.45, 0.55), byrow = TRUE, ncol = 2),
    matrix(c(0.25, 0.75, 0.3, 0.7),  byrow = TRUE, ncol = 2)
  )
)
fit <- MTDest(X, S = c(1, 3), init = init, iter = TRUE)
print(fit)
summary(fit)
coef(fit) # Works by inheritance
logLik(fit)
BIC(fit)

## End(Not run)
```

`MTDmodel` *Creates a Mixture Transition Distribution (MTD) Model*

Description

Generates an MTD model as an object of class `MTD` given a set of parameters.

Usage

```
MTDmodel(
  Lambda,
  A,
  lam0 = NULL,
  lamj = NULL,
  pj = NULL,
  p0 = NULL,
  single_matrix = FALSE,
  indep_part = TRUE
)
```

Arguments

`Lambda`	A numeric vector of positive integers representing the relevant lag set. The elements will be sorted from smallest to greatest. The smallest number represents the latest (most recent) time in the past, and the largest number represents the earliest time in the past.

A	A vector with nonnegative integers representing the state space.
lam0	A numeric value in [0,1), representing the weight of the independent distribution.
lamj	A numeric vector of weights for the transition probability matrices in pj. Values must be in the range [0, 1), and their sum with lam0 must be equal to 1. The first element in lamj must be the weight for the first element in Lambda and so on.
pj	A list with length(Lambda) stochastic matrices, each of size length(A) x length(A). The first matrix in pj must refer to the first element in Lambda and so on.
p0	A probability vector for the independent component of the MTD model. If NULL and indep_part=TRUE, the distribution will be sampled from a uniform distribution. If indep_part=FALSE, then there is no independent distribution and p0 entries will be set to zero. If you enter p0=0, indep_part is set to FALSE.
single_matrix	Logical. If TRUE, all matrices in list pj are identical.
indep_part	Logical. If FALSE, the model does not include an independent distribution and p0 is set to zero.

Value

A list of class MTD containing:

P The transition probability matrix of the MTD model.

lambdas A vector with MTD weights (lam0 and lamj).

pj A list of stochastic matrices defining conditional transition probabilities.

p0 The independent probability distribution.

Lambda The vector of relevant lags.

A The state space.

single_matrix A logical argument, if TRUE indicates that all matrices in pj are identical.

Methods (S3)

Objects returned by MTDmodel() have class "MTD" and support:

- print: compact display of the relevant lag set and the state space (see MTD-methods).
- summary: detailed summary of the model components (see MTD-methods).
- coef: extracts the model parameters lambdas, pj, and p0 (see MTD-methods).
- logLik: logLik(object, X) computes the log-likelihood, provided that the user supplies a sample X (see MTD-methods).
- plot: diagnostic plots, including oscillations by lag, mixture weights, and transition graphs (see plot.MTD).
- oscillation: oscillations by lag computed from the model parameters.
- perfectSample: perfect sampling from the stationary distribution, provided that $\lambda_0 > 0$.
- probs: one-step predictive probabilities.

Accessors

Stable access to model components is provided by MTD-accessors: lags, Lambda, lambdas, pj, p0, states, and transitP.

See Also

MTDest for EM-based parameter estimation, hdMTD for lag selection procedures, MTD-methods for methods applicable to "MTD" objects, MTD-accessors for stable access to model components, oscillation, perfectSample, and probs for additional inference and simulation utilities.

Examples

```
summary(MTDmodel(Lambda = c(1, 3), A = c(4, 8, 12)))

MM <- MTDmodel(Lambda = c(2, 4, 9), A = c(0, 1), lam0 = 0.05,
 lamj = c(0.35, 0.2, 0.4), pj = list(matrix(c(0.5, 0.7, 0.5, 0.3), ncol = 2)),
 p0 = c(0.2, 0.8), single_matrix = TRUE)
transitP(MM); pj(MM); oscillation(MM)


MM <- MTDmodel(Lambda = c(2, 4, 9), A = c(0, 1), lam0 = 0.05,
 pj = list(matrix(c(0.5, 0.7, 0.5, 0.3), ncol = 2)), single_matrix = TRUE,
 indep_part = FALSE)
p0(MM); lambdas(MM)
```

---

oscillation *Oscillations of an MTD Markov chain*

Description

Calculates the oscillations of an MTD model object, of an MTDest fit, or estimates the oscillations of a chain sample.

Usage

```
oscillation(x, ...)

## S3 method for class 'MTD'
oscillation(x, ...)

## Default S3 method:
oscillation(x, S, A = NULL, ...)
```

Arguments

x	Either an MTD model object, an MTDest fit, or a chain sample.
...	Ignored.
S	A numeric vector of distinct positive integers. Represents a set of lags.
A	Optional integer vector giving the state space. If omitted, defaults to `sort(unique(x))`.

Details

For an MTD model or an MTDest fit, the oscillation for lag j (i.e., $\{\delta_j : j \in \Lambda\}$ for MTD or $\{\delta_j : j \in S\}$ for MTDest) is the product of the mixture weight λ_j and the maximum of the total variation distance between the distributions in a stochastic matrix p_j. That is,

$$\delta_j = \lambda_j \max_{b,c \in \mathcal{A}} d_{TV}(p_j(\cdot|b), p_j(\cdot|c)).$$

When x is an object of class MTD (or MTDest), the quantities Λ (or S), $\mathcal{A}$, λ_j, and p_j are directly available. In this case, the oscillations can be computed exactly from the model parameters.

Alternatively, when estimating oscillations from data, x must be a realization of a chain and S must be provided as a candidate lag set. The transition probabilities are then estimated from the sample before computing the corresponding oscillations. Let $\hat{p}(\cdot|x_S)$ symbolize an estimated distribution in $\mathcal{A}$ given a certain past x_S (which is a sequence of elements of $\mathcal{A}$ where each element occurred at a lag in S), and $\hat{p}(\cdot|b_j x_S)$ an estimated distribution given past x_S and that the symbol $b \in \mathcal{A}$ occurred at lag j. If N is the sample size, $d =$`max(S)` and $N(x_S)$ is the number of times the sequence x_S appeared in the sample, then

$$\delta_j = \max_{c_j, b_j \in \mathcal{A}} \frac{1}{N-d} \sum_{x_S \in \mathcal{A}^S} N(x_S) d_{TV}(\hat{p}(.|b_j x_S), \hat{p}(.|c_j x_S))$$

is the estimated oscillation for a lag $j \in \{1, \ldots, d\} \backslash$ S. Note that $\mathcal{A}^S$ is the space of sequences of $\mathcal{A}$ indexed by S.

Value

A named numeric vector of oscillations. If the x parameter is an MTD or MTDest object, it will provide the oscillations for each element in Lambda (or S for MTDest). If x is a chain sample, it estimates the oscillations for a user-inputted set of lags S.

Methods (by class)

- `oscillation(MTD)`: For an MTD (or MTDest) object: computes δ_j for all $j \in \Lambda$ (or for all $j \in S$).
- `oscillation(default)`: For a chain sample: estimates δ_j for each j in S.

Examples

```
oscillation(MTDmodel(Lambda = c(1, 4), A = c(2, 3)))
oscillation(MTDmodel(Lambda = c(1, 4), A = c(2, 3), lam0 = 0.01, lamj = c(0.49, 0.5),
                     pj = list(matrix(c(0.1, 0.9, 0.9, 0.1), ncol = 2)),
                     single_matrix = TRUE))
```

perfectSample *Perfectly samples an MTD Markov chain*

Description

Samples an MTD Markov Chain from the stationary distribution.

Usage

```
perfectSample(object, N, ...)
```

Arguments

object	An object of class "MTD" or "MTDest".
N	Positive integer. Sample size to generate. Must be > max(Lambda(object)).
...	Additional arguments passed to methods.

Details

This perfect sample algorithm requires that the MTD model has an independent distribution (p0) with a positive weight (i.e., `lambdas(object)["lam0"]>0` which means $\lambda_0 > 0$).

Value

Returns a size N sample from an MTD model (the first element is the most recent).

Examples

```
M <- MTDmodel(Lambda = c(1, 3, 4), A = c(0, 2))
perfectSample(M, N = 200)

M <- MTDmodel(Lambda = c(2, 5), A = c(1, 2, 3))
perfectSample(M, N = 300)
```

plot.MTD *Plot method for MTD objects*

Description

Produces plots for an `MTD` object. By default, it shows the following sequence of plots: (i) barplot of oscillations by relevant lag, (ii) barplot of mixture weights λ_j (including `lam0` if `> 0`) and (iii) graphs of `pj`, one graph for each lag in `Lambda`. When `type` is specified, only the requested plot is drawn.

Usage

```
## S3 method for class 'MTD'
plot(x, type, main, ylim, col = "gray70", border = NA, pj_index = 1, ...)
```

Arguments

x	An object of class "MTD".
type	If type is missing, all plots are shown sequentially (press Enter to proceed). Else, type is a character string indicating what to plot: "oscillation", "lambdas" or "pj".
main	Optional main title. When type is missing, panel-specific defaults are used and main is ignored.
ylim	Optional y-axis limits. When type is missing, panel-specific defaults are used and ylim is ignored.
col	Bar fill color (passed to barplot). Defaults to "gray70".
border	Bar border color (passed to barplot). Defaults to NA.
pj_index	Integer index of pj(x) to plot when type = "pj" (default 1).
...	Further graphical parameters: passed to barplot() when type %in% c("oscillation","lambdas") and to plot.igraph() when type = "pj". Ignored when type is missing.

Details

For type = "oscillation", the function calls oscillation(x) to obtain $\delta_j = \lambda_j \max_{b,c} d_{TV}(p_j(\cdot|b), p_j(\cdot|c))$ for each lag in Lambda(x), and draws a bar plot named by the lags.

For type = "lambdas", it plots the mixture weights λ_j by lag. If lam0 > 0, the weight for the independent component is included and labeled "0".

For type = "pj", the function draws the directed, weighted graph of a transition matrix pj taken from pj(x). Vertices correspond to the states states(x). A directed edge a -> b carries weight p_j(b | a). Edge widths and edge labels are proportional to the transition probabilities (labels shown in the plot are rounded to two decimals). By default, self-loops (a -> a) are not drawn. The self-loop probability at a state a can be inferred as

$$1 - \sum_{b:b \neq a} p_j(b \mid a).$$

For type = "pj", a specific matrix can be selected via pj_index (e.g., pj_index = 2 plots pj(x)[[2]]). In automatic mode (when type is missing), graphs for all pj matrices are shown sequentially.

Value

If type is "oscillation" or "lambdas", invisibly returns the numeric vector that was plotted. If type = "pj", invisibly returns the selected transition matrix. If type is missing, invisibly returns a list with components oscillation and lambdas.

See Also

oscillation, lambdas, Lambda

Examples

```
## Not run:
m <- MTDmodel(Lambda = c(1, 3), A = c(0, 1))

## Automatic mode (press Enter between plots)
plot(m)

## Single plot:
plot(m, type = "oscillation")
plot(m, type = "lambdas")
plot(m, type = "pj", pj_index = 2)

## End(Not run)
```

`plot.MTDest` *Plot method for MTDest objects*

Description

Produces plots for an `MTDest` object. By default, it shows in sequence: (i) barplot of oscillations by relevant lag, (ii) barplot of mixture weights λ_j (including `lam0` if `> 0`), and (iii) graphs of `pj` (one graph for each lag in `Lambda`). If EM diagnostics are available, a convergence panel (log-likelihood variation per update) is shown last. When `type` is specified, only the requested plot is drawn.

Usage

```
## S3 method for class 'MTDest'
plot(x, type, main, ylim, col = "gray70", border = NA, pj_index = 1, ...)
```

Arguments

`x`	An object of class `"MTDest"`.
`type`	If `type` is missing, `oscillation`, `lambdas`, `pj` graphs and—if available—the `convergence` plots are shown sequentially (press Enter to proceed). Else, `type` is a character string indicating what to plot: `"oscillation"`, `"lambdas"`, `"pj"` or `"convergence"`.
`main`	Optional main title. When `type` is missing, panel-specific defaults are used and `main` is ignored.
`ylim`	Optional y-axis limits. When `type` is missing, panel-specific defaults are used and `ylim` is ignored.
`col`	Bar fill color (passed to `barplot`). Defaults to `"gray70"`.
`border`	Bar border color (for bar plots). Defaults to `NA`.
`pj_index`	Integer index of `pj(x)` to plot when `type = "pj"` (default `1`).
`...`	Further graphical parameters: passed to `barplot()` when `type %in% c("oscillation","lambdas")`, to `plot.igraph()` when `type = "pj"`, and to `plot()` when `type = "convergence"`. Ignored when `type` is missing.

Details

For `type = "oscillation"`, the function delegates to `plot.MTD()`, which returns $\delta_j = \lambda_j \max_{b,c} d_{TV}(p_j(\cdot|b), p_j(\cdot|c))$ for each lag in `lags(x)`, and draws a bar plot named by the lags.

For `type = "lambdas"`, the function delegates to `plot.MTD()`, it plots the mixture weights λ_j by lag. If `lam0 > 0`, the weight for the independent component is included and labeled `"0"`.

For `type = "pj"`, the function delegates to `plot.MTD()`, it draws the directed, weighted graph of the transition matrices `pj` taken from `pj(x)`. Vertices correspond to the states `states(x)`. A directed edge `a -> b` carries weight `p_j(b | a)`. Edge widths and edge labels are proportional to the transition probabilities (labels shown in the plot are rounded to two decimals). By default, self-loops (`a -> a`) are not drawn. The self-loop probability at a state `a` can be inferred as

$$1 - \sum_{b:b \neq a} p_j(b \mid a).$$

For `type = "pj"`, a specific matrix can be selected via `pj_index` (e.g., `pj_index = 2` plots `pj(x)[[2]]`). In automatic mode (when `type` is missing), graphs for all `pj` matrices are shown sequentially.

If EM iteration diagnostics are available (i.e., the object was fitted with `iter = TRUE` and `length(deltaLogLik) > 0`), a convergence panel showing the log-likelihood variation per update is displayed automatically when `type` is missing. You can also request it explicitly with `type = "convergence"`.

Value

If `type` is `"oscillation"` or `"lambdas"`, invisibly returns the numeric vector that was plotted. If `type = "pj"`, invisibly returns the selected transition matrix `pj(x)[[pj_index]]`. If `type = "convergence"`, invisibly returns the numeric vector `deltaLogLik`. If `type` is missing, invisibly returns the list produced by `plot.MTD(m)` (typically with components `oscillation` and `lambdas`); if `deltaLogLik` is available, the returned list also includes `deltaLogLik`.

See Also

`plot.MTD`, `as.MTD`, `MTDest`

Examples

```
## Not run:
set.seed(1)
M <- MTDmodel(Lambda = c(1, 3), A = c(0, 1), lam0 = 0.05)
X <- perfectSample(M, N = 300)
fit <- MTDest(X, S = c(1, 3), init = coef(M), iter = TRUE)

plot(fit)
plot(fit, type = "pj", pj_index = 2)
plot(fit, type = "convergence")

## End(Not run)
```

probs *Predictive probabilities for MTD / MTDest*

Description

Compute one-step-ahead predictive probabilities under an MTD model or an MTDest fit. For objects of class "MTDest", the method is inherited from the "MTD" class via S3 inheritance.

Conventions:

- Samples are read most recent first: `x[1] = X_{t-1}`, `x[2] = X_{t-2}`, etc.
- The global transition matrix `P` (or transitP) is indexed by row labels that list the past context from oldest to newest. A cell at row `"s_k...s_1"` and column `"a"` is read as `p(a | s_1...s_k)`.
- If both `newdata` and `context` are missing, `probs()` returns the full global transition matrix (`transitP(object)`).

Usage

```
probs(object, context = NULL, newdata = NULL, oldLeft = FALSE)
```

Arguments

`object`	An `MTD` or `MTDest` object.
`context`	Optional vector or matrix/data.frame of contexts (rows). By default, each row follows the "most recent first" convention; set `oldLeft = TRUE` if rows are supplied oldest to newest. Must have exactly `length(Lambda(object))` columns (one symbol per lag).
`newdata`	Optional vector or matrix/data.frame of samples (rows). Columns follow the "most recent first" convention. Must have at least `max(Lambda(object))` columns. Only one of `newdata` or `context` can be provided at a time. When there are extra columns, only the columns at the model lags are used.
`oldLeft`	Logical. If `TRUE`, interpret rows in `newdata`/`context` as oldest to newest (e.g. leftmost = `newdata[ ,1]` = oldest). If `FALSE` (default), rows are most recent first.

Details

All entries of `newdata`/`context` must belong to the model's state space `states(object)`.

Value

A numeric matrix of predictive probabilities with one row per input context and columns indexed by `states(object)`. Row names are the context labels (oldest to newest) formed by concatenating state symbols without a separator. If both `newdata` and `context` are missing, the full global transition matrix is returned, which can be large for big state spaces or many lags.

See Also

transitP, states, Lambda, empirical_probs, MTDmodel, MTDest

Examples

```
set.seed(1)
m <- MTDmodel(Lambda = c(1,3), A = c(0,1), lam0 = 0.1)

# Full matrix
P <- probs(m)

# Using a sample row (most recent first): newdata has >= max(Lambda) columns
new_ctx <- c(1, 0, 1, 0)      # X_{t-1}=1, X_{t-2}=0, X_{t-3}=1, ...
probs(m, newdata = new_ctx)   # one row of probabilities

# Explicit contexts (exactly |Lambda| symbols per row)
probs(m, context = c(0, 1), oldLeft = FALSE)  # most recent first
probs(m, context = c(0, 1), oldLeft = TRUE)   # oldest to newest

# Multiple contexts (rows)
ctxs <- rbind(c(1,0,1), c(0,1,1), c(1,1,0))
probs(m, newdata = ctxs)
```

---

tempdata *Maximum temperatures in the city of Brasília, Brazil.*

Description

A data frame with the maximum temperature of the last hour, by each hour, in the city of Brasília, Brazil. The data spans from 01/01/2003 to 31/08/2024.

Usage

```
tempdata
```

Format

A data frame with 189936 rows and 3 columns. Each row corresponds to a time in a day specified in columns 2 ("TIME") and 1 ("DATE") respectively. The value in column 3 ("MAXTEMP") is the maximum temperature measured in the last hour, in Celsius (Cº), in the city of Brasília, the capital of Brazil, located in the central-western part of the country.

DATE The day, from 01/01/2003 to 31/08/2024

TIME The time, form 00:00 to 23:00 each day

MAXTEMP The maximum temperature measured in the last hour in Celsius

Source

Meteorological data provided by INMET (National Institute of Meteorology, Brazil). Data collected from automatic weather station in Brasília (latitude: -15.79°, longitude: -47.93°, altitude: 1159.54 m). Available at: https://bdmep.inmet.gov.br/

Examples

```
data(tempdata)
```

Index

∗ **datasets**
    tempdata, 35

as.MTD, 2, *24, 25*, *33*

coef, *23*, *27*
countsTab, 3
countsTab(), *7*

dTV_sample, 4

empirical_probs, 6, *13*, *35*

freqTab, *5*, 7, *13*
freqTab(), *5*, *7*

hdMTD, 8, *10, 11*, *20*, *24*, *28*
hdMTD(), *13*, *15, 16*, *18*
hdMTD-methods, 10
hdMTD_BIC, *10, 11*, 12
hdMTD_BIC(), *9*
hdMTD_CUT, *10, 11*, 14
hdMTD_CUT(), *9*
hdMTD_FS, *10, 11*, 16
hdMTD_FS(), *9*
hdMTD_FSC, *10, 11*, 17
hdMTD_FSC(), *9*

lags, *10, 11*, *21*, *24*, *28*
lags *(*MTD-accessors*)*, 19
Lambda, *21*, *24*, *28*, *31*, *35*
Lambda *(*MTD-accessors*)*, 19
lambdas, *21*, *24*, *28*, *31*
lambdas *(*MTD-accessors*)*, 19
logLik, *21*, *24*, *27*

MTD-accessors, 19
MTD-methods, 20
MTDest, *3*, *20, 21*, 22, *25*, *28*, *33*, *35*
MTDest-methods, 25
MTDmodel, *2, 3*, *20, 21*, *24*, 26, *35*

oscillation, *21*, *23–25*, *27*, 28, *28*, *31*

p0, *21*, *24*, *28*
p0 *(*MTD-accessors*)*, 19
perfectSample, *21*, *24, 25*, *27, 28*, 30
pj, *21*, *24*, *28*
pj *(*MTD-accessors*)*, 19
plot, *24*, *27*
plot.MTD, *27*, 30, *33*
plot.MTDest, *24*, 32
print, *10*, *23*, *27*
probs, *21*, *24, 25*, *27, 28*, 34

S, *10, 11*, *24*
S *(*MTD-accessors*)*, 19
states, *21*, *24*, *28*, *35*
states *(*MTD-accessors*)*, 19
summary, *10*, *23*, *27*

tempdata, 35
transitP, *21*, *24*, *28*, *35*
transitP *(*MTD-accessors*)*, 19