Package: markovchain (via r-universe)

October 10, 2026

Type Package

Title Easy Handling Discrete Time Markov Chains

Version 1.2

Maintainer Giorgio Alfredo Spedicato <spedicato_giorgio@yahoo.it>

Description Description: Functions and S4 classes to create, manage and analyse discrete time Markov chains. Includes probabilistic analysis of chain structure (state classification, hitting times, stationary distributions, spectral and mixing diagnostics), statistical inference (maximum likelihood and Bayesian estimation, tests of the Markov property, homogeneity, stationarity and order, simulation), higher-order and multivariate chains, state aggregation, and import/export to JSON, YAML and CSV. Basic support for continuous time Markov chains is also provided; some of these functions depend on the suggested 'ctmcd' package. See Spedicato (2017) <doi:10.32614/RJ-2017-036>.

License MIT + file LICENSE

Depends R (>= 4.4.0), Matrix (>= 1.5-0), methods

Imports igraph (>= 1.0.0), expm, stats4, parallel, Rcpp (>= 1.0.2), RcppParallel, utils, stats, grDevices

Suggests knitr, testthat, diagram, DiagrammeR, ggplot2 (>= 3.4.0), msm, Rsolnp, rmarkdown, ctmcd, bookdown, rticles, MCMCpack, microbenchmark, quarto, jsonlite, yaml, xml2

Enhances etm

VignetteBuilder rmarkdown, knitr, bookdown, rticles, quarto

LinkingTo Rcpp, RcppParallel, RcppArmadillo (>= 0.9.600.4.0)

SystemRequirements GNU make, Quarto

LazyLoad yes

ByteCompile yes

Encoding UTF-8

BugReports https://github.com/spedygiorgio/markovchain/issues

URL https://github.com/spedygiorgio/markovchain/

NeedsCompilation yes

Config/roxygen2/version 8.1.0

Author Giorgio Alfredo Spedicato [aut, cre] (ORCID: <https://orcid.org/0000-0002-0315-8888>), Tae Seung Kang [aut], Sai Bhargav Yalamanchi [aut], Mildenberger Thoralf [ctb] (ORCID: <https://orcid.org/0000-0001-7242-1873>), Deepak Yadav [aut], Ignacio Cordón [aut] (ORCID: <https://orcid.org/0000-0002-3152-0231>), Vandit Jain [ctb], Toni Giorgino [ctb] (ORCID: <https://orcid.org/0000-0001-6449-0596>), Richèl J.C. Bilderbeek [ctb] (ORCID: <https://orcid.org/0000-0003-1107-7049>), Daniel Ebbert [ctb] (ORCID: <https://orcid.org/0000-0003-3666-7205>), Shreyash Maheshwari [ctb], Reinhold Koch [ctb]

Config/pak/sysreqs make

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

Date/Publication 2026-10-10 13:52:56 UTC

RemoteUrl https://github.com/cran/markovchain

RemoteRef HEAD

RemoteSha e4a696a5c5e0ef57bcea7707ccaff7912fade101

Contents

markovchain-package 4
absorptionProbabilities 5
aggregateStates 6
assessIndependence 9
autoLump 10
autoplot.markovchain 11
birthDeath 12
blanden 14
closestReversible 15
committorAB 17
conditionalDistribution 18
craigsendi 19
ctmc-class 20
ctmcFit 21
dirichletChain 22
entropyRate 23
expectedRewards 25
expectedRewardsBeforeHittingA 26
ExpectedTime 26
firstPassage 27
firstPassageMultiple 28

fitHigherOrder . . . 29
fitHighOrderMultivarMC . . . 31
fitMTD . . . 32
freq2Generator . . . 34
fundamentalMatrix . . . 35
gamblersRuin . . . 36
generatorToTransitionMatrix . . . 37
higherOrderLogLik . . . 38
HigherOrderMarkovChain-class . . . 40
higherOrderPredict . . . 40
hittingProbabilities . . . 42
holson . . . 43
hommc-class . . . 44
ictmc-class . . . 45
identityChain . . . 46
impliedTimescales . . . 47
impreciseProbabilityatT . . . 48
inferHyperparam . . . 49
is.accessible . . . 50
is.CTMCirreducible . . . 52
is.irreducible . . . 53
is.lumpable . . . 54
is.regular . . . 54
is.reversible . . . 55
is.stochasticallyMonotone . . . 57
is.TimeReversible . . . 58
kemenyConstant . . . 59
kullback . . . 60
lazyChain . . . 60
lump . . . 62
markovchain-class . . . 62
markovchainFit . . . 65
markovchainList-class . . . 69
markovchainListFit . . . 70
markovchainSequence . . . 71
meanAbsorptionTime . . . 73
meanFirstPassageTime . . . 74
meanNumVisits . . . 75
meanRecurrenceTime . . . 76
mergeWith . . . 77
mixingTime . . . 78
multinomialConfidenceIntervals . . . 80
name . . . 81
name<- . . . 82
names,markovchain-method . . . 82
noofVisitsDist . . . 83
normalizedEntropyRate . . . 84
ones . . . 85

period . . . 85
populationGeneticsModel . . . 87
predictHommc . . . 89
predictiveDistribution . . . 90
preproglucacon . . . 91
priorDistribution . . . 92
probabilityatT . . . 93
rain . . . 94
randomMarkovChain . . . 95
rctmc . . . 96
redistribute . . . 97
relaxationTime . . . 99
rmarkovchain . . . 100
rouwenhorst . . . 102
sales . . . 103
selectOrder . . . 104
sensitivity . . . 106
show,hommc-method . . . 108
slem . . . 108
spectralGap . . . 110
states . . . 111
steadyStates . . . 112
subchain . . . 113
tauchen . . . 115
timeCorrelations . . . 116
tm_abs . . . 118
toBoundedChain . . . 119
toDictionary . . . 120
toDot . . . 122
toFile . . . 123
toNthOrder . . . 125
topologicalEntropy . . . 126
transition2Generator . . . 127
transitionProbability . . . 128
urnModel . . . 129
verifyMarkovProperty . . . 130
zeros . . . 132

Index **133**

`markovchain-package` *Easy Handling Discrete Time Markov Chains*

Description

The package contains classes and method to create and manage (plot, print, export for example) discrete time Markov chains (DTMC). In addition it provide functions to perform statistical (fitting and drawing random variates) and probabilistic (analysis of DTMC proprieties) analysis

Author(s)

Giorgio Alfredo Spedicato Maintainer: Giorgio Alfredo Spedicato <spedicato_giorgio@yahoo.it>

References

Discrete-Time Markov Models, Bremaud, Springer 1999

See Also

Useful links:

- https://github.com/spedygiorgio/markovchain/
- Report bugs at https://github.com/spedygiorgio/markovchain/issues

Examples

```
# create some markov chains
statesNames=c("a","b")
mcA<-new("markovchain", transitionMatrix=matrix(c(0.7,0.3,0.1,0.9),byrow=TRUE,
         nrow=2, dimnames=list(statesNames,statesNames)))

statesNames=c("a","b","c")
mcB<-new("markovchain", states=statesNames, transitionMatrix=
         matrix(c(0.2,0.5,0.3,0,1,0,0.1,0.8,0.1), nrow=3,
         byrow=TRUE, dimnames=list(statesNames, statesNames)))

statesNames=c("a","b","c","d")
matrice<-matrix(c(0.25,0.75,0,0,0.4,0.6,0,0,0,0,0.1,0.9,0,0,0.7,0.3), nrow=4, byrow=TRUE)
mcC<-new("markovchain", states=statesNames, transitionMatrix=matrice)
mcD<-new("markovchain", transitionMatrix=matrix(c(0,1,0,1), nrow=2,byrow=TRUE))


#operations with S4 methods
mcA^2
steadyStates(mcB)
absorbingStates(mcB)
markovchainSequence(n=20, markovchain=mcC, include=TRUE)
```

`absorptionProbabilities` *Absorption probabilities*

Description

Computes the absorption probability from each transient state to each recurrent one (i.e. the (i, j) entry or (j, i), in a stochastic matrix by columns, represents the probability that the first not transient state we can go from the transient state i is j (and therefore we are going to be absorbed in the communicating recurrent class of j)

Usage

```
absorptionProbabilities(object)
```

Arguments

object	the markovchain object

Value

A named vector with the expected number of steps to go from a transient state to any of the recurrent ones

Author(s)

Ignacio Cordón

References

C. M. Grinstead and J. L. Snell. Introduction to Probability. American Mathematical Soc., 2012.

Examples

```
m <- matrix(c(1/2, 1/2, 0,
              1/2, 1/2, 0,
                0, 1/2, 1/2), ncol = 3, byrow = TRUE)
mc <- new("markovchain", states = letters[1:3], transitionMatrix = m)
absorptionProbabilities(mc)
```

---

aggregateStates *Aggregate a Markov chain's state space by Kullback-Leibler minimization*

Description

Reduces the state space of a finite, irreducible, aperiodic Markov chain to k macro-states by the spectral-theoretic method of Deng, Mehta and Meyn (2011): the macro-chain returned is the one whose "lifted" behavior (each macro-state visit standing in for its micro-states, weighted by their share of the stationary distribution) is closest, in Kullback-Leibler divergence rate, to the original chain.

Usage

```
aggregateStates(
  object,
  k = NULL,
  method = c("adaptive", "spectral-bottom-up", "spectral-top-down")
)
```

```
## S4 method for signature 'markovchain'
aggregateStates(
  object,
  k = NULL,
  method = c("adaptive", "spectral-bottom-up", "spectral-top-down")
)
```

Arguments

object	A markovchain object representing a finite, irreducible, aperiodic discrete-time Markov chain with at least 3 states.
k	The number of macro-states to reduce to, an integer between 2 and the number of states minus 1. The default, NULL, chooses k automatically via the eigengap heuristic: the transition matrix's eigenvalues are sorted by modulus, and k is set to the number of eigenvalues before the largest relative drop – a standard, parameter-free way to guess how many "slow", well-separated modes the chain has. This is a genuine automatic selection, unlike PyDTMC's adaptive, which only picks which of the two algorithms below to run for a k the caller must still supply.
method	One of "adaptive" (the default), "spectral-bottom-up" or "spectral-top-down". "spectral-bottom-up" grows the partition one split at a time from a single macro-state, and is the more reliable choice for a large reduction (k much smaller than the number of states); "spectral-top-down" starts from every micro-state on its own and repeatedly merges the least costly pair, which suits a small reduction (k close to the number of states). "adaptive" follows the same rule of thumb as PyDTMC: top-down below 30 states, otherwise bottom-up when k is at most 30% of the number of states and top-down otherwise.

Details

This targets the same problem as autoLump, but by a different and more principled route: autoLump clusters the leading eigenvectors with k-means (a generic, randomized heuristic, fixed here to a deterministic seed only for reproducibility), while aggregateStates greedily minimizes the actual information-theoretic quantity that measures how much the aggregation distorts the chain's dynamics.

When is the divergence exactly zero? Not simply whenever object is strongly lumpable with respect to partition in the sense of is.lumpable. Strong lumpability only requires that, for every pair of macro-states, all micro-states in the same source macro-state have the same *total* probability of moving to the destination macro-state; it says nothing about how that total is split among the destination macro-state's own members. The Kullback-Leibler divergence used here is sensitive to exactly that split: it is zero (up to rounding) only when every source state distributes its outgoing probability across a destination macro-state's members in the same proportions – those of the stationary distribution restricted to that macro-state – regardless of which source state it is. This is a genuinely stronger condition, and a strongly lumpable chain need not satisfy it: the classical Land of Oz weather chain (see the package vignette), lumped into Bad_Weather = {rainy, snowy} and Nice_Weather = {nice}, is strongly lumpable, and aggregateStates correctly recovers that exact partition as optimal and reproduces lump's aggregated transition matrix, but its divergence is

strictly positive, because rainy and snowy split their probability between rainy and snowy themselves differently from one another.

Both methods require the chain to be irreducible (for a unique, strictly positive stationary distribution) and aperiodic (the internal averaging step used by "spectral-top-down" to re-estimate a working stationary distribution as macro-states are merged assumes convergence, which is not guaranteed for a periodic chain). Use lazyChain first to remove periodicity if needed.

Value

A named list:

partition A named list of character vectors giving the original state names belonging to each macro-state, suitable for passing to lump or is.lumpable. Unlike PyDTMC, which only labels the reduced chain's states generically (e.g. "ASBU1"), this traces every macro-state back to the original states it stands for.

aggregatedChain The reduced markovchain object, row-stochastic, with states named after partition.

klDivergence The Kullback-Leibler divergence rate (in bits) between object and the lifted aggregatedChain. It is zero (up to rounding) when every source state splits its outgoing probability among a destination macro-state's members in the same proportions – those of the stationary distribution restricted to that macro-state – regardless of the source; this is *stronger* than the Kemeny-Snell strong lumpability checked by is.lumpable, which only requires the macro-to-macro *totals* to agree across sources (see Details).

method The method actually used, after resolving "adaptive".

k The number of macro-states actually used, after resolving an automatic NULL.

References

Deng, K., Mehta, P. G. and Meyn, S. P. (2011). Optimal Kullback-Leibler Aggregation via Spectral Theory of Markov Chains. *IEEE Transactions on Automatic Control*, 56(12). doi:10.1109/TAC.2011.2141350

Fill, J. A. (1991). Eigenvalue bounds on convergence to stationarity for nonreversible Markov chains, with an application to the exclusion process. *The Annals of Applied Probability*, 1(1). doi:10.1214/aoap/1177005981

See Also

autoLump, lump, is.lumpable, closestReversible

Examples

```
# A chain aggregated into two macro-states {a,b}/{c,d} where, in addition
# to being strongly lumpable, "a" and "b" also split their probability
# *within* each destination block identically (0.1/0.1 and 0.4/0.4): this
# stronger property is what makes the divergence exactly zero (see
# Details for a lumpable-but-nonzero counterexample).
statesNames <- c("a", "b", "c", "d")
P <- matrix(c(0.1, 0.1, 0.4, 0.4,
              0.1, 0.1, 0.4, 0.4,
              0.3, 0.3, 0.2, 0.2,
```

```
              0.3, 0.3, 0.2, 0.2), byrow = TRUE, nrow = 4,
            dimnames = list(statesNames, statesNames))
mc <- new("markovchain", states = statesNames, transitionMatrix = P)

result <- aggregateStates(mc, k = 2)
result$partition
result$klDivergence # zero up to rounding
```

`assessIndependence` *Test independence of consecutive states of an empirical sequence*

Description

Tests the null hypothesis that consecutive observations are independent, $P(X_{t+1} = j \mid X_t = i) = P(X_{t+1} = j)$ for all i, j, against the alternative of a first-order Markov chain. This is the classical test of Anderson and Goodman (1957): a chi-squared (or likelihood-ratio) test of independence applied to the table of observed one-step transition counts, whose rows are the state at time t and whose columns are the state at time $t + 1$.

Usage

```
assessIndependence(sequence, method = c("Pearson", "G"), verbose = TRUE)
```

Arguments

`sequence`	An empirical sequence of states (at least three observations, without missing values).
`method`	Test statistic: `"Pearson"` (default) for the chi-squared statistic or `"G"` for the likelihood-ratio statistic.
`verbose`	Should test results be printed?

Details

Only states actually observed as a departure state (rows) or as an arrival state (columns) contribute to the degrees of freedom, which are $(r - 1)(c - 1)$ for r such rows and c such columns. When r or c is 1 (for instance a constant sequence) the test is not defined: the degrees of freedom are 0 and the p-value is `NA`.

The statistic is asymptotic and, as for any chi-squared test on a table of counts, unreliable when many expected counts are small (say below 5). Successive transitions overlap (each observation is the arrival state of one transition and the departure state of the next); this is the standard treatment of the Anderson-Goodman test and is asymptotically valid under the null hypothesis.

Value

An `htest` object, returned invisibly, with the additional components `observed` (transition counts, rows are departure states) and `expected` (counts expected under independence).

References

Anderson, T. W. and Goodman, L. A. (1957). Statistical inference about Markov chains. *The Annals of Mathematical Statistics*, 28(1), 89–110.

See Also

verifyMarkovProperty, assessOrder, assessStationarity

Other statisticalTests: verifyMarkovProperty()

Examples

```
# an independent sequence: the test should not reject
set.seed(1)
iid <- sample(c("a", "b", "c"), 500, replace = TRUE)
assessIndependence(iid)

# a strongly dependent (Markov) sequence: the test rejects
mc <- new("markovchain", states = c("a", "b"),
          transitionMatrix = matrix(c(0.9, 0.1, 0.2, 0.8), nrow = 2, byrow = TRUE))
dep <- rmarkovchain(500, mc, t0 = "a")
assessIndependence(dep, method = "G")
```

---

autoLump *Automatically aggregate a Markov chain by spectral clustering*

Description

Finds an approximate partition by clustering the leading right eigenvectors of the transition matrix, then returns the forced lumping over that partition. This is a heuristic for approximate lumping/metastable aggregation, not a proof of exact lumpability.

Usage

```
autoLump(object, k)

## S4 method for signature 'markovchain'
autoLump(object, k)
```

Arguments

object	A markovchain object.
k	Number of macro-states to discover.

Value

A list with partition and lumped_chain.

`autoplot.markovchain` *Plot a Markov chain with ggplot2*

Description

Creates a ggplot2 representation of a discrete-time Markov chain. Nodes are states and directed edges represent transitions with positive probability. Communicating classes are shown with different node fills.

Usage

```
autoplot.markovchain(
  object,
  threshold = 0,
  show_probabilities = TRUE,
  digits = 2,
  node_size = 6,
  edge_width = 1,
  type = c("graph", "eigenvalues", "flow", "comparison"),
  steps = 20,
  initial = NULL,
  other = NULL,
  what = c("transition", "stationary"),
  ...
)
```

Arguments

`object`	An object of class 'markovchain'.
`threshold`	Minimum transition probability to display. Defaults to 0.
`show_probabilities`	Logical; whether to label edges with transition probabilities. Defaults to 'TRUE'.
`digits`	Number of digits used to format transition probabilities.
`node_size`	Size of state nodes in the ggplot2 plot.
`edge_width`	Minimum width multiplier for transition edges.
`type`	Type of plot: `"graph"` (default) draws the transition graph; `"eigenvalues"` draws the eigenvalues of the transition matrix in the complex plane together with the unit circle; `"flow"` draws the evolution of the distribution over the states, as computed by `redistribute`; `"comparison"` compares `object` with the chains in `other`.
`steps`	Number of steps of the `"flow"` plot. Defaults to 20. Ignored for the other types.
`initial`	Initial distribution of the `"flow"` plot, see `redistribute`. Defaults to the uniform distribution. Ignored for the other types.

other	For type = "comparison": a markovchain object or a (possibly named) list of them, to be compared with object. All chains must be defined on the same set of states, which are matched by name, so their order may differ. Ignored for the other types.
what	For type = "comparison": "transition" (default) draws the transition matrices side by side as heatmaps on a common probability scale; "stationary" compares the stationary distributions as grouped bars (every chain must then be irreducible). Ignored for the other types.
...	Currently unused, reserved for future extensions.

Value

A ggplot object.

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  weather <- matrix(c(0.7, 0.2, 0.1,
                      0.3, 0.4, 0.3,
                      0.2, 0.45, 0.35),
                    nrow = 3, byrow = TRUE,
                    dimnames = list(c("sunny", "cloudy", "rain"),
                                    c("sunny", "cloudy", "rain")))
  mc <- new("markovchain", states = rownames(weather),
            transitionMatrix = weather, name = "Weather")
  ggplot2::autoplot(mc)
  ggplot2::autoplot(mc, type = "eigenvalues")
  ggplot2::autoplot(mc, type = "flow", steps = 10, initial = "rain")

  # compare with a "stickier" version of the same chain
  sticky <- lazyChain(mc, alpha = 0.5)
  sticky@name <- "Lazy weather"
  ggplot2::autoplot(mc, type = "comparison", other = sticky)
  ggplot2::autoplot(mc, type = "comparison", other = sticky,
                    what = "stationary")
}
```

---

birthDeath *Build a birth-death Markov chain*

Description

Constructs a markovchain object for a birth-death process: a chain on linearly ordered states $1, 2, \ldots, n$ that, from any state, can only move to itself or to an immediately adjacent state.

Usage

```
birthDeath(p, q, states = NULL)
```

Arguments

p	A numeric vector of length $n-1$: p[i] is the "birth" probability of moving from state i up to state $i+1$.
q	A numeric vector of length $n-1$, the same length as p: q[i] is the "death" probability of moving from state $i+1$ down to state i.
states	An optional character vector of n = length(p) + 1 state names. Defaults to as.character(1:n).

Details

Why p and q have length $n-1$, not n. Every birth-death transition is a move between two adjacent states, and there are exactly $n-1$ adjacent pairs among n linearly ordered states: p[i]/q[i] unambiguously describe the pair $(i, i+1)$. This sidesteps a common source of confusion in this construction, namely what to do with a stray "birth probability of the top state" or "death probability of the bottom state" – quantities that do not correspond to any actual transition, since there is no state $n+1$ to be born into or state 0 to die into. Some implementations accept two length-n vectors and quietly renormalize every row so that any such leftover probability mass is redistributed among the transitions that do exist; birthDeath() instead makes the $n-1$ genuine transition probabilities the only inputs, so there is no leftover mass to (silently) dispose of in the first place.

Every row's diagonal entry is determined by the requirement that the row sums to 1, so p and q alone fully determine P: no separate "staying" probability is accepted or needed. p+q is allowed to reach 1 for an interior state (no staying probability there), but each element of p and q must itself lie in $[0, 1]$ and p[i]+q[i] for the shared index i need not be checked against 1 the way it would for a single state's own two probabilities, since p[i] leaves state i while q[i] leaves state $i+1$: the actual per-state constraint, $p_i + q_{i-1} \leq 1$, is checked directly on the assembled diagonal.

The two boundary states 1 and n are reflecting only in the weak sense that no birth/death carries them outside $\{1, \ldots, n\}$ – they still generally have a positive probability of staying put ($1 - p_1$ and $1 - q_{n-1}$ respectively) rather than being forced to bounce back, unlike gamblersRuin's absorbing ends or toBoundedChain's explicit reflecting condition, which can be applied afterwards to force deterministic bouncing or absorption at the ends of any chain, including one built here.

Value

A new, row-stochastic markovchain object on n states, with transition matrix

$$P_{ii} = 1 - p_i - q_{i-1}, \quad P_{i,i+1} = p_i, \quad P_{i,i-1} = q_{i-1}$$

(boundary terms q_0 and p_n are understood to not exist, i.e. $P_{11} = 1 - p_1$ and $P_{nn} = 1 - q_{n-1}$).

See Also

gamblersRuin, toBoundedChain, urnModel

Examples

```
# A simple 4-state birth-death chain with constant birth/death rates.
bd <- birthDeath(p = c(0.3, 0.4, 0.5), q = c(0.2, 0.3, 0.1))
bd
```

```
rowSums(bd@transitionMatrix)
```

`blanden` *Mobility between income quartiles*

Description

This table show mobility between income quartiles for father and sons for the 1970 cohort born

Usage

```
data(blanden)
```

Format

An object of class `table` with 4 rows and 4 columns.

Details

The rows represent fathers' income quartile when the son is aged 16, whilst the columns represent sons' income quartiles when he is aged 30 (in 2000).

Source

Personal reworking

References

Jo Blanden, Paul Gregg and Stephen Machin, Intergenerational Mobility in Europe and North America, Center for Economic Performances (2005)

Examples

```
data(blanden)
mobilityMc<-as(blanden, "markovchain")
```

`closestReversible` *Closest reversible approximation of a Markov chain*

Description

Finds the Markov chain closest to a given one among those that are reversible with respect to a fixed stationary distribution.

Usage

```
closestReversible(
  object,
  stationaryDistribution = NULL,
  tolerance = sqrt(.Machine$double.eps)
)

## S4 method for signature 'markovchain'
closestReversible(
  object,
  stationaryDistribution = NULL,
  tolerance = sqrt(.Machine$double.eps)
)
```

Arguments

`object`	A `markovchain` object representing a finite, irreducible discrete-time Markov chain.
`stationaryDistribution`	Optional numeric vector giving the stationary distribution π to make the result reversible with respect to. It must have one strictly positive entry per state, sum to one (it is normalized if it does not), and be stationary for `object`, i.e. satisfy $\pi P = \pi$; otherwise an error is raised. The default, `NULL`, uses the chain's own unique stationary distribution from `steadyStates`.
`tolerance`	A single finite non-negative number, used when checking that a supplied `stationaryDistribution` really is stationary. It is a numerical-noise tolerance, not a modelling one.

Details

For a row-stochastic transition matrix P with stationary distribution π, define the *time reversal* P^* by

$$P^*_{ij} = \frac{\pi_j P_{ji}}{\pi_i}.$$

P^* is the transition matrix of the same chain run backwards in time, and P is reversible exactly when $P = P^*$. The approximation returned is the *additive reversibilization*

$$R = \tfrac{1}{2}\left(P + P^*\right),$$

which is stochastic, non-negative, and reversible with respect to the same π (see Details).

In what sense is this the closest chain? Work in the space $\ell^2(\pi)$ of functions on the states with inner product $\langle f, g\rangle_\pi = \sum_i \pi_i f_i g_i$. Reversible chains are exactly the self-adjoint operators on that space, and they form a linear subspace. The matching Hilbert-Schmidt inner product on operators is

$$\langle A, B\rangle_\pi = \sum_{i,j} \frac{\pi_i}{\pi_j} A_{ij} B_{ij}, \qquad \|A\|_\pi^2 = \sum_{i,j} \frac{\pi_i}{\pi_j} A_{ij}^2,$$

and $A \mapsto A^*$ is an isometric involution for it. The orthogonal projection onto the fixed points of such an involution is the average of a point and its image, so $R = (P + P^*)/2$ is the closest π-reversible matrix to P in $\|\cdot\|_\pi$. The stochastic and non-negativity constraints come for free: P^* has row sums $\sum_j \pi_j P_{ji}/\pi_i = (\pi P)_i/\pi_i = 1$ because π is stationary, and both P and P^* are non-negative, so the minimizer over the subspace already lies in the set of transition matrices and no constrained optimization is needed.

What this function does not do. The minimization is over reversible chains *with π held fixed*, in the π-weighted norm above. Two related problems are different and are not solved here:

- Minimizing the *plain* Frobenius distance $\|P - R\|_F$ with π fixed. The involution $A \mapsto A^*$ is not an isometry for that norm, so R is generally not its minimizer; `frobeniusDistance` is reported only as a descriptive figure.
- Letting the stationary distribution *vary*, i.e. finding the reversible chain nearest to P over all choices of π. That is a genuinely harder constrained optimization problem, studied by Nielsen and Weber (2015), and it needs a numerical optimizer rather than a closed form. If you need it, supply candidate distributions through `stationaryDistribution` and compare `distance` values, or use a dedicated implementation.

Properties worth knowing. R has the same stationary distribution π as P, and it preserves the support pattern in the symmetrized sense: $R_{ij} > 0$ whenever $P_{ij} > 0$ or $P_{ji} > 0$. It may therefore allow transitions the original chain forbids, which is inherent to making a chain reversible rather than a defect of this construction. If `object` is already reversible, $R = P$ and `distance` is zero (up to rounding). Fill (1991) introduces this construction, alongside the *multiplicative* reversibilization PP^*, which is a different object and is not computed here.

Only irreducibility is required, not aperiodicity. Irreducibility guarantees both a unique π and $\pi_i > 0$ for every state, so the division defining P^* is always safe.

The implementation calls `steadyStates` at most once and is then $O(n^2)$ in time and memory for a dense n-state transition matrix; no eigendecomposition or optimization is involved.

Value

A named list with four elements:

`chain` The approximating `markovchain` object R, with the same states and the same row/column-stochastic storage convention as `object`.

`stationaryDistribution` The π used, as a named numeric vector.

`distance` The distance $\|P - R\|_\pi$ actually minimized (see Details).

`frobeniusDistance` The plain Frobenius distance $\|P - R\|_F$, reported for convenience. It is *not* the quantity being minimized.

References

Fill, J. A. (1991). Eigenvalue bounds on convergence to stationarity for nonreversible Markov chains, with an application to the exclusion process. *The Annals of Applied Probability*, 1(1). doi:10.1214/aoap/1177005981

Nielsen, A. and Weber, M. (2015). Computing the nearest reversible Markov chain. *Numerical Linear Algebra with Applications*, 22. doi:10.1002/nla.1967

See Also

`is.reversible`, `steadyStates`, `is.irreducible`

Examples

```
# A directed 3-cycle is as far from reversible as a chain gets: it only
# ever moves one way round. Its closest reversible approximation is the
# undirected random walk on the same triangle.
statesNames <- c("a", "b", "c")
cycle3 <- new("markovchain", states = statesNames,
  transitionMatrix = matrix(c(0, 1, 0,
                              0, 0, 1,
                              1, 0, 0), byrow = TRUE, nrow = 3,
                            dimnames = list(statesNames, statesNames)))
is.reversible(cycle3)

approximation <- closestReversible(cycle3)
approximation$chain
is.reversible(approximation$chain)
approximation$distance
```

`committorAB` *Calculates committor of a markovchain object with respect to set A, B*

Description

Returns the probability of hitting states rom set A before set B with different initial states

Usage

```
committorAB(object,A,B,p)
```

Arguments

`object`	a markovchain class object
`A`	a set of states
`B`	a set of states
`p`	initial state (default value : 1)

Details

The function solves a system of linear equations to calculate probaility that the process hits a state from set A before any state from set B

Value

Return a vector of probabilities in case initial state is not provided else returns a number

Examples

```
transMatr <- matrix(c(0,0,0,1,0.5,
                      0.5,0,0,0,0,
                      0.5,0,0,0,0,
                      0,0.2,0.4,0,0,
                      0,0.8,0.6,0,0.5),
                      nrow = 5)
object <- new("markovchain", states=c("a","b","c","d","e"),transitionMatrix=transMatr)
committorAB(object,c(5),c(3))
```

---

conditionalDistribution

conditionalDistribution *of a Markov Chain*

Description

It extracts the conditional distribution of the subsequent state, given current state.

Usage

```
conditionalDistribution(object, state)
```

Arguments

object	A markovchain object.
state	Subsequent state.

Value

A named probability vector

Author(s)

Giorgio Spedicato, Deepak Yadav

References

A First Course in Probability (8th Edition), Sheldon Ross, Prentice Hall 2010

See Also

markovchain

Examples

```
# define a markov chain
statesNames <- c("a", "b", "c")
markovB <- new("markovchain", states = statesNames, transitionMatrix =
               matrix(c(0.2, 0.5, 0.3, 0, 1, 0, 0.1, 0.8, 0.1),nrow = 3,
                      byrow = TRUE, dimnames = list(statesNames, statesNames)))

conditionalDistribution(markovB, "b")
```

---

craigsendi *CD4 cells counts on HIV Infects between zero and six month*

Description

This is the table shown in Craig and Sendi paper showing zero and six month CD4 cells count in six brakets

Usage

```
data(craigsendi)
```

Format

The format is: table [1:3, 1:3] 682 154 19 33 64 19 25 47 43 - attr(*, "dimnames")=List of 2 ..$: chr [1:3] "0-49" "50-74" "75-UP" ..$: chr [1:3] "0-49" "50-74" "75-UP"

Details

Rows represent counts at the beginning, cols represent counts after six months.

Source

Estimation of the transition matrix of a discrete time Markov chain, Bruce A. Craig and Peter P. Sendi, Health Economics 11, 2002.

References

see source

Examples

```
data(craigsendi)
csMc<-as(craigsendi, "markovchain")
steadyStates(csMc)
```

ctmc-class *Continuous time Markov Chains class*

Description

The S4 class that describes ctmc (continuous time Markov chain) objects.

Slots

`states` Name of the states. Must be the same of `colnames` and `rownames` of the generator matrix

`byrow` TRUE or FALSE. Indicates whether the given matrix is stochastic by rows or by columns

`generator` Square generator matrix

`name` Optional character name of the Markov chain

Methods

dim `signature(x = "ctmc")`: method to get the size

initialize `signature(.Object = "ctmc")`: initialize method

states `signature(object = "ctmc")`: states method.

steadyStates `signature(object = "ctmc")`: method to get the steady state vector.

plot `signature(x = "ctmc", y = "missing")`: plot method for ctmc objects

Note

1. ctmc classes are written using S4 classes
2. Validation method is used to assess whether either columns or rows totals to zero. Rounding is used up to 5th decimal. If state names are not properly defined for a generator `matrix`, coercing to ctmc object leads to overriding states name with artificial "s1", "s2", ... sequence

References

Introduction to Stochastic Processes with Applications in the Biosciences (2013), David F. Anderson, University of Wisconsin at Madison. Sai Bhargav Yalamanchi, Giorgio Spedicato

See Also

generatorToTransitionMatrix,rctmc

Examples

```
energyStates <- c("sigma", "sigma_star")
byRow <- TRUE
gen <- matrix(data = c(-3, 3,
                       1, -1), nrow = 2,
              byrow = byRow, dimnames = list(energyStates, energyStates))
molecularCTMC <- new("ctmc", states = energyStates,
```

```
                    byrow = byRow, generator = gen,
                    name = "Molecular Transition Model")
                    steadyStates(molecularCTMC)
## Not run: plot(molecularCTMC)
```

---

ctmcFit *Function to fit a CTMC*

Description

This function fits the underlying CTMC give the state transition data and the transition times using the maximum likelihood method (MLE)

Usage

```
ctmcFit(data, byrow = TRUE, name = "", confidencelevel = 0.95)
```

Arguments

data	It is a list of two elements. The first element is a character vector denoting the states. The second is a numeric vector denoting the corresponding transition times.
byrow	Determines if the output transition probabilities of the underlying embedded DTMC are by row.
name	Optional name for the CTMC.
confidencelevel	Confidence level for the confidence interval construnction.

Details

Note that in data, there must exist an element wise corresponding between the two elements of the list and that data[[2]][1] is always 0.

Value

It returns a list containing the CTMC object and the confidence intervals.

Author(s)

Sai Bhargav Yalamanchi

References

Continuous Time Markov Chains (vignette), Sai Bhargav Yalamanchi, Giorgio Alfredo Spedicato 2015

See Also

rctmc

Examples

```
data <- list(c("a", "b", "c", "a", "b", "a", "c", "b", "c"), c(0, 0.8, 2.1, 2.4, 4, 5, 5.9, 8.2, 9))
ctmcFit(data)
```

`dirichletChain` *Markov chain from a Dirichlet process*

Description

Generates a Markov chain whose rows are drawn from a truncated Dirichlet process with the stick-breaking (GEM) construction, as `MarkovChain.dirichlet_process()` of PyDTMC does.

Usage

```
dirichletChain(
  n,
  diffusion,
  states = NULL,
  diagonalBias = NULL,
  shiftConcentration = FALSE,
  byrow = TRUE,
  seed = NULL,
  name = "Dirichlet process chain"
)
```

Arguments

`n`	The number of states, at least 2. It can be omitted when `states` is given.
`diffusion`	The concentration parameter $\alpha > 0$ of the Dirichlet process. Small values concentrate the probability of each row on its first states; large values spread it more evenly.
`states`	An optional character vector of `n` state names. Defaults to `as.character(1:n)`.
`diagonalBias`	An optional positive number β. When given, a draw from $\mathrm{Beta}(\beta, 1)$ is added to each diagonal entry before the row is renormalised, which makes the chain more likely to stay where it is; larger values give a stronger bias.
`shiftConcentration`	If `TRUE`, the columns are reversed, so that the probability concentrates on the last states instead of the first ones.
`byrow`	Whether the transition matrix of the result is stored by rows (the default) or by columns.
`seed`	An optional whole number, as in `randomMarkovChain`.
`name`	The `name` slot of the result.

Details

For each row, $b_1, \ldots, b_n$ are independent $\mathrm{Beta}(1, \alpha)$ draws and the weights are

$$w_j = b_j \prod_{k<j} (1 - b_k),$$

normalised to sum to one (the truncation at n states leaves out the mass $\prod_k (1 - b_k)$). PyDTMC only accepts whole values of α between 1 and n; any positive value is accepted here, since the construction is defined for every $\alpha > 0$.

Value

A `markovchain` object with `n` states.

References

Sethuraman, J. (1994). A constructive definition of Dirichlet priors. *Statistica Sinica*, 4(2), 639-650.

See Also

`randomMarkovChain`

Examples

```
dirichletChain(5, diffusion = 2, seed = 1)
# a chain that tends to stay in its current state
dirichletChain(5, diffusion = 2, diagonalBias = 5, seed = 1)
```

`entropyRate` *Entropy rate of a Markov chain*

Description

Computes the entropy rate of a finite, irreducible discrete-time Markov chain from its stationary distribution and transition matrix.

Usage

```
entropyRate(object, base = 2)

## S4 method for signature 'markovchain'
entropyRate(object, base = 2)
```

Arguments

`object`	A `markovchain` object representing a finite, irreducible discrete-time Markov chain.
`base`	A finite numeric scalar strictly greater than one. The default, `2`, returns entropy in bits per transition. Use `exp(1)` for nats per transition.

Details

For a row-stochastic transition matrix P and stationary distribution π, the entropy rate is

$$H = -\sum_i \pi_i \sum_j p_{ij} \log_b(p_{ij}),$$

with zero-probability transitions contributing zero by continuity.

For a stationary first-order Markov chain, the entropy rate equals the conditional entropy $H(X_{t+1} \mid X_t)$. Irreducibility guarantees a unique stationary distribution; aperiodicity is not required.

Reducible chains may admit multiple stationary distributions and therefore different entropy rates. This method rejects them rather than silently selecting one stationary distribution.

Transitions with probability zero are ignored, implementing the standard convention $0 \log(0) = 0$ without evaluating `log(0)`.

The stationary-distribution computation dominates the running time. Once the stationary distribution is available, evaluating the entropy rate takes $O(n^2)$ time and $O(n^2)$ temporary memory for a dense n-state transition matrix.

Value

A non-negative numeric scalar containing the entropy rate in units determined by `base`.

References

Cover, T. M. and Thomas, J. A. (2006). *Elements of Information Theory*, 2nd edition. Wiley.

Strelioff, C. C., Crutchfield, J. P. and Huebler, A. W. (2007). Inferring Markov chains: Bayesian estimation, model comparison, entropy rate, and out-of-class modeling. *Physical Review E*, 76, 011106.

See Also

`steadyStates`, `is.irreducible`

Examples

```
statesNames <- c("a", "b")
mc <- new("markovchain",
  states = statesNames,
  transitionMatrix = matrix(c(0.7, 0.3, 0.1, 0.9),
    byrow = TRUE, nrow = 2,
    dimnames = list(statesNames, statesNames)))
entropyRate(mc)
entropyRate(mc, base = exp(1))
```

`expectedRewards` *Expected Rewards for a markovchain*

Description

Given a markovchain object and reward values for every state, function calculates expected reward value after n steps.

Usage

```
expectedRewards(markovchain,n,rewards)
```

Arguments

`markovchain`	the markovchain-class object
`n`	no of steps of the process
`rewards`	vector depicting rewards coressponding to states

Details

the function uses a dynamic programming approach to solve a recursive equation described in reference.

Value

returns a vector of expected rewards for different initial states

Author(s)

Vandit Jain

References

Stochastic Processes: Theory for Applications, Robert G. Gallager, Cambridge University Press

Examples

```
transMatr<-matrix(c(0.99,0.01,0.01,0.99),nrow=2,byrow=TRUE)
simpleMc<-new("markovchain", states=c("a","b"),
              transitionMatrix=transMatr)
expectedRewards(simpleMc,1,c(0,1))
```

expectedRewardsBeforeHittingA

Expected first passage Rewards for a set of states in a markovchain

Description

Given a markovchain object and reward values for every state, function calculates expected reward value for a set A of states after n steps.

Usage

```
expectedRewardsBeforeHittingA(markovchain, A, state, rewards, n)
```

Arguments

markovchain	the markovchain-class object
A	set of states for first passage expected reward
state	initial state
rewards	vector depicting rewards coressponding to states
n	no of steps of the process

Details

The function returns the value of expected first passage rewards given rewards coressponding to every state, an initial state and number of steps.

Value

returns a expected reward (numerical value) as described above

Author(s)

Sai Bhargav Yalamanchi, Vandit Jain

ExpectedTime

Returns expected hitting time from state i to state j

Description

Returns expected hitting time from state i to state j

Usage

```
ExpectedTime(C,i,j,useRCpp)
```

Arguments

C	A CTMC S4 object
i	Initial state i
j	Final state j
useRCpp	logical whether to use Rcpp

Details

According to the theorem, holding times for all states except j should be greater than 0.

Value

A numerical value that returns expected hitting times from i to j

Author(s)

Vandit Jain

References

Markovchains, J. R. Norris, Cambridge University Press

Examples

```
states <- c("a","b","c","d")
byRow <- TRUE
gen <- matrix(data = c(-1, 1/2, 1/2, 0, 1/4, -1/2, 0, 1/4, 1/6, 0, -1/3, 1/6, 0, 0, 0, 0),
nrow = 4,byrow = byRow, dimnames = list(states,states))
ctmc <- new("ctmc",states = states, byrow = byRow, generator = gen, name = "testctmc")
ExpectedTime(ctmc,1,4,TRUE)
```

---

firstPassage *First passage across states*

Description

This function compute the first passage probability in states

Usage

```
firstPassage(object, state, n)
```

Arguments

object	A markovchain object
state	Initial state
n	Number of rows on which compute the distribution

Details

Based on Feres' Matlab listings

Value

A matrix of size 1:n x number of states showing the probability of the first time of passage in states to be exactly the number in the row.

Author(s)

Giorgio Spedicato

References

Renaldo Feres, Notes for Math 450 Matlab listings for Markov chains

See Also

conditionalDistribution

Examples

```
simpleMc <- new("markovchain", states = c("a", "b"),
                transitionMatrix = matrix(c(0.4, 0.6, .3, .7),
                                          nrow = 2, byrow = TRUE))
firstPassage(simpleMc, "b", 20)
```

---

firstPassageMultiple *function to calculate first passage probabilities*

Description

The function calculates first passage probability for a subset of states given an initial state.

Usage

```
firstPassageMultiple(object, state, set, n)
```

Arguments

object	a markovchain-class object
state	intital state of the process (charactervector)
set	set of states A, first passage of which is to be calculated
n	Number of rows on which compute the distribution

Value

A vector of size n showing the first time proabilities

Author(s)

Vandit Jain

References

Renaldo Feres, Notes for Math 450 Matlab listings for Markov chains; MIT OCW, course - 6.262, Discrete Stochastic Processes, course-notes, chap -05

See Also

firstPassage

Examples

```
statesNames <- c("a", "b", "c")
markovB <- new("markovchain", states = statesNames, transitionMatrix =
matrix(c(0.2, 0.5, 0.3,
         0, 1, 0,
         0.1, 0.8, 0.1), nrow = 3, byrow = TRUE,
       dimnames = list(statesNames, statesNames)
))
firstPassageMultiple(markovB,"a",c("b","c"),4)
```

`fitHigherOrder` *Functions to fit a higher order Markov chain*

Description

Given a sequence of states arising from a stationary state, it fits the underlying Markov chain distribution with higher order.

Usage

```
fitHigherOrder(sequence, order = 2, method = c("lsq", "mle"))
seq2freqProb(sequence)
seq2matHigh(sequence, order)
```

Arguments

`sequence`	A character list.
`order`	Markov chain order
`method`	How the weights λ are estimated: `"lsq"` (default) or `"mle"`, see Details.

Details

The fitted model expresses the distribution of the next state as the mixture $\sum_{i=1}^{k} \lambda_i Q_i x_{t-i}$ of the empirical lag-i transition matrices Q_i (see `seq2matHigh`), with weights $\lambda_i \geq 0$ summing to one. The matrices Q_i are the same for both methods; only the weights differ.

`method = "lsq"` (the default, and the only behaviour before the argument existed) chooses λ to minimize the squared distance between the stationary distribution and its image under the mixture, as in Ching et al.; it needs the **Rsolnp** package and returns `NULL` with a message if it is unavailable. This criterion is weak: each lag matrix maps the empirical distribution onto itself up to end effects, $Q_i X \approx X$ with an error of order i/n, so the objective is nearly flat in λ and the weights it returns can be unstable from one sample to another. It is kept as the default for backward compatibility; `method = "mle"` is preferable when the weights are interpreted or models are compared by likelihood.

`method = "mle"` chooses λ to maximize the log-likelihood $\sum_{t=k+1}^{n} \log \sum_i \lambda_i Q_i[x_t, x_{t-i}]$ of the observations a model of order k can predict. For fixed Q_i the problem is concave, so the maximum is global, and it is solved by the EM algorithm for mixture weights, without **Rsolnp**. The weights are therefore those that give the highest value of `higherOrderLogLik` for the same observations. They are maximum likelihood estimates *conditional on* the empirical matrices Q_i, which are not re-estimated: this is not the maximum likelihood estimator of the mixture with free matrices, in which the weights are in general not identifiable (a common distribution can be moved from $\lambda_j Q_j$ to $\lambda_i Q_i$ without changing any transition probability), so the weights should not be read as the relative importance of the lags beyond this conditional sense. Note that this is not the mixture transition distribution model of Raftery (1985), in which a single matrix is shared by all lags and is estimated together with the weights; that model is fitted by `fitMTD`.

Value

A list containing lambda, Q, and X.

Author(s)

Giorgio Spedicato, Tae Seung Kang

References

Ching, W. K., Huang, X., Ng, M. K., & Siu, T. K. (2013). Higher-order markov chains. In Markov Chains (pp. 141-176). Springer US.

Ching, W. K., Ng, M. K., & Fung, E. S. (2008). Higher-order multivariate Markov chains and their applications. Linear Algebra and its Applications, 428(2), 492-507.

Raftery, A. E. (1985). A model for high-order Markov chains. Journal of the Royal Statistical Society, Series B, 47(3), 528-539.

Examples

```
sequence<-c("a", "a", "b", "b", "a", "c", "b", "a", "b", "c", "a", "b",
            "c", "a", "b", "c", "a", "b", "a", "b")
fitHigherOrder(sequence)
# weights by maximum likelihood (no Rsolnp needed)
fit <- fitHigherOrder(sequence, order = 2, method = "mle")
fit$lambda
```

```
higherOrderLogLik(sequence, fit)$logLik
```

`fitHighOrderMultivarMC` *Function to fit Higher Order Multivariate Markov chain*

Description

Given a matrix of categorical sequences it fits Higher Order Multivariate Markov chain.

Usage

```
fitHighOrderMultivarMC(seqMat, order = 2, Norm = 2)
```

Arguments

`seqMat`	a matrix or a data frame where each column is a categorical sequence
`order`	Multivariate Markov chain order. Default is 2.
`Norm`	Norm to be used. Default is 2.

Value

an hommc object

Author(s)

Giorgio Spedicato, Deepak Yadav

References

W.-K. Ching et al. / Linear Algebra and its Applications

Examples

```
data <- matrix(c('2', '1', '3', '3', '4', '3', '2', '1', '3', '3', '2', '1',
                 c('2', '4', '4', '4', '4', '2', '3', '3', '1', '4', '3', '3')),
                 ncol = 2, byrow = FALSE)

fitHighOrderMultivarMC(data, order = 2, Norm = 2)
```

`fitMTD` *Fit a mixture transition distribution (MTD) model*

Description

Estimates by maximum likelihood the mixture transition distribution model of Raftery (1985) for a high-order Markov chain: the probability of the next state is a weighted mixture of the contributions of the last `order` states, all governed by the same transition matrix.

Usage

```
fitMTD(
  sequence,
  order = 2,
  start = NULL,
  nstart = 1,
  tol = 1e-10,
  maxit = 10000L
)
```

Arguments

`sequence`	An empirical sequence of states (a character vector, or a vector coercible to character), without missing values.
`order`	Order of the model, a positive integer smaller than the length of the sequence.
`start`	Index of the first observation entering the likelihood, at least `order + 1` (the default).
`nstart`	`nstart - 1` is the number of random starting points of the EM algorithm added to the deterministic ones (see Details).
`tol`	Convergence tolerance on the relative change of the log-likelihood between two iterations.
`maxit`	Maximum number of EM iterations for each starting point.

Details

For an order k the model is

$$P(X_t = j \mid X_{t-1} = i_1, \ldots, X_{t-k} = i_k) = \sum_{g=1}^{k} \lambda_g \, q_{i_g j},$$

where $Q = (q_{ij})$ is an $r \times r$ transition matrix (rows are departure states) and the lag weights λ_g sum to one. It needs $r(r-1)+k-1$ parameters instead of the $r^k(r-1)$ of a fully parameterized Markov chain of order k, which makes high orders practicable (Raftery, 1985; Berchtold and Raftery, 2002). This is different from the model fitted by `fitHigherOrder`, which mixes a different empirical matrix for each lag and does not estimate them jointly with the weights.

The weights are constrained to be non-negative, as in most applications of the model; Raftery's original formulation also allows negative weights provided all the transition probabilities stay in $[0, 1]$, which is not supported here. Under this constraint the likelihood is maximized by the EM algorithm of Lebre and Bourguignon (2008), in which the latent variable is the lag that generated each observation; it is implemented in C++. Every iteration increases the likelihood, but the likelihood of the MTD model can have several local maxima (Berchtold, 2001). The models of order $1, \ldots,$ `order` are therefore fitted in turn on the same observations, and order k is started from equal weights with the transition matrix of all lags pooled, and from the fit of order $k-1$ extended with a zero and with a small positive weight for the new lag. The first of these extensions has the likelihood of order $k-1$, so the likelihood returned never decreases with the order, as it must for nested models. `nstart - 1` further random starting points can be added for the requested order; they are drawn at random, so call `set.seed` beforehand for reproducible results. The fit with the highest likelihood is returned.

The likelihood is conditional on the observations before `start`: it is the product of the transition probabilities of x_t for $t =$ `start`, $\ldots, n$. The default, `order + 1`, uses every observation an order-`order` model can predict. To compare models of different orders (or with the Markov chains of `fitHigherOrder` via `higherOrderLogLik`) use the same `start` for all, for instance $1+$ the largest order; Berchtold and Raftery (2002) condition on the first 14 observations, i.e. `start = 15`.

A state that never occurs in a conditioning position (for instance one observed only at the end of the sequence) does not enter the likelihood; its row of Q is not identified and is returned as uniform.

AIC and BIC use the number of free parameters $r(r-1)+k-1$, and BIC the number of observations entering the likelihood. Berchtold and Raftery (2002) do not count the elements of Q estimated as exactly zero; their BIC can be obtained by subtracting the number of those elements from `npar`.

Value

A list with components

`lambda`	the estimated lag weights, named `lag1`, `lag2`, ...
`estimate`	a `markovchain` object (by row) with the estimated transition matrix Q
`Q`	a list of `order` copies of Q stored by column (`Q[[g]][to, from]`), the layout returned by `fitHigherOrder`, so that the fit can be passed to `higherOrderLogLik`
`X`	the relative frequencies of the states in `sequence`
`logLikelihood`, `npar`, `AIC`, `BIC`, `nobs`	maximized log-likelihood, number of free parameters, information criteria and number of observations entering the likelihood
`order`, `start`	as used in the fit
`iterations`, `converged`	EM iterations and convergence flag of the returned fit
`model`	the string `"MTD"`

References

Raftery, A. E. (1985). A model for high-order Markov chains. Journal of the Royal Statistical Society, Series B, 47(3), 528-539.

Berchtold, A. (2001). Estimation in the mixture transition distribution model. Journal of Time Series Analysis, 22(4), 379-397.

Berchtold, A. and Raftery, A. E. (2002). The mixture transition distribution model for high-order Markov chains and non-Gaussian time series. Statistical Science, 17(3), 328-356.

Lebre, S. and Bourguignon, P.-Y. (2008). An EM algorithm for estimation in the mixture transition distribution model. Journal of Statistical Computation and Simulation, 78(1), 1-15.

See Also

fitHigherOrder, higherOrderLogLik, higherOrderPredict, markovchainFit

Examples

```
# hourly wind directions at Koeberg (Berchtold and Raftery, 2002)
wind <- read.csv(system.file("extdata", "koeberg_wind.csv",
                             package = "markovchain"))$state
fit <- fitMTD(wind, order = 2, start = 15)
fit$lambda
fit$estimate
c(logLik = fit$logLikelihood, BIC = fit$BIC)

# several starting points guard against local maxima
set.seed(1)
fitMTD(wind, order = 3, start = 15, nstart = 5)$logLikelihood
```

---

freq2Generator *Returns a generator matrix corresponding to frequency matrix*

Description

The function provides interface to calculate generator matrix corresponding to a frequency matrix and time taken

Usage

```
freq2Generator(P, t = 1, method = "QO", logmethod = "Eigen")
```

Arguments

P	relative frequency matrix
t	(default value = 1)
method	one among "QO"(Quasi optimaisation), "WA"(weighted adjustment), "DA"(diagonal adjustment)
logmethod	method for computation of matrx algorithm (by default : Eigen)

Value

returns a generator matix with same dimnames

References

E. Kreinin and M. Sidelnikova: Regularization Algorithms for Transition Matrices. Algo Research Quarterly 4(1):23-40, 2001

Examples

```
sample <- matrix(c(150,2,1,1,1,200,2,1,2,1,175,1,1,1,1,150),nrow = 4,byrow = TRUE)
sample_rel = rbind((sample/rowSums(sample))[1:dim(sample)[1]-1,],c(rep(0,dim(sample)[1]-1),1))
freq2Generator(sample_rel,1)

data(tm_abs)
tm_rel=rbind((tm_abs/rowSums(tm_abs))[1:7,],c(rep(0,7),1))
## Derive quasi optimization generator matrix estimate
freq2Generator(tm_rel,1)
```

`fundamentalMatrix` *Fundamental matrix of an absorbing Markov chain*

Description

Computes the fundamental matrix of a finite absorbing discrete-time Markov chain. If Q is the transition submatrix restricted to transient states, the fundamental matrix is

$$N = I + Q + Q^2 + \cdots = (I - Q)^{-1}.$$

The (i, j) entry is the expected number of visits to transient state j, including the initial visit when $i = j$, before absorption, when the chain starts in transient state i.

Usage

```
fundamentalMatrix(object)
```

Arguments

`object`	A `markovchain` object representing an absorbing Markov chain.

Details

For a finite absorbing chain, the state space can be reordered so that the transition matrix has canonical form with transient block Q and an absorbing block. The spectral radius of Q is less than one, so the Neumann series converges and $I - Q$ is nonsingular.

The fundamental matrix also gives the expected time to absorption through $t = N1$, where 1 is a vector of ones, and absorption probabilities through $B = NR$, where R contains transition probabilities from transient to absorbing states.

The function requires an absorbing Markov chain: at least one absorbing state must exist and every recurrent state must be absorbing. A chain with no transient states is a valid degenerate case and returns a 0-by-0 matrix.

Value

A numeric matrix containing the fundamental matrix, with transient state names as row and column names. If all states are absorbing, the result is a 0-by-0 matrix because there are no transient states.

References

Kemeny, J. G. and Snell, J. L. (1976). *Finite Markov Chains*. Springer.

Grinstead, C. M. and Snell, J. L. (1997). *Introduction to Probability*. American Mathematical Society.

See Also

absorbingStates, transientStates, meanAbsorptionTime, absorptionProbabilities

Examples

```
states <- c("a", "b", "absorbed")
mc <- new("markovchain", states = states,
  transitionMatrix = matrix(c(
    0.5, 0.4, 0.1,
    0.2, 0.6, 0.2,
    0,   0,   1
  ), nrow = 3, byrow = TRUE,
  dimnames = list(states, states)))

fundamentalMatrix(mc)
```

---

gamblersRuin *Build a gambler's ruin Markov chain*

Description

Constructs the classic gambler's ruin chain: a gambler with a fortune between `0` and `upperBound` wins each round (and gains one unit) with probability `prob`, otherwise loses one unit; play stops as soon as the fortune reaches `0` (ruin) or `upperBound` (the gambler's target).

Usage

```
gamblersRuin(upperBound, prob, states = NULL)
```

Arguments

`upperBound`	A single positive integer: the fortune at which the gambler stops (having won). The chain has `upperBound + 1` states, $0, 1, \ldots,$ `upperBound`.
`prob`	A single number in $[0, 1]$: the probability of winning an individual round (moving up by one unit) while the fortune is strictly between `0` and `upperBound`.
`states`	An optional character vector of `upperBound + 1` state names, in increasing order of fortune. Defaults to `as.character(0:upperBound)`.

Details

This is the special case of `birthDeath` with constant birth probability `prob` and constant death probability `1-prob` at every interior state, together forced to be *absorbing* rather than merely reflecting at the two ends – which is why it is provided as its own constructor rather than expressed purely in terms of `birthDeath()`, which cannot produce absorbing boundaries by itself (see `toBoundedChain` for turning any chain's ends absorbing or reflecting after construction).

With `prob != 0.5`, the classical ruin probability of reaching `0` before `upperBound`, starting from fortune i, is

$$P(\text{ruin} \mid X_0 = i) = \frac{\left(\frac{1-\texttt{prob}}{\texttt{prob}}\right)^{i} - \left(\frac{1-\texttt{prob}}{\texttt{prob}}\right)^{\texttt{upperBound}}}{1 - \left(\frac{1-\texttt{prob}}{\texttt{prob}}\right)^{\texttt{upperBound}}},$$

and $i/\texttt{upperBound}$ when `prob = 0.5`; this is a standard textbook result (see Norris (1998), Section 1.3) and is not itself computed by this function, but can be read off from `absorptionProbabilities` applied to the returned chain.

Value

A new, row-stochastic `markovchain` object with `upperBound + 1` states. States `"0"` and `as.character(upperBound)` are absorbing; every interior state i has $P_{i,i+1} = \texttt{prob}$ and $P_{i,i-1} = 1 - \texttt{prob}$.

References

Norris, J. R. (1998). *Markov Chains*. Cambridge University Press.

See Also

`birthDeath`, `absorptionProbabilities`, `toBoundedChain`

Examples

```
ruin <- gamblersRuin(upperBound = 5, prob = 0.4)
ruin
absorbingStates(ruin)
```

`generatorToTransitionMatrix` *Function to obtain the transition matrix from the generator*

Description

The transition matrix of the embedded DTMC is inferred from the CTMC's generator

Usage

```
generatorToTransitionMatrix(gen, byrow = TRUE)
```

Arguments

gen	The generator matrix
byrow	Flag to determine if rows (columns) sum to 0

Value

Returns the transition matrix.

Author(s)

Sai Bhargav Yalamanchi

References

Introduction to Stochastic Processes with Applications in the Biosciences (2013), David F. Anderson, University of Wisconsin at Madison

See Also

rctmc,ctmc-class

Examples

```
energyStates <- c("sigma", "sigma_star")
byRow <- TRUE
gen <- matrix(data = c(-3, 3, 1, -1), nrow = 2,
              byrow = byRow, dimnames = list(energyStates, energyStates))
generatorToTransitionMatrix(gen)
```

---

higherOrderLogLik *Log-likelihood, deviance and information criteria of a higher order Markov chain*

Description

Evaluates the log-likelihood of an empirical sequence under the higher order Markov chain returned by fitHigherOrder, and derives the deviance, AIC and BIC, so that models of different orders can be compared.

Usage

```
higherOrderLogLik(sequence, fit = NULL, order = 2, start = NULL)
```

Arguments

sequence	The empirical sequence of states, a character vector or a vector coercible to character (numbers and factors are matched to the states of the fit as character strings).
fit	The list returned by fitHigherOrder (or by fitMTD) for sequence. If NULL, fitHigherOrder(sequence, order) is computed.
order	Order of the model to fit when fit is NULL (ignored otherwise; the order is then length(fit$lambda)).
start	Index of the first observation included in the likelihood. Defaults to order + 1, the earliest observation a model of that order can predict.

Details

The fitted model is the mixture-transition-distribution model of Raftery (1985) in the form used by Ching et al.: the probability of moving to state x_t given the past is

$$P(x_t \mid x_{t-1}, \ldots, x_{t-k}) = \sum_{i=1}^{k} \lambda_i \, Q_i[x_t, x_{t-i}],$$

where Q_i is the lag-i transition matrix (see seq2matHigh) and k is the order. The log-likelihood is the sum of the logarithms of these probabilities over the observations $t =$ start, $\ldots$, T, and the deviance is -2 times the log-likelihood.

Two points matter when interpreting the output. First, fitHigherOrder chooses λ by default (method = "lsq") by least squares on the stationary distribution, not by maximum likelihood, so the value returned is the log-likelihood *of the fitted model*, not the maximum attainable one; with method = "mle" the weights maximize this log-likelihood for the observations a model of that order can predict. Second, a model of order k can only be evaluated from observation $k + 1$ onwards; to compare orders on exactly the same data set start to $1+$ the largest order compared, otherwise the models are evaluated on different numbers of observations and neither the log-likelihood nor the information criteria are comparable.

The number of parameters used for AIC and BIC is the dimension of the set of transition laws the model can represent, $(r-1)(1+k(r-1))$, with r the number of states: each of the k lag matrices has $r(r-1)$ free probabilities and there are $k-1$ free weights, but the weights of a mixture of lag matrices are not identifiable (a distribution common to all departure states can be moved from $\lambda_j Q_j$ to $\lambda_i Q_i$ without changing any transition probability), which removes $r(k-1)$ parameters from the naive count $k\,r(r-1)+(k-1)$: for each next state the transition probability is a sum of one term for each lag, a main-effects function of the k past states, with $1 + k(r-1)$ free coefficients. For $k = 1$ both counts are $r(r-1)$. For a fit returned by fitMTD, whose lags share a single matrix, it is $r(r-1)+(k-1)$.

Value

A list with components logLik, deviance, AIC, BIC, nobs (number of observations entering the likelihood), npar, order and start. The log-likelihood is -Inf if an observed transition has probability zero under the model (possible only for a sequence other than the one the model was fitted on).

References

Raftery, A. E. (1985). A model for high-order Markov chains. Journal of the Royal Statistical Society, Series B, 47(3), 528-539.

Ching, W. K., Huang, X., Ng, M. K., & Siu, T. K. (2013). Higher-order markov chains. In Markov Chains (pp. 141-176). Springer US.

See Also

`fitHigherOrder`, `fitMTD`, `higherOrderPredict`

Examples

```
sequence <- c("a", "a", "b", "b", "a", "c", "b", "a", "b", "c", "a", "b",
              "c", "a", "b", "c", "a", "b", "a", "b")
# compare orders 1 and 2 on the same observations (start = 3)
if (requireNamespace("Rsolnp", quietly = TRUE)) {
  fit1 <- fitHigherOrder(sequence, order = 1)
  fit2 <- fitHigherOrder(sequence, order = 2)
  sapply(list(order1 = fit1, order2 = fit2), function(f)
   unlist(higherOrderLogLik(sequence, f, start = 3)[c("logLik", "deviance", "AIC", "BIC")]))
}
```

HigherOrderMarkovChain-class *Higher order Markov Chains class*

Description

The S4 class that describes `HigherOrderMarkovChain` objects.

---

higherOrderPredict *Next-state probabilities and simulation for higher order Markov chains*

Description

`higherOrderPredict` returns the distribution of the next state given the most recent states, and `higherOrderSimulate` draws a sequence of states, under a higher order model fitted by `fitHigherOrder` or `fitMTD`.

Usage

```
higherOrderPredict(fit, history)

higherOrderSimulate(n, fit, t0, include.t0 = FALSE)
```

Arguments

`fit`	The list returned by `fitHigherOrder` or `fitMTD`.
`history`	The most recent states, oldest first: a vector of at least `order` states, or a matrix (or data frame) with one history per row.
`n`	Number of states to simulate.
`t0`	The states preceding the simulated ones, oldest first; at least `order` states.
`include.t0`	Should `t0` be included at the beginning of the returned sequence?

Details

Both functions use the transition probabilities of the fitted model of order k,

$$P(x_t = j \mid x_{t-1}, \ldots, x_{t-k}) = \sum_{i=1}^{k} \lambda_i \, Q_i[j, x_{t-i}],$$

the same that enter `higherOrderLogLik`. For a fit returned by `fitMTD` all the Q_i are the single MTD transition matrix. Only the last k states of a history are used, the last element being the most recent.

A model of order k needs k previous states, so `t0` (and every history) must contain at least `order` states. The probabilities are normalized to sum to one, which only matters when the weights of a least squares fit sum to one up to the optimizer's tolerance.

Value

`higherOrderPredict`: a named vector of next-state probabilities for a single history, or a matrix with one row per history. `higherOrderSimulate`: a character vector of `n` states, preceded by `t0` if `include.t0 = TRUE`.

See Also

`fitHigherOrder`, `fitMTD`, `higherOrderLogLik`, `rmarkovchain`

Examples

```
wind <- read.csv(system.file("extdata", "koeberg_wind.csv",
                             package = "markovchain"))$state
fit <- fitMTD(wind, order = 2)
# next wind direction after directions 1 and then 2
higherOrderPredict(fit, c(1, 2))
# several histories at once
higherOrderPredict(fit, rbind(c(1, 1), c(2, 2), c(4, 1)))
# simulate one day of hourly directions starting from the last two observed
set.seed(1)
higherOrderSimulate(24, fit, t0 = tail(wind, 2))

# the same works for fitHigherOrder()
data(rain)
fit2 <- fitHigherOrder(rain$rain, order = 2, method = "mle")
higherOrderPredict(fit2, c("0", "6+"))
```

`hittingProbabilities` *Hitting probabilities for markovchain*

Description

Given a markovchain object, this function calculates the probability of ever arriving from state i to j

Usage

```
hittingProbabilities(object, targets = NULL,
  solver = c("direct", "bicgstab", "doubling"), tol = 1e-13, maxIter = 200)
```

Arguments

`object`	the markovchain-class object
`targets`	optional character vector of state names: only the hitting probabilities *towards* these states are computed. The default, `NULL`, means all the states, which gives the full matrix as before. Every target is handled independently of the others, so the work is proportional to the number of targets: for a large chain, asking only for the states of interest is much faster than computing the whole matrix and subsetting it. Duplicated or unknown names are an error.
`solver`	the method used to solve the linear system $(I - Q)h = R$ on the states whose probability is neither structurally zero nor one (see Details). `"direct"`, the default, is an LU factorisation: it costs $O(m^3)$ once per target and is the most accurate. `"bicgstab"` is an unpreconditioned BiCGSTAB iteration on the sparse system: every iteration costs two sparse matrix-vector products instead of a dense $O(m^3)$ step, so it is the fastest choice on large sparse chains, at the price of a looser residual. On a breakdown of the iteration it restarts from the current residual, and if the breakdown persists it switches to `"direct"` with a warning. `"doubling"` is the doubled Neumann series used by versions up to 1.2, kept for reproducibility of earlier results; it is also the automatic fallback of `"direct"` on a numerically singular system.
`tol`	relative residual at which the iterative solvers (`"bicgstab"`, `"doubling"`) stop. Ignored by `"direct"`, except when it falls back to `"doubling"`.
`maxIter`	maximum number of iterations of the iterative solvers: the number of BiCGSTAB steps, or the number of squarings of the doubled Neumann series. A warning is raised, and the current values returned, if the requested `tol` is not reached within this many iterations.

Details

On each target the states are first split by graph reachability: a state that cannot reach the target has probability zero, and one that can reach the target but no closed class outside it has probability one. Only the remaining states need the linear system that `solver` controls, so on chains where that split already decides every state (an irreducible chain, for instance) all three solvers do the same negligible amount of work. The choice matters on chains with several closed classes, i.e. on genuine absorption probabilities.

Value

a matrix of hitting probabilities. Entry `[i, j]` is the probability of ever arriving from state `i` to state `j` (the probability of returning, after at least one transition, on the diagonal); for a chain with `byrow = FALSE` the matrix is transposed, as the transition matrix is. With `targets`, only the columns (rows if `byrow = FALSE`) of the targets are returned, in the order given, and they coincide with those of the full matrix.

Author(s)

Ignacio Cordón

References

R. Vélez, T. Prieto, Procesos Estocásticos, Librería UNED, 2013

H. A. van der Vorst (1992). Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems. *SIAM Journal on Scientific and Statistical Computing*, 13(2), 631-644.

Examples

```
M <- markovchain:::zeros(5)
M[1,1] <- M[5,5] <- 1
M[2,1] <- M[2,3] <- 1/2
M[3,2] <- M[3,4] <- 1/2
M[4,2] <- M[4,5] <- 1/2

mc <- new("markovchain", transitionMatrix = M)
hittingProbabilities(mc)

# only the probabilities of ever reaching the first state
hittingProbabilities(mc, targets = "1")

# on a large sparse chain, the iterative solver avoids the dense products
hittingProbabilities(mc, targets = "1", solver = "bicgstab")
```

---

holson *Holson data set*

Description

A data set containing 1000 life histories trajectories and a categorical status (1,2,3) observed on eleven evenly spaced steps.

Usage

```
data(holson)
```

Format

A data frame with 1000 observations on the following 12 variables.

`id` unique id

`time1` observed status at i-th time

`time2` observed status at i-th time

`time3` observed status at i-th time

`time4` observed status at i-th time

`time5` observed status at i-th time

`time6` observed status at i-th time

`time7` observed status at i-th time

`time8` observed status at i-th time

`time9` observed status at i-th time

`time10` observed status at i-th time

`time11` observed status at i-th time

Details

The example can be used to fit a `markovchain` or a `markovchainList` object.

Source

Private communications

References

Private communications

Examples

```
data(holson)
head(holson)
```

`hommc-class` *An S4 class for representing High Order Multivariate Markovchain (HOMMC)*

Description

An S4 class for representing High Order Multivariate Markovchain (HOMMC)

Usage

```
hommc
```

Slots

`order` an integer equal to order of Multivariate Markovchain

`states` a vector of states present in the HOMMC model

`P` array of transition matrices

`Lambda` a vector which stores the weightage of each transition matrices in P

`byrow` if FALSE each column sum of transition matrix is 1 else row sum = 1

`name` a name given to hommc

Author(s)

Giorgio Spedicato, Deepak Yadav

Examples

```
statesName <- c("a", "b")

P <- array(0, dim = c(2, 2, 4), dimnames = list(statesName, statesName))
P[,,1] <- matrix(c(0, 1, 1/3, 2/3), byrow = FALSE, nrow = 2)
P[,,2] <- matrix(c(1/4, 3/4, 0, 1), byrow = FALSE, nrow = 2)
P[,,3] <- matrix(c(1, 0, 1/3, 2/3), byrow = FALSE, nrow = 2)
P[,,4] <- matrix(c(3/4, 1/4, 0, 1), byrow = FALSE, nrow = 2)

Lambda <- c(0.8, 0.2, 0.3, 0.7)

ob <- new("hommc", order = 1, states = statesName, P = P,
          Lambda = Lambda, byrow = FALSE, name = "FOMMC")
```

---

ictmc-class *An S4 class for representing Imprecise Continuous Time Markovchains*

Description

An S4 class for representing Imprecise Continuous Time Markovchains

Slots

`states` a vector of states present in the ICTMC model

`Q` matrix representing the generator demonstrated in the form of variables

`range` a matrix that stores values of range of variables

`name` name given to ICTMC

identityChain *Identity Markov chain*

Description

Builds the Markov chain whose transition matrix is the identity: every state is absorbing, so the chain never moves.

Usage

```
identityChain(n, states = NULL, name = "Identity chain")
```

Arguments

n	The number of states. It can be omitted when states is given.
states	An optional character vector of n state names. Defaults to as.character(1:n).
name	The name slot of the result.

Details

The chain is the neutral element of the product of transition matrices (identityChain(n) * mc equals mc for a chain mc on the same states) and the extreme case of lazyChain. Every state is its own closed class, so every distribution is stationary.

Value

A markovchain object with n states.

See Also

randomMarkovChain, lazyChain

Examples

```
identityChain(3)
identityChain(states = c("a", "b"))
absorbingStates(identityChain(3))
```

`impliedTimescales` *Implied timescales of a Markov chain*

Description

Computes the implied relaxation timescale associated with each non-trivial eigenvalue of a finite, irreducible discrete-time Markov chain.

Usage

```
impliedTimescales(object)

## S4 method for signature 'markovchain'
impliedTimescales(object)
```

Arguments

`object`	A `markovchain` object representing a finite, irreducible discrete-time Markov chain.

Details

For a row-stochastic transition matrix P with eigenvalues $1 = \lambda_1, \lambda_2, \ldots, \lambda_n$ ($|\lambda_1|$ the unique unit eigenvalue of an irreducible chain), the implied timescale of λ_k, $k > 1$, is

$$\tau_k = -\frac{1}{\log|\lambda_k|}$$

for $0 < |\lambda_k| < 1$. Each τ_k measures how many steps the mode associated with λ_k takes to decay by a factor of $1/e$; larger timescales correspond to slower-decaying, more persistent modes.

Only irreducibility is required, not aperiodicity: this is the same convention used by `slem` and `spectralGap`, and it lets `impliedTimescales()` document periodic and boundary cases explicitly rather than rejecting them:

- If $|\lambda_k|$ is (numerically) exactly 1 – which happens for non-trivial eigenvalues of periodic chains, e.g. $\lambda = -1$ for a 2-cycle – the corresponding mode never decays and `tau_k = Inf` is returned. This is a boundary case of the formula above (as $|\lambda| \to 1^-$, $\tau \to \infty$) that is handled explicitly rather than by evaluating $-1/\log(1)$, which is numerically `-Inf` rather than the mathematically correct `+Inf`.
- If $|\lambda_k|$ is (numerically) exactly 0, the mode decays immediately and `tau_k = 0` is returned. `log(0)` evaluates to `-Inf` in R, so this case is already handled correctly by the formula itself and needs no special-casing.

The term "implied timescale" follows the Markov state model literature in molecular kinetics, where it is additionally used, across chains estimated at increasing lag times, as a self-consistency check on the Markov (memoryless) approximation: implied timescales that are approximately constant across lag times support the model, while ones that drift indicate it should be revisited (see Prinz et al. (2011)). Building such a lag-time comparison is left to the user, since it requires re-estimating the

chain at each lag: `impliedTimescales()` itself only evaluates a single, already-fitted `markovchain` object.

The implementation calls `eigen()` with `only.values = TRUE`, so it never computes eigenvectors. Its time complexity is $O(n^3)$ and its memory use is $O(n^2)$ for a dense n-state transition matrix. It supports both row- and column-stochastic storage.

Value

A named numeric vector of length $n - 1$ (one entry per non-trivial eigenvalue), sorted by decreasing timescale, i.e. by decreasing eigenvalue modulus. Names are `"tau2"`, `"tau3"`, ..., matching the usual eigenvalue indexing $\lambda_2, \lambda_3, \ldots$ in decreasing modulus. For the trivial one-state chain, a length-zero named numeric vector is returned.

References

Swope, W. C., Pitera, J. W. and Suits, F. (2004). Describing protein folding kinetics by molecular dynamics simulations, 1: Theory. *J. Phys. Chem. B*, 108(21), 6571-6581.

Prinz, J.-H., Wu, H., Sarich, M., Keller, B., Senne, M., Held, M., Chodera, J. D., Schutte, C. and Noe, F. (2011). Markov models of molecular kinetics: Generation and validation. *Journal of Chemical Physics*, 134(17), 174105.

See Also

`slem`, `spectralGap`, `is.irreducible`

Examples

```
statesNames <- c("a", "b")
mc <- new("markovchain",
  states = statesNames,
  transitionMatrix = matrix(c(0.7, 0.3, 0.1, 0.9),
    byrow = TRUE, nrow = 2,
    dimnames = list(statesNames, statesNames)))
impliedTimescales(mc)
```

---

impreciseProbabilityatT

Calculating full conditional probability using lower rate transition matrix

---

Description

This function calculates full conditional probability at given time s using lower rate transition matrix

Usage

```
impreciseProbabilityatT(C,i,t,s,error,useRCpp)
```

Arguments

C	a ictmc class object
i	initial state at time t
t	initial time t. Default value = 0
s	final time
error	error rate. Default value = 0.001
useRCpp	logical whether to use RCpp implementation; by default TRUE

Author(s)

Vandit Jain

References

Imprecise Continuous-Time Markov Chains, Thomas Krak et al., 2016

Examples

```
states <- c("n","y")
Q <- matrix(c(-1,1,1,-1),nrow = 2,byrow = TRUE,dimnames = list(states,states))
range <- matrix(c(1/52,3/52,1/2,2),nrow = 2,byrow = 2)
name <- "testictmc"
ictmc <- new("ictmc",states = states,Q = Q,range = range,name = name)
impreciseProbabilityatT(ictmc,2,0,1,10^-3,TRUE)
```

---

inferHyperparam — *Function to infer the hyperparameters for Bayesian inference from an a priori matrix or a data set*

Description

Since the Bayesian inference approach implemented in the package is based on conjugate priors, hyperparameters must be provided to model the prior probability distribution of the chain parameters. The hyperparameters are inferred from a given a priori matrix under the assumption that the matrix provided corresponds to the mean (expected) values of the chain parameters. A scaling factor vector must be provided too. Alternatively, the hyperparameters can be inferred from a data set.

Usage

```
inferHyperparam(transMatr = matrix(), scale = numeric(), data = character())
```

Arguments

transMatr	A valid transition matrix, with dimension names.
scale	A vector of scaling factors, each element corresponds to the row names of the provided transition matrix transMatr, in the same order.
data	A data set from which the hyperparameters are inferred.

Details

transMatr and scale need not be provided if data is provided.

Value

Returns the hyperparameter matrix in a list.

Note

The hyperparameter matrix returned is such that the row and column names are sorted alphanumerically, and the elements in the matrix are correspondingly permuted.

Author(s)

Sai Bhargav Yalamanchi, Giorgio Spedicato

References

Yalamanchi SB, Spedicato GA (2015). Bayesian Inference of First Order Markov Chains. R package version 0.2.5

See Also

markovchainFit, predictiveDistribution

Examples

```
data(rain, package = "markovchain")
inferHyperparam(data = rain$rain)

weatherStates <- c("sunny", "cloudy", "rain")
weatherMatrix <- matrix(data = c(0.7, 0.2, 0.1,
                                 0.3, 0.4, 0.3,
                                 0.2, 0.4, 0.4),
                        byrow = TRUE, nrow = 3,
                        dimnames = list(weatherStates, weatherStates))
inferHyperparam(transMatr = weatherMatrix, scale = c(10, 10, 10))
```

---

is.accessible *Verify if a state j is reachable from state i.*

Description

This function verifies if a state is reachable from another, i.e., if there exists a path that leads to state j leaving from state i with positive probability

Usage

```
is.accessible(object, from, to)
```

Arguments

object	A markovchain object.
from	The name of state "i" (beginning state).
to	The name of state "j" (ending state).

Details

It wraps an internal function named reachabilityMatrix.

Value

A boolean value.

Author(s)

Giorgio Spedicato, Ignacio Cordón

References

James Montgomery, University of Madison

See Also

is.irreducible

Examples

```
statesNames <- c("a", "b", "c")
markovB <- new("markovchain", states = statesNames,
               transitionMatrix = matrix(c(0.2, 0.5, 0.3,
                                           0,   1,   0,
                                         0.1, 0.8, 0.1), nrow = 3, byrow = TRUE,
                                        dimnames = list(statesNames, statesNames)
                                       )
               )
is.accessible(markovB, "a", "c")
```

`is.CTMCirreducible` *Check if CTMC is irreducible*

Description

This function verifies whether a CTMC object is irreducible

Usage

```
is.CTMCirreducible(ctmc)
```

Arguments

`ctmc`	a ctmc-class object

Value

a boolean value as described above.

Author(s)

Vandit Jain

References

Continuous-Time Markov Chains, Karl Sigman, Columbia University

Examples

```
energyStates <- c("sigma", "sigma_star")
byRow <- TRUE
gen <- matrix(data = c(-3, 3,
                       1, -1), nrow = 2,
              byrow = byRow, dimnames = list(energyStates, energyStates))
molecularCTMC <- new("ctmc", states = energyStates,
                     byrow = byRow, generator = gen,
                     name = "Molecular Transition Model")
is.CTMCirreducible(molecularCTMC)
```

`is.irreducible` *Function to check if a Markov chain is irreducible (i.e. ergodic)*

Description

This function verifies whether a `markovchain` object transition matrix is composed by only one communicating class.

Usage

```
is.irreducible(object)
```

Arguments

`object`	A `markovchain` object

Details

It is based on `.communicatingClasses` internal function.

Value

A boolean values.

Author(s)

Giorgio Spedicato

References

Feres, Matlab listings for Markov Chains.

See Also

`summary`

Examples

```
statesNames <- c("a", "b")
mcA <- new("markovchain", transitionMatrix = matrix(c(0.7,0.3,0.1,0.9),
                                                    byrow = TRUE, nrow = 2,
                                                    dimnames = list(statesNames, statesNames)
           ))
is.irreducible(mcA)
```

is.lumpable *Check exact lumpability of a Markov chain*

Description

Verifies the strong lumpability condition with respect to a partition of the state space. For every pair of macro-states, all micro-states in the same source macro-state must have the same total probability of moving to the destination macro-state.

Usage

```
is.lumpable(object, partition, tol = 1e-10)

## S4 method for signature 'markovchain'
is.lumpable(object, partition, tol = 1e-10)
```

Arguments

object	A markovchain object.
partition	A named list of character vectors defining macro-states.
tol	Non-negative numerical tolerance for equality checks.

Value

A logical value.

References

Kemeny, J. G. and Snell, J. L. (1960). *Finite Markov Chains*.

is.regular *Check if a DTMC is regular*

Description

Function to check wether a DTCM is regular

Usage

```
is.regular(object)
```

Arguments

object	a markovchain object

Details

A Markov chain is regular if some of the powers of its matrix has all elements strictly positive

Value

A boolean value

Author(s)

Ignacio Cordón

References

Matrix Analysis. Roger A.Horn, Charles R.Johnson. 2nd edition. Corollary 8.5.8, Theorem 8.5.9

See Also

is.irreducible

Examples

```
P <- matrix(c(0.5,  0.25, 0.25,
              0.5,     0, 0.5,
              0.25, 0.25, 0.5), nrow = 3)
colnames(P) <- rownames(P) <- c("R","N","S")
ciao <- as(P, "markovchain")
is.regular(ciao)
```

---

is.reversible *Check whether a Markov chain is reversible*

Description

Checks whether a finite, irreducible discrete-time Markov chain is reversible with respect to its (unique) stationary distribution, i.e. whether it satisfies the detailed balance equations.

Usage

```
is.reversible(object, tolerance = sqrt(.Machine$double.eps))

## S4 method for signature 'markovchain'
is.reversible(object, tolerance = sqrt(.Machine$double.eps))
```

Arguments

`object`	A `markovchain` object representing a finite, irreducible discrete-time Markov chain.
`tolerance`	A single finite non-negative number. Detailed balance is accepted as holding when every pair (i, j) satisfies $\lvert\pi_i P_{ij} - \pi_j P_{ji}\rvert \leq$ `tolerance`. The default is a small numerical-noise tolerance, not a modelling tolerance: it exists to absorb floating-point rounding in the eigendecomposition-based `steadyStates` computation, not to declare "almost reversible" chains reversible.

Details

A chain with transition matrix P and stationary distribution π is reversible if

$$\pi_i P_{ij} = \pi_j P_{ji} \quad \text{for every } i, j.$$

Intuitively, if you started the chain from π and watched a long run of it, running the recorded sequence of states backwards would look statistically identical to running it forwards: at stationarity, the "flow" of probability from i to j exactly balances the flow from j to i.

Only irreducibility is required, not aperiodicity: detailed balance is a purely algebraic condition on P and π and is perfectly well defined for periodic chains too. For example, a simple random walk on any undirected graph (moving to a uniformly random neighbour) is always reversible, whether or not it happens to be periodic.

A 2-state irreducible chain is always reversible: with only two states, the single detailed balance equation $\pi_1 P_{12} = \pi_2 P_{21}$ is just a restatement of the stationarity equation $\pi P = \pi$, so it holds automatically.

Every reversible chain has a real spectrum (all eigenvalues of P are real), which is why `slem` and `spectralGap` are especially easy to interpret for reversible chains: there are no complex-conjugate eigenvalue pairs to reason about.

The implementation calls `steadyStates` once, then compares the two triangles of the flow matrix $\pi_i P_{ij}$. Its time complexity is dominated by `steadyStates()`, plus an additional $O(n^2)$ comparison for a dense n-state transition matrix. It supports both row- and column-stochastic storage.

Value

A single logical value, `TRUE` or `FALSE`.

References

Norris, J. R. (1998). *Markov Chains*. Cambridge University Press.

Levin, D. A. and Peres, Y. (2017). *Markov Chains and Mixing Times*, 2nd edition. American Mathematical Society.

See Also

`steadyStates`, `is.irreducible`, `slem`, `mixingTime`

Examples

```
# A random walk on a triangle is reversible.
statesNames <- c("a", "b", "c")
triangle <- new("markovchain", states = statesNames,
  transitionMatrix = matrix(c(0, 0.5, 0.5,
                              0.5, 0, 0.5,
                              0.5, 0.5, 0), byrow = TRUE, nrow = 3,
                            dimnames = list(statesNames, statesNames)))
is.reversible(triangle)

# A directed cycle (states only move "forward") is not reversible: there
# is a net clockwise flow of probability at stationarity.
cycle3 <- new("markovchain", states = statesNames,
  transitionMatrix = matrix(c(0, 1, 0,
                              0, 0, 1,
                              1, 0, 0), byrow = TRUE, nrow = 3,
                            dimnames = list(statesNames, statesNames)))
is.reversible(cycle3)
```

---

is.stochasticallyMonotone

Check if a Markov chain is stochastically monotone

Description

Verifies if the transition matrix of the Markov chain is stochastically monotone.

Usage

```
is.stochasticallyMonotone(object)

## S4 method for signature 'markovchain'
is.stochasticallyMonotone(object)

## S4 method for signature 'matrix'
is.stochasticallyMonotