Package 'localScore'

April 24, 2025

Type Package

Title Package for Sequence Analysis by Local Score

Version 2.0.3

Date 2025-04-24

Copyright See the file COPYRIGHTS for various embedded Eigen library copyright details

Maintainer David Robelin <david.robelin@inrae.fr>

Description Functionalities for calculating the local score and calculating statistical relevance (p-value) to find a local Score in a sequence of given distribution (S. Mercier and J.-J. Daudin (2001) <https://hal.science/hal-00714174/>) ; S. Karlin and S. Altschul (1990) <https://pmc.ncbi.nlm.nih.gov/articles/PMC53667/> ; S. Mercier, D. Cellier and F. Charlot (2003) <https://hal.science/hal-00937529v1/> ; A. Lagnoux, S. Mercier and P. Valois (2017) <doi:10.1093/bioinformatics/btw699>).

License GPL (>= 2) | file LICENSE

Imports Rcpp (>= 0.12.16), utils

LinkingTo Rcpp

RoxygenNote 7.3.2

LazyData true

Encoding UTF-8

Suggests knitr, rmarkdown, testthat (>= 2.1.0)

VignetteBuilder knitr

Depends R (>= 3.2.0)

NeedsCompilation yes

Author Sebastian Simon [aut],
Chris Verschelden [aut],
Charly Marty [aut],
David Robelin [aut, cre],
Sabine Mercier [aut],
Sebastien Dejean [aut],
The authors of Eigen the library for the included version of Eigen [cph]

Repository CRAN

Date/Publication 2025-04-24 10:50:02 UTC

Contents

`Aeso` *Congenital oesophageal atresia data*

Description

The data consists of individual dates of birth over n=35 cases of the birth defects oesophageal and tracheo-oesophagean fistula observed in a hospital in Birmingham, U.K., over 2191 days from 1950 through 1955, with Day one set as 1 January 1950

Usage

```
data(Aeso)
```

Format

A matrix of 2191 lines and 2 columns. Each line is a day on the first column, and associated to a case (0/1) on the second column.

Source

Dolk H. Secular pattern of congenital oesophageal atresia–George Knox, 1959. J Epidemiol Community Health. 1997;51(2):114-115. <doi:10.1136/jech.51.2.114>

Examples

```
data(Aeso)
head(Aeso)
p <- sum(Aeso[,2]) / dim(Aeso)[1]
print(p)
```

---

automatic_analysis *Automatic analysis*

Description

Calculates local score and p-value for sequence(s) with integer scores.

Usage

```
automatic_analysis(
  sequences,
  model,
  scores,
  transition_matrix,
  distribution,
  method_limit = 2000,
  score_extremes,
  modelFunc,
  simulated_sequence_length = 1000,
  ...
)
```

Arguments

sequences	sequences to be analyzed (named list)
model	the underlying model of the sequence (either "iid" for identically independently distributed variable or "markov" for Markov chains)
scores	vector of minimum and maximum score range
transition_matrix	if the sequences are markov chains, this is their transition matrix
distribution	vector of probabilities in ascending score order (iid sequences). Note that names of the vector must be the associated scores.
method_limit	limit length from which on computation-intensive exact calculation methods for p-value are replaced by approximative methods
score_extremes	a vector with two elements: minimal score value, maximal score value
modelFunc	function to create similar sequences. In this case, Monte Carlo is used to calculate p-value
simulated_sequence_length	if a modelFunc is provided and the sequence happens to be longer than method_limit, the method karlinMonteCarlo is used. This method requires the length of the sequences that will be created by the modelFunc for estimation of Gumble parameters.
...	parameters for modelFunc

Details

This method picks the adequate p-value method for your input.
If no sequences are passed to this function, it will let you pick a FASTA file.
If this is the case, and if you haven't provided any score system (as you can do by passing a named list with the appropriate scores for each character), the second file dialog which will pop up is for choosing a file containing the score (and if you provide an extra column for the probabilities, they will be used, too - see section File Formats in the vignette for details).
The function then either uses empirical distribution based on your input - or if you provided a distribution, then yours - to calculate the p-value based on the length of each of the sequences given as input.
You can influence the choice of the method by providing the modelFunc argument. In this case, the function uses exclusively simulation methods (monteCarlo, karlinMonteCarlo).
By setting the method_limit you can further decide to which extent computation-intensive methods (daudin, exact_mc) should be used to calculate the p-value. Remark that the warnings of the localScoreC() function have be deleted when called by automatic_analysis() function

Value

A list object containing

Local score	local score...
p-value	p-value ...
Method	the method used for the calculus of the p-value

Examples

```
# Minimal example
l = list()
seq1 = sample(-2:1, size = 3000, replace = TRUE)
seq2 = sample(-3:1, size = 150, replace = TRUE)
l["hello"]] = seq1
l["world"]] = seq2
automatic_analysis(l, "iid")
# Example with a given distribution
automatic_analysis(l,"iid",scores=-3:1,distribution=c(0.3,0.3,0.1,0.1,0.2))
#Equivalent call
automatic_analysis(l,"iid",score_extremes=c(-3,1),distribution=c(0.3,0.3,0.1,0.1,0.2))
# forcing the exact method for the longest sequence
aa1=automatic_analysis(l,"iid")
aa1$hello$`method applied`
aa1$hello$`p-value`
aa2=automatic_analysis(l,"iid",method_limit=3000)
aa2$hello$`method applied`
aa2$hello$`p-value`
# Markovian example
MyTransMat <-
matrix(c(0.3,0.1,0.1,0.1,0.4, 0.3,0.2,0.2,0.2,0.1, 0.3,0.4,0.1,0.1,0.1, 0.3,0.3,0.3,0.0,0.1,
        0.1,0.3,0.2,0.3,0.1), ncol = 5, byrow=TRUE)
MySeq.CM=transmatrix2sequence(matrix = MyTransMat,length=150, score =-2:2)
MySeq.CM2=transmatrix2sequence(matrix = MyTransMat,length=110, score =-2:2)
automatic_analysis(sequences = list("x1" = MySeq.CM, "x2" = MySeq.CM2), model = "markov")
```

CharSequence2ScoreSequence

Convert a character sequence into a score sequence

Description

Convert a character sequence into a score sequence. See CharSequences2ScoreSequences() function for several sequences

Usage

```
CharSequence2ScoreSequence(charseq, dictionary)
```

Arguments

charseq	a character sequence, given as a string
dictionary	a data.frame with rownames containing letters, first column containing associated scores, optional second column containing associated probabilities

Value

a vector of a score sequence

See Also

CharSequences2ScoreSequences

Examples

```
data(Seq31)
Seq31
data(HydroScore)
CharSequence2ScoreSequence(Seq31,HydroScore)
```

CharSequences2ScoreSequences

Convert several character sequences into score sequences

Description

Convert several character sequence into score sequences. For only one sequence see CharSequence2ScoreSequence() function.

Usage

```
CharSequences2ScoreSequences(sequences, dictionary)
```

Arguments

sequences	a list of character sequences given as string
dictionary	a data.frame with rownames containing letters, first column containing associated scores, optional second column containing associated probabilities

Value

a list of score sequences

See Also

CharSequence2ScoreSequence

Examples

```
data(Seq31)
Seq31
data(Seq219)
Seq219
data(HydroScore)
MySequences=list("A1"=Seq31,"A2"=Seq219)
CharSequences2ScoreSequences(MySequences,HydroScore)
```

daudin *Daudin [p-value] [iid]*

Description

Calculates the exact p-value in the identically and independently distributed of a given local score, a sequence length that 'must not be too large' and for a given score distribution

Usage

```
daudin(
  local_score,
  sequence_length,
  score_probabilities,
  sequence_min = NULL,
  sequence_max = NULL,
  score_values = NULL
)
```

Arguments

`local_score`	the observed local score
`sequence_length`	length of the sequence
`score_probabilities`	the probabilities for each score from lowest to greatest (Optionnaly with scores as names)
`sequence_min`	minimum score (optional if `score_values` OR names(score_probabilities) is defined)
`sequence_max`	maximum score (optional if `score_values` OR names(score_probabilities) is defined)
`score_values`	vector of integer score values, associated to score_probabilities (optional if `sequence_min` and `sequence_max` OR names(score_probabilities) are defined)

Details

Either `sequence_min` and `sequence_max` are specified as input, OR all possible score values in `score_values` vector ; one of this choice is required. <cr> Small in this context depends heavily on your machine. On a 3,7GHZ machine this means for daudin(1000, 5000, c(0.2, 0.2, 0.2, 0.1, 0.2, 0.1), -2, 3) an execution time of ~2 seconds. This is due to the calculation method using matrix exponentiation which takes times. The size of the matrix of the exponentiation is equal to a+1 with a the local score value. The matrix must be put at the power n, with n the sequence length. Moreover, it is known that the local score value is expected to be in mean of order log(n).

Value

A double representing the probability of a local score as high as the one given as argument.

Examples

```
p1 <- daudin(local_score = 4, sequence_length = 50,
       score_probabilities = c(0.2, 0.3, 0.1, 0.2, 0.1, 0.1),
       sequence_min = -3, sequence_max = 2)
p2 <- daudin(local_score = 4, sequence_length = 50,
       score_probabilities = c(0.2, 0.3, 0.1, 0.2, 0.1, 0.1),
       score_values = as.integer(-3:2))
p1 == p2 # TRUE

prob <- c(0.08, 0.32, 0.08, 0.00, 0.08, 0.00, 0.00, 0.08, 0.02, 0.32, 0.02)
score_values <- which(prob != 0) - 6 # keep only non null probability scores
prob0 <- prob[prob != 0]             # and associated probability
p <- daudin(150, 10000, prob, sequence_min = -5, sequence_max =  5)
p0 <- daudin(150, 10000, prob0, score_values = score_values)
names(prob0) <- score_values
p1 <- daudin(150, 10000, prob0)
p == p0 # TRUE
p == p1 # TRUE
```

---

exact_mc *Exact method for p-value [Markov chains]*

Description

Calculates the exact p-value for short numerical Markov chains. Memory usage and time computation can be too large for a high local score value and high score range (see details).

Usage

```
exact_mc(local_score, m, sequence_length, score_values = NULL, prob0 = NULL)
```

Arguments

local_score	Integer local score for which the p-value should be calculated
m	Transition matrix [matrix object]. Optionality, rownames can be corresponding score values. m should be a transition matrix of an ergodic Markov chain.
sequence_length	Length of the sequence
score_values	A integer vector of sequence score values (optional). If not set, the rownames of m are used if they are numeric and set.
prob0	Vector of probability distribution of the first score of the sequence (optional). If not set, the stationnary distribution of m is used.

Details

This method computation needs to allocate a square matrix of size local_score^(range(score_values)). This matrix is then exponentiated to sequence_length.

Value

A double representing the probability of a local score as high as the one given as argument

See Also

monteCarlo

Examples

```
mTransition <- matrix(c(0.2, 0.3, 0.5, 0.3, 0.4, 0.3, 0.2, 0.4, 0.4), byrow = TRUE, ncol = 3)
scoreValues <- -1:1
initialProb <- stationary_distribution(mTransition)
exact_mc(local_score = 12, m = mTransition, sequence_length = 100,
        score_values = scoreValues, prob0 = initialProb)
exact_mc(local_score = 150, m = mTransition, sequence_length = 1000,
         score_values = scoreValues, prob0 = initialProb)
rownames(mTransition) <- scoreValues
exact_mc(local_score = 12, m = mTransition, sequence_length = 100, prob0 = initialProb)
# Minimal specification
exact_mc(local_score = 12, m = mTransition, sequence_length = 100)
# Score valeus with "holes"
scoreValues <- c(-2, -1, 2)
mTransition <- matrix(c(0.2, 0.3, 0.5, 0.3, 0.4, 0.3, 0.2, 0.4, 0.4), byrow = TRUE, ncol = 3)
initialProb <- stationary_distribution(mTransition)
exact_mc(local_score = 50, m = mTransition, sequence_length = 100,
        score_values = scoreValues, prob0 = initialProb)
```

HydroScore *Dictionary*

Description

Provides integer scores related to an hydrophobicity level of each amino acid. This score function is inspired by the Kyte and Doolittle (1982) scale.

Usage

```
data(HydroScore)
```

Format

A score function for the 20 amino acid

Source

Kyte & Doolittle (1982) J. Mol. Biol. 157, 105-132

Examples

```
data(HydroScore)
HydroScore
data(Seq219)
Seq219
seqScore <- CharSequence2ScoreSequence(Seq219,HydroScore)
seqScore[1:30]
localScoreC(seqScore)$localScore
```

`karlin` *Karlin [p-value] [iid]*

Description

`karlin` Calculates an approximated p-value of a given local score value and a long sequence length in the identically and independently distributed model for the sequence. See also `mcc` function for another approximated method in the i.i.d. model that improved the one given by `karlin` or `daudin` for exact calculation.
`karlin_parameters` is a annex function returning the parameters λ, K^+ and K^* defined in Karlin and Dembo (1992).

Usage

```
karlin(
  local_score,
  sequence_length,
  score_probabilities,
  sequence_min = NULL,
  sequence_max = NULL,
  score_values = NULL
)

karlin_parameters(
  score_probabilities,
  sequence_min = NULL,
  sequence_max = NULL,
  score_values = NULL
)
```

Arguments

`local_score`	the observed local score
`sequence_length`	length of the sequence
`score_probabilities`	the probabilities for each score from lowest to greatest (Optionnaly with scores as names)

sequence_min	minimum score (optional if score_values OR names(score_probabilities) is defined)
sequence_max	maximum score (optional if score_values OR names(score_probabilities) is defined)
score_values	vector of integer score values, associated to score_probabilities (optional if sequence_min and sequence_max OR names(score_probabilities) are defined)

Details

This method works the better the longer the sequence is. Important note : the calculus of the parameter of the distribution uses the resolution of a polynome which is a function of the score distribution, of order max(score)-min(score). There exists only empirical methods to solve a polynome of order greater that 5 with no warranty of reliable solution. The found roots are checked internally to the function and an error message is throw in case of inconsistent. In such case, you could try to change your score scheme (in case of discretization) or use the function karlinMonteCarlo . This function implements the formulae given in Karlin and Dembo (1992), page 115-6. As the score is discrete here (lattice score function), there is no limit distribution of the local score with the size of the sequence, but an inferior and a superior bound are given. The output of this function is conservative as it gives the upper bound for the p-value. Notice the lower bound can easily be found as it is the same call of function with parameter value local_score+1.

Value

A double representing the probability of a local score as high as the one given as argument

See Also

mcc, daudin, karlinMonteCarlo, monteCarlo

Examples

```
karlin(150, 10000, c(0.08, 0.32, 0.08, 0.00, 0.08, 0.00, 0.00, 0.08, 0.02, 0.32, 0.02), -5, 5)
p1 <- karlin(local_score = 15, sequence_length = 5000,
       score_probabilities = c(0.2, 0.3, 0.1, 0.2, 0.1, 0.1),
       sequence_min = -3, sequence_max = 2)
p2 <- karlin(local_score = 15, sequence_length = 5000,
       score_probabilities = c(0.2, 0.3, 0.1, 0.2, 0.1, 0.1),
       score_values = -3:2)
p1 == p2 # TRUE

prob <- c(0.08, 0.32, 0.08, 0.00, 0.08, 0.00, 0.00, 0.08, 0.02, 0.32, 0.02)
score_values <- which(prob != 0) - 6 # keep only non null probability scores
prob0 <- prob[prob != 0]             # and associated probability
p <- karlin(150, 10000, prob, sequence_min = -5, sequence_max =  5)
p0 <- karlin(150, 10000, prob0, score_values = score_values)
names(prob0) <- score_values
p1 <- karlin(150, 10000, prob0)
p == p0 # TRUE
p == p1 # TRUE
```

karlinMonteCarlo *Monte Carlo - Karlin [p-value]*

Description

Estimates p-value of the local score based on a Monte Carlo estimation of Gumble parameters from simulations of smaller sequences with same distribution. Appropriate for great sequences with length > 10^3, for i.i.d and markovian sequence models.

Usage

```
karlinMonteCarlo(
  local_score,
  sequence_length,
  simulated_sequence_length,
  FUN,
  ...,
  numSim = 1000,
  plot = TRUE,
  keepSimu = FALSE
)

karlinMonteCarlo_double(
  local_score,
  sequence_length,
  simulated_sequence_length,
  FUN,
  ...,
  numSim = 1000,
  plot = TRUE,
  keepSimu = FALSE
)
```

Arguments

local_score	local score observed in a sequence.
sequence_length	length of the sequence
simulated_sequence_length	length of simulated sequences produced by
FUN	function to simulate similar sequences with.
...	parameters for FUN
numSim	number of sequences to create for estimation FUN
plot	boolean value if to display plots for cumulated function, density and linearization of cumulative density function
keepSimu	Boolean, default to FALSE. If TRUE, the simulated local scores are returned as the localScores element of the output list.

Details

The length of the simulated sequences is an argument specific to the function provided for simulation. Thus, it has to be provided also in the parameter `simulated_sequence_length` in the arguments of the "Monte Carlo - Karlin" function. It is a crucial detail as it influences precision and computation time of the result. Note that to get an appropriate estimation, the given average score must be non-positive. Be careful that the parameters names of the function `FUN` should differ from those of `karlinMonteCarlo` function.
Methods - Parameters $K^\star$ and λ of Karlin and Dembo (1990) are estimated by a linear regression on the log(-log(cumulative distribution function of the local scores)) on shorter sequences (size `simulated_sequence_length`). The formula used are : $\hat{\lambda} = -\hat{b}$ and $\hat{K^\star} = exp(\hat{a})/simulated\_sequence\_length$ where $\hat{a}$ is the intercept of the regression and $\hat{b}$ is the slope of the regression. Then p-value is given by $p = exp(-K^\star * exp(-\lambda * x))$ where $x = local\_score - \log(sequence\_length)/\lambda$.
The density plot produced by `plot == TRUE` depends on the type of the simulated local scores: if they are integer, a barplot of relative frequency is used, else `plot(density(...))` is used.
This function calls `localScoreC` which type of the output depends on the type of the input. To be efficient, be aware to use a simulating function `FUN` that return a vector of adequate type ("integer" or "numeric"). Warning: in R, `typeof(c(1,3,4,10)) == "double"`. You can set a type of a vector with `mode()` or `as.integer()` functions for example.
`karlinMonteCarlo_double()` is deprecated. At this point, it is just a call to `karlinMonteCarlo()` function.

Value

If `keepSimu` is FALSE, returns a list containing:

`p_value`	Probability to obtain a local score with a value greater or equal to the parameter `local_score`
`K*`	Parameter K^* defined in Karlin and Dembo (1990)
`lambda`	Parameter λ defined in Karlin and Dembo (1990)

If `keepSimu` is TRUE, returns a list containing:

`p_value`	Probability to obtain a local score with a value greater or equal to the parameter `local_score`
`K*`	Parameter K^* defined in Karlin and Dembo (1990)
`lambda`	Parameter λ defined in Karlin and Dembo (1990)
`localScores`	Vector of size `numSim` containing the simulated local scores for sequence size of `simulated_sequence_leng`

See Also

`monteCarlo` `localScoreC`

Examples

```
mySeq <- sample(-7:6, replace = TRUE, size = 100000)
```

```
#MonteCarlo taking random sample from the input sequence itself
karlinMonteCarlo(local_score = 160, sequence_length = 100000,
               simulated_sequence_length = 1000,
               FUN = function(x, sim_length) {
                        return(sample(x = x,
                               size = sim_length,
                               replace = TRUE))
                     },
               x = mySeq,
               sim_length = 1000,
               numSim = 1000)


#Markovian example (longer computation)
MyTransMat_reels <-  matrix(c(0.3, 0.1, 0.1, 0.1, 0.4,
                              0.2, 0.2, 0.1, 0.2, 0.3,
                              0.3, 0.4, 0.1, 0.1, 0.1,
                              0.3, 0.3, 0.1, 0.2, 0.1,
                              0.2, 0.1, 0.2, 0.4, 0.1),
                              ncol = 5, byrow=TRUE)
karlinMonteCarlo(local_score = 18.5, sequence_length = 200000,
                 simulated_sequence_length = 1500,
                 FUN = transmatrix2sequence,
                 matrix = MyTransMat_reels,
                 score =c(-1.5,-0.5,0,0.5,1), length = 1500,
                 plot=TRUE, numSim = 1500)
```

`lindley` *Lindley process*

Description

Creates a sequence of a Lindley process, also called CUSUM process, on a given sequence. For a sequence (X_k)k, the Lindley process is defined as follows: W_0:=0 and W_(k+1)=max(0,W_k+X_(k+1)). It defines positive excursions above 0.

Usage

```
lindley(sequence)
```

Arguments

`sequence`	numeric sequence of a Lindley process, eg service time per customer

Value

a vector with the Lindley process steps

Examples

```
MySeq <- c(1,2,3,-4,1,-3,-1,2,3,-4,1)
lindley(MySeq)
plot(1:length(MySeq),lindley(MySeq),type='b')
```

---

loadMatrixFromFile *Loads matrix from csv-File*

Description

Reads a csv file without header and returns the matrix. For file formats please see section "File Formats" in vignette.

Usage

```
loadMatrixFromFile(filepath)
```

Arguments

filepath	optional: Location of file on disk. If not provided, a file picker dialog will be opened.

Value

A Matrix Object

---

loadScoreFromFile *Load score from file*

Description

Reads a csv file containing an alphabet, the associated scores, and eventually the associated probabilities.

Usage

```
loadScoreFromFile(filepath, header = TRUE, ...)
```

Arguments

filepath	optional: location of file on disk. If not provided, a file picker dialogue box will be opened.
header	does the file contain a header line (defaut to TRUE)
...	optional: use arguments from read.csv

Details

The file should contains a header and 2 or 3 columns : first column the letters, second column the associated scores, optional third column associated probabilities. Letters should be unique and probabilities should sum to 1.

Value

A data.frame. Rownames correspond to the first column, usually Letters. Associated numerical scores are in the second column. If probabilities are provided, they will be loaded too and presumed to be in the third column

---

`localScoreC` *Local score*

Description

Calculates the local score for a sequence of scores, the sub-optimal segments, and the associated record times. The local score is the maximal sum of values contained in a segment among all possible segments of the sequence. In other word, it generalizes a sliding window approach, considering of all possible windows size.

Usage

```
localScoreC(v, suppressWarnings = FALSE)

localScoreC_double(v, suppressWarnings = FALSE)

localScoreC_int(v, suppressWarnings = FALSE)
```

Arguments

`v`	a sequence of numerical values as vector (integer or double).
`suppressWarnings`	(optional) if warnings should not be displayed

Details

The `localScoreC` function is implemented in a templated C function. Be aware that the type of the output (`integer` or `double`) depends on the type of the input. The function `localScoreC_double` `localScoreC_int` explicitly use the corresponding type (with an eventual conversion in case of integer). Warning: in R, `typeof(c(1,3,4,10)) == "double"`. You can set a type of a vector with `mode()` or `as.integer()` functions for example.
`localScoreC_int` is just a call to `as.integer()` before calling `localScoreC`. `localScore_double` is just a call to `localScoreC`, and as such is deprecated.

Value

A list containing:

`localScore`	the local score value and the begin and end index of the segment realizing this optimal score;
`suboptimalSegmentScores`	An array containing sub-optimal local scores, that is all the local maxima of the Lindley rocess (non negative excursion) and their begin and end index;
`RecordTime`	The record times of the Lindley process as defined in Karlin and Dembo (1990).

See Also

`lindley`

Examples

```
localScoreC(c(1.2,-2.1,3.5,1.7,-1.1,2.3))
# one segment realizing the local score value
seq.OneSegment <- c(1,-2,3,1,-1,2)
localScoreC(seq.OneSegment)
seq.TwoSegments <- c(1,-2,3,1,2,-2,-2,-1,1,-2,3,1,2,-1,-2,-2,-1,1)
# two segments realizing the local score value
localScoreC(seq.TwoSegments)
# only the first realization
localScoreC(seq.TwoSegments)$localScore
# all the realization of the local together with the suboptimal ones
localScoreC(seq.TwoSegments)$suboptimalSegmentScores
# for small sequences, you can also use lindley() function to check if
# several segments achieve the local Score
lindley(seq.TwoSegments)
plot(1:length(seq.TwoSegments),lindley(seq.TwoSegments),type='b')
seq.TwoSegments.InSameExcursion <- c(1,-2,3,2,-1,0,1,-2,-2,-4,1)
localScoreC(seq.TwoSegments.InSameExcursion)
# lindley() shows two realizations in the same excursion (no 0 value between the two LS values)
lindley(seq.TwoSegments.InSameExcursion)
# same beginning index but two possible ending indexes
# only one excursion realizes the local score even in there is two possible lengths of segment
localScoreC(seq.TwoSegments.InSameExcursion)$suboptimalSegmentScores
plot(1:length(seq.TwoSegments.InSameExcursion),lindley(seq.TwoSegments.InSameExcursion),type='b')
# Technical note about type correspondance
seq.OneSegment <- c(1,-2,3,1,-1,2)
seq.OneSegmentI <- as.integer(seq.OneSegment)
typeof(seq.OneSegment)  # "double" (beware)
typeof(seq.OneSegmentI) # "integer"
LS1 <- localScoreC(seq.OneSegment, suppressWarnings = TRUE)
LS1I <- localScoreC(seq.OneSegmentI, suppressWarnings = TRUE)
typeof(LS1$localScore)  # "double"
typeof(LS1I$localScore) # "integer"
typeof(LS1$suboptimalSegmentScores$value)  # "double"
typeof(LS1I$suboptimalSegmentScores$value) # "integer"
# Force to use integer values (trunk values if necessary)
```

```
seq2 <- seq.OneSegment + 0.5
localScoreC(seq2, suppressWarnings = TRUE)
localScoreC_int(seq.OneSegment, suppressWarnings = TRUE)
```

`MarkovParameters` *Functions parameters global description*

Description

Functions parameters global description

Arguments

`theta`	alphabet used (vector of character)
`lambda`	transition probability matrix of size, theta x theta
`score_function`	vector containing the scores of each letters of the alphabet (must be in the same order as theta)
`a`	score strictly positive
`prob0`	Distribution of the first element of the Markov chain. Default to stationary distribution of lambda

---

`maxPartialSumd` *Maximum of the partial sum [probability] [iid]*

Description

Calculates the distribution of the maximum of the partial sum process for a given value in the identically and independently distributed model

Usage

```
maxPartialSumd(
  k,
  score_probabilities,
  sequence_min = NULL,
  sequence_max = NULL,
  score_values = NULL
)
```

Arguments

k	value at which calculates the probability
score_probabilities	the probabilities for each score from lowest to greatest (Optionnaly with scores as names)
sequence_min	minimum score (optional if score_values OR names(score_probabilities) is defined)
sequence_max	maximum score (optional if score_values OR names(score_probabilities) is defined)
score_values	vector of integer score values, associated to score_probabilities (optional if sequence_min and sequence_max OR names(score_probabilities) are defined)

Details

Implement the formula (4) of the article Mercier, S., Cellier, D., & Charlot, D. (2003). An improved approximation for assessing the statistical significance of molecular sequence features. Journal of Applied Probability, 40(2), 427-441. doi:10.1239/jap/1053003554
Important note : the calculus of the parameter of the distribution uses the resolution of a polynome which is a function of the score distribution, of order max(score)-min(score). There exists only empirical methods to solve a polynome of order greater that 5 with no warranty of reliable solution. The found roots are checked internally to the function and an error message is throw in case of inconsistency.

Value

A double representing the probability of the maximum of the partial sum process equal to k

Examples

```
maxPartialSumd(10, c(0.08, 0.32, 0.08, 0.00, 0.08, 0.00, 0.00, 0.08, 0.02, 0.32, 0.02), -6, 4)
prob <- c(0.08, 0.32, 0.08, 0.00, 0.08, 0.00, 0.00, 0.08, 0.02, 0.32, 0.02)
score_values <- which(prob != 0) - 7 # keep only non null probability scores
prob0 <- prob[prob != 0]             # and associated probability
p <- maxPartialSumd(10, prob, sequence_min = -6, sequence_max =  4)
p0 <- maxPartialSumd(10,prob0, score_values = score_values)
p == p0 # TRUE
```

---

mcc *MCC [p-value] [iid]*

Description

Calculates an approximated p-value for a given local score value and a medium to long sequence length in the identically and independently distributed model

Usage

```
mcc(
  local_score,
  sequence_length,
  score_probabilities,
  sequence_min = NULL,
  sequence_max = NULL,
  score_values = NULL
)
```

Arguments

`local_score`	the observed local score
`sequence_length`	length of the sequence
`score_probabilities`	the probabilities for each score from lowest to greatest (Optionnaly with scores as names)
`sequence_min`	minimum score (optional if `score_values` OR names(score_probabilities) is defined)
`sequence_max`	maximum score (optional if `score_values` OR names(score_probabilities) is defined)
`score_values`	vector of integer score values, associated to score_probabilities (optional if `sequence_min` and `sequence_max` OR names(score_probabilities) are defined)

Details

This methods is actually an improved method of Karlin and produces more precise results. It should be privileged whenever possible.
As with karlin, the method works the better the longer the sequence. Important note : the calculus of the parameter of the distribution uses the resolution of a polynome which is a function of the score distribution, of order max(score)-min(score). There exists only empirical methods to solve a polynome of order greater that 5 with no warranty of reliable solution. The found roots are checked internally to the function and an error message is throw in case of inconsistency. In such case, you could try to change your score scheme (in case of discretization) or use the function `karlinMonteCarlo` .

Value

A double representing the probability of a local score as high as the one given as argument

See Also

`karlin`, `daudin`, `karlinMonteCarlo`, `monteCarlo`

Examples

```
mcc(40, 100, c(0.08, 0.32, 0.08, 0.00, 0.08, 0.00, 0.00, 0.08, 0.02, 0.32, 0.02), -6, 4)
mcc(40, 10000, c(0.08, 0.32, 0.08, 0.00, 0.08, 0.00, 0.00, 0.08, 0.02, 0.32, 0.02), -6, 4)
mcc(150, 10000, c(0.08, 0.32, 0.08, 0.00, 0.08, 0.00, 0.00, 0.08, 0.02, 0.32, 0.02), -5, 5)
p1 <- mcc(local_score = 15, sequence_length = 5000,
       score_probabilities = c(0.2, 0.3, 0.1, 0.2, 0.1, 0.1),
       sequence_min = -3, sequence_max = 2)
p2 <- mcc(local_score = 15, sequence_length = 5000,
       score_probabilities = c(0.2, 0.3, 0.1, 0.2, 0.1, 0.1),
       score_values = -3:2)
p1 == p2 # TRUE

prob <- c(0.08, 0.32, 0.08, 0.00, 0.08, 0.00, 0.00, 0.08, 0.02, 0.32, 0.02)
score_values <- which(prob != 0) - 6 # keep only non null probability scores
prob0 <- prob[prob != 0]             # and associated probability
p <- mcc(150, 10000, prob, sequence_min = -5, sequence_max =  5)
p0 <- mcc(150, 10000, prob0, score_values = score_values)
names(prob0) <- score_values
p1 <- mcc(150, 10000, prob0)
p == p0 # TRUE
p == p1 # TRUE
```

`monteCarlo` *Monte Carlo method [p-value]*

Description

Calculates an empirical p-value based on Monte Carlo simulations. Perfect for small sequences (both markov chains and identically and independently distributed) with length ~ 10^3.

Usage

```
monteCarlo(local_score, FUN, ..., plot = TRUE, numSim = 1000, keepSimu = FALSE)

monteCarlo_double(
  local_score,
  FUN,
  ...,
  plot = TRUE,
  numSim = 1000,
  keepSimu = FALSE
)
```

Arguments

`local_score`	local score observed in a segment.
`FUN`	function to simulate similar sequences with.
`...`	parameters for FUN

plot	boolean value if to display plots for cumulated function and density
numSim	number of sequences to generate during simulation
keepSimu	Boolean, default to FALSE. If TRUE, the simulated local scores are returned as the localScores element of the output list.

Details

Be careful that the parameters names of the function FUN should differ from those of monteCarlo function.
The density plot produced by `plot == TRUE` depends on the type of the simulated local scores: if they are integer, a barplot of relative frequency is used, else `plot(density(...))` is used.
This function calls localScoreC which type of the output depends on the type of the input. To be efficient, be aware to use a simulating function FUN that return a vector of adequate type ("integer" or "numeric"). Warning: in R, `typeof(c(1,3,4,10)) == "double"`. You can set a type of a vector with `mode()` or `as.integer()` functions for example.
`monteCarlo_double()` is deprecated. At this point, it is just a call to `monteCarlo()` function.

Value

If keepSimu is FALSE, returns a numeric value corresponding to the probability to obtain a local score with value greater or equal to the parameter `local_score`.
If keepSimu is TRUE, returns a list containing:

p_value	Floating value corresponding to the probability to obtain a local score with a value greater or equal to the para
localScores	Vector of size numSim containing the simulated local scores

Functions

- `monteCarlo_double()`: Monte-Carlo function for double [deprecated]

See Also

karlinMonteCarlo localScoreC

Examples

```
monteCarlo(120, FUN = rbinom, n = 100, size = 5, prob=0.2)


mySeq <- sample(-7:3, replace = TRUE, size = 1000)
monteCarlo(local_score = 18, FUN = function(x) {return(sample(x = x,
           size = length(x), replace = TRUE))}, x = mySeq)


#Examples of non integer score function
mySeq2 <- sample(-7:6 - 0.5, replace = TRUE, size = 1000)
monteCarlo(local_score = 50.5, FUN = function(x) {return(sample(x = x,
           size = length(x), replace = TRUE))}, x = mySeq2)
```

```
#Examinating simulated local scores
mySeq2 <- sample(-7:6, replace = TRUE, size = 1000)
simu <- monteCarlo(local_score = 50.5, FUN = function(x) {return(sample(x = x,
           size = length(x), replace = TRUE))}, x = mySeq2, keepSimu = TRUE)
hist(simu$localScores)


# Markovian example
MyTransMat <-
   matrix(c(0.3,0.1,0.1,0.1,0.4, 0.2,0.2,0.1,0.2,0.3, 0.3,0.4,0.1,0.1,0.1, 0.3,0.3,0.1,0.0,0.3,
             0.1,0.1,0.2,0.3,0.3), ncol = 5, byrow=TRUE)
monteCarlo(local_score = 50,
          FUN = transmatrix2sequence, matrix = MyTransMat,
          length=150, score = c(-2,-1,0,2,3), plot=FALSE, numSim = 5000)
```

`proba_theoretical_first_excursion_iid` *Probability* $P(Q(1) \geq a)$ *that the height of the first excursion is greater or equal to* `a` *given a i.i.d. model on the letters sequence*

Description

Mathematical definition of an excursion of the Lindley process is based on the record times of the partial sum sequence associated to the score sequence (see Karlin and Altschul 1990, Karlin and Dembo 1992) and define the successive times where the partial sums are strictly decreasing. There must be distinguished from the visual excursions of the Lindley sequence. The number `i` is the number of excursion in sequential order. Detailed definitions are given in the vignette.

Usage

```
proba_theoretical_first_excursion_iid(
  a,
  theta,
  theta_distribution,
  score_function
)
```

Arguments

`a`	score strictly positive
`theta`	vector containing the alphabet used
`theta_distribution`	distribution vector of theta
`score_function`	vector containing the scores of each letters of the alphabet (must be in the same order as theta)

Details

Beware that a sequence beginning with a negative score gives a "flat" excursion, with score 0 are considered.

Value

theoretical probability of reaching a score of a on the first excursion supposing an i.i.d model on the letters sequence

Examples

```
proba_theoretical_first_excursion_iid(3, c("a","b","c","d"),
                                      c(a=0.1,b=0.2,c=0.4,d=0.3), c(a=-3,b=-1,c=1,d=2))
```

---

proba_theoretical_ith_excursion_iid

Probability $P(Q(i) \geq a)$ that the height of the ith excursion (sequential order) is greater or equal to `a` *given a i.i.d. model on the letters sequence*

Description

Mathematical definition of an excursion of the Lindley process is based on the record times of the partial sum sequence associated to the score sequence (see Karlin and Altschul 1990, Karlin and Dembo 1992) and define the successive times where the partial sums are strictly decreasing. There must be distinguished from the visual excursions of the Lindley sequence. The number `i` is the number of excursion in sequential order. Detailed definitions are given in the vignette.

Usage

```
proba_theoretical_ith_excursion_iid(
  a,
  theta,
  theta_distribution,
  score_function,
  i = 1
)
```

Arguments

`a`	score strictly positive
`theta`	vector containing the alphabet used
`theta_distribution`	distribution vector of theta
`score_function`	vector containing the scores of each letters of the alphabet (must be in the same order as theta)
`i`	Number of excursion in sequential order

Details

In the i.i.d., the distribution of the ith excursion is the same as the first excursion. This function is just for convenience, and the result is the same as `proba_theoretical_first_excursion_iid`. Beware that a sequence beginning with a negative score gives a "flat" excursion, with score 0 are considered.

Value

theoretical probability of reaching a score of a on the ith excursion supposing an i.i.d model on the letters sequence

See Also

`proba_theoretical_first_excursion_iid`

Examples

```
p1 <- proba_theoretical_ith_excursion_iid(3, c("a","b","c","d"),
                                c(a=0.1,b=0.2,c=0.4,d=0.3), c(a=-3,b=-1,c=1,d=2), i = 10)
p2 <- proba_theoretical_first_excursion_iid(3, c("a","b","c","d"),
                                      c(a=0.1,b=0.2,c=0.4,d=0.3), c(a=-3,b=-1,c=1,d=2))
p1 == p2  #TRUE
```

---

proba_theoretical_ith_excursion_markov

theoretical probability of reaching the threshold score a on the i-th excursion (sequential order) of a markov's score sequence

---

Description

This function implements the results of the paper: "Exact distribution of excursion heights of the Lindley process in a Markovian model", this is an exact method. Scores of score function have to be integers, the expectancy of the score have to be negative, the score function have to contains at least one positive integer, i have to be >0

Usage

```
proba_theoretical_ith_excursion_markov(
  a,
  theta,
  lambda,
  score_function,
  i,
  prob0 = NULL,
  epsilon = 1e-16
)
```

Arguments

a	threshold score
theta	vector containing the alphabet used
lambda	transition probability matrix of the markov chain, square, same size and order as theta
score_function	vector containing the scores of each letters of the alphabet (must be in the same order as theta)
i	number of wanted excursion (sequential order)
prob0	probability distribution of the first letter of the sequence (stationary distribution of lambda if the parameter is NULL), used only for computation of proba_q_i_geq_a
epsilon	threshold for the computation of matrix M: to calculate the probabilities, this function need to compute the matrix M (transition probability matrix of the beginning of excursions, as in the paper) with a recurrence, this recurrence stop at a certain number of iterations or when the maximal absolute value of the difference between an iteration and the next one is < epsilon.

Details

Be careful: the computational time is exponential in function of the maximum score of the score function. The computational time also increase with the rise of the threshold score a and with the lowering of the minimum of the score function and have a cubic complexity in function of the size of theta. Lowering epsilon can also decrease the execution time of the function but it can have an impact on the accuracy of the probabilities.

Value

a list containing:

proba_q_i_geq_a	reel number between 0 and 1 representing the theoretical probability that the i-th excursion is greater or e
P_alpha	vector containing the probabilities (of reaching the threshold score on the i-th excursion) depending on th

Examples

```
## In this example: the probability to reach a score of a in the third excursion is: 0.04237269
## The conditional probabilities to reach a score of a in the third excursion if the first
## letter of the sequence is K, L or M are respectively: 0.04239004, 0.04247805 and 0.04222251
proba_theoretical_ith_excursion_markov(a = 5,
               theta = c("K","L","M"),
               lambda = matrix(c(0.5, 0.3, 0.2,
                                 0.4, 0.2, 0.4,
                                 0.4, 0.4, 0.2),
                               ncol = 3, byrow = TRUE),
               score_function = c(-2,-1,2),
               i = 3,
               prob0 = c(0.4444444, 0.2962963, 0.2592593))
### This example implements the numerical application of the paper,
```

```
###    the global probability is 0.2095639
proba_theoretical_ith_excursion_markov(a = 6,
            theta = c("a","b","c","d","e"),
            lambda = matrix(c(0.1, 0.7, 0.05, 0.05, 0.1,
                              0.3, 0.3, 0.2, 0.15, 0.05,
                              0.1, 0.4, 0.15, 0.2, 0.15,
                              0.5, 0.05, 0.25, 0.1, 0.1,
                              0.25, 0.05, 0.5, 0.15, 0.05),
                              ncol = 5,nrow=5, byrow = TRUE),
            score_function = c(-3,-2,-1,6,7),
            i = 3)
```

`RealScores2IntegerScores` *Convert a real scores vector into an integer scores vector*

Description

Convert real scores into integer scores

Usage

```
RealScores2IntegerScores(RealScore, ProbRealScore, coef = 10)
```

Arguments

`RealScore`	vector of real scores
`ProbRealScore`	vector of probability
`coef`	coefficient

Details

Convert real scores into integer scores by multiplying real scores by a coefficient (default 10) and then assigning probability to corresponding extended (from the minimum to the maximum) integer scores

Value

list containing ExtendedIntegerScore and ProbExtendedIntegerScore

Examples

```
score <- c(-1,-0.5,0,0.5,1)
prob.score <- c(0.2,0,0.4,0.1,0.3)
(res1 <- RealScores2IntegerScores(score, prob.score, coef=10))
prob.score.err <- c(0.1,0,0.4,0.1,0.3)
(res2 <- RealScores2IntegerScores(score, prob.score.err, coef=10))
# When coef=1, the function can handle integer scores
ex.integer.score <- c(-3,-1,0,1, 5)
(res3 <- RealScores2IntegerScores(ex.integer.score, prob.score, coef=1))
```

recordTimes *Calculate the record times of a sequence*

Description

For a given sequence of real numbers, this function returns the index of the values where Lindley/CUSUM process are <0

Usage

```
recordTimes(sequence)
```

Arguments

sequence	numeric sequence of a Lindley process, eg service time per customer

Details

Note that the first record times which is always 0 is not included in the returned vector

Value

a vector with the record times. If no record time are found, return empty vector `integer(0)`

Examples

```
####This example should return this vector: c(1,3,4,5,10)
seq1 <- c(-1,2,-2.00001,-4,-1,3,-1,-2,3,-4,2)
recordTimes(seq1)
####This example should return integer(0) because there is no record times in this sequence
seq2 <- c(4,1,0,-1,-2,5,-5,0,4)
recordTimes(seq2)
```

scoreSequences2probabilityVector *Empirical distribution from sequences*

Description

Builds empirical distribution from a list of numerical sequences

Usage

```
scoreSequences2probabilityVector(sequences)
```

Arguments

sequences	list of numerical sequences

Details

By determining the extreme scores in the sequences, this function creates a vector of probabilities including values that do not occur at all. In this it differs from table(). For example, two sequences containing values from 1:2 and 5:6 will produce a vector of size 6.

Value

empirical distribution from minimum score to maximum score as a vector of floating numbers.

Examples

```
seq1 = sample(7:8, size = 10, replace = TRUE)
seq2 = sample(2:3, size = 15, replace = TRUE)
l = list(seq1, seq2)
scoreSequences2probabilityVector(l)
```

Seq1093 *Long protein sequence*

Description

A long protein sequence.

Usage

```
data(Seq1093)
```

Format

A character string with 1093 characters corresponding to Q60519.fasta in UniProt Data base.

Source

https://www.uniprot.org/

Examples

```
data(Seq1093)
Seq1093
nchar(Seq1093)
data(HydroScore)
seqScore <- CharSequence2ScoreSequence(Seq1093,HydroScore)
seqScore[1:50]
localScoreC(seqScore)$localScore
LS <- localScoreC(seqScore)$localScore[1]
```

```
prob1 <- scoreSequences2probabilityVector(list(seqScore))
daudin(local_score = LS, sequence_length = nchar(Seq1093),
       score_probabilities = prob1,
       sequence_min = min(seqScore),
       sequence_max = max(seqScore))
karlin(local_score = LS, sequence_length = nchar(Seq1093),
       score_probabilities = prob1,
       sequence_min = min(seqScore),
       sequence_max = max(seqScore))
```

Seq219 *Protein sequence*

Description

A protein sequence.

Usage

```
data(Seq219)
```

Format

A character string with 219 characters corresponding to P49755.fasta query in UniProt Data base.

Source

https://www.uniprot.org/

Examples

```
data(Seq219)
Seq219
nchar(Seq219)
data(HydroScore)
seqScore=CharSequence2ScoreSequence(Seq219,HydroScore)
seqScore[1:30]
localScoreC(seqScore)$localScore
prob1 <- scoreSequences2probabilityVector(list(seqScore))
daudin(local_score = 52, sequence_length = nchar(Seq219),
               score_probabilities = prob1,
               sequence_min = min(seqScore),
               sequence_max = max(seqScore))
score <- -5:5
prob2 <- c(0.15,0.15,0.1,0.1,0.0,0.05,0.15,0.05,0.2,0.0,0.05)
daudin(local_score = 52, sequence_length = nchar(Seq219),
       score_probabilities = prob2,
       sequence_min = min(seqScore),
       sequence_max = max(seqScore))
```

Seq31 *Short protein sequence*

Description

A short protein sequence of 31 amino acids corresponding to Q09FU3.fasta query in UniProt Data base.

Usage

```
data(Seq31)
```

Format

A character string with 31 characters "MLTITSYFGFLLAALTITSVLFIGLNKIRLI"

Source

https://www.uniprot.org/

Examples

```
data(Seq31)
Seq31
nchar(Seq31)
data(HydroScore)
SeqScore <- CharSequence2ScoreSequence(Seq31,HydroScore)
SeqScore
localScoreC(SeqScore)$localScore
LS <- localScoreC(SeqScore)$localScore[1]
prob1 <- scoreSequences2probabilityVector(list(SeqScore))
daudin(local_score = LS, sequence_length = nchar(Seq31),
               score_probabilities = prob1,
               sequence_min = min(SeqScore),
               sequence_max = max(SeqScore))
score <- -5:5
prob2 <- c(0.15,0.15,0.1,0.1,0.0,0.05,0.15,0.05,0.2,0.0,0.05)
sum(prob2*score)
karlin(local_score = LS, sequence_length = nchar(Seq31),
       score_probabilities = prob2,
       sequence_min = min(SeqScore),
       sequence_max = max(SeqScore))
```

SeqListSCOPe *Several sequences*

Description

A vector of character strings

Usage

```
data(SeqListSCOPe)
```

Format

A list of 285 character strings with their entry codes as names

Source

Structural Classification Of Proteins database (SCOP). More precisely this data contain the 285 protein sequences of the data called "CF_scop2dom_20140205aa" with length from 31 to 404.

Examples

```
data(SeqListSCOPe)
head(SeqListSCOPe)
SeqListSCOPe[1]
nchar(SeqListSCOPe[1])
summary(sapply(SeqListSCOPe, nchar))
data(HydroScore)
MySeqScoreList <- lapply(SeqListSCOPe, FUN=CharSequence2ScoreSequence, HydroScore)
head(MySeqScoreList)
AA <- automatic_analysis(sequences=MySeqScoreList, model='iid')
AA[[1]]
# the p-value of the first 10 sequences
sapply(AA, function(x){x$`p-value`})[1:10]
# the 20th smallest p-values
sort(sapply(AA, function(x){x$`p-value`}))[1:20]
which(sapply(AA, function(x){x$`p-value`})<0.05)
table(sapply(AA, function(x){x$`method`}))
# The maximum sequence length equals 404 so it here normal that the exact method is used for
# all the 606 sequences of the data base
# Score distribution learned on the data set
scoreSequences2probabilityVector(MySeqScoreList)
```

sequences2transmatrix *Transition matrix from sequence(s)*

Description

Calculates the transition matrix by counting occurrences of tuples in given vector list

Usage

```
sequences2transmatrix(sequences)
```

Arguments

sequences	Sequences to be analyzed, can be a list of vectors, or a vector

Details

In output, score_value is coerced as integer if possible. Else, it is a character vector containing the states of the Markov chain.

Value

A list object containing

transition_matrix	Transition Matrix with row names and column names are the associated score/state
score_value	a vector containing the score/state, ordered in the same way as the matrix columns and rows

Examples

```
myseq <- sample(LETTERS[1:2], size=20,replace=TRUE)
sequences2transmatrix(myseq)
```

SJSyndrome *Stevens-Johnson syndrome data*

Description

The Stevens-Johnson syndrome is an acute and serious dermatological disease due to a drug allergy. The syndrome appearance is life-threatening emergency. They are very rare, around 2 cases per million people per year.

Usage

```
data(SJSyndrome.data)
```

Format

A data.frame of 824 lines, each describing a syndrome appearance described by 15 covariates: Case ID, Initial FDA Received Date, days since last fda, Event Date, Latest FDA Received Date, Suspect Product Names, Suspect Product Active Ingredients, Reason for Use, Reactions, Serious, Outcomes, Sex, Patient Age, Sender, Concomitant Product Names The third column correspond to the number of days between two adverse events.

Source

FDA open data.

Examples

```
data(SJSyndrome)
summary(SJSyndrome)
```

---

stationary_distribution

Stationary distribution [Markov chains]

Description

Calculates stationary distribution of markov transition matrix by use of eigenvectors of length 1

Usage

```
stationary_distribution(m)
```

Arguments

m	Transition Matrix [matrix object]

Value

A vector with the probabilities

Examples

```
B = t(matrix (c(0.2, 0.8, 0.4, 0.6), nrow = 2))
stationary_distribution(B)
```

`transmatrix2sequence` *Sampling function for Markov chains*

Description

Creates Markov chains based on a transition matrix. Can be used as parameter for the Monte Carlo function.

Usage

```
transmatrix2sequence(matrix, length, initialIndex, score)
```

Arguments

`matrix`	transition matrix of Markov process
`length`	length of sequence to be sampled
`initialIndex`	(optional) index of matrix which should be initial value of sequence. If none supplied, a value from the stationary distribution is sampled as initial value.
`score`	(optional) a vector representing the scores (in ascending order) of the matrix index. If supplied, the result will be a vector of these values.

Details

The transition matrix is considered representing the transition from one score to another such that the score in the first row is the lowest and the last row are the transitions from the highest score to another. The matrix must be stochastic (no rows filled up with only '0' values).

It is possible to have the same score for different states of the markov chain. If no score supplied, the function returns a markov chain with state in 1:ncol(matrix)

Value

a Markov chain sampled from the transition matrix

Examples

```
B <-  matrix (c(0.2, 0.8, 0.4, 0.6), nrow = 2, byrow = TRUE)
transmatrix2sequence(B, length = 10)
transmatrix2sequence(B, length = 10, score = c(-2,1))
transmatrix2sequence(B, length = 10, score = c("A","B"))
MyTransMat <-
   matrix(c(0.3,0.1,0.1,0.1,0.4,
            0.2,0.2,0.1,0.2,0.3,
            0.3,0.4,0.1,0.1,0.1,
            0.3,0.3,0.1,0.0,0.3,
            0.1,0.1,0.2,0.3,0.3),
            ncol = 5, byrow=TRUE)
MySeq.CM <- transmatrix2sequence(matrix = MyTransMat,length=90, score =c(-2,-1,0,2,3))
MySeq.CM
```

Index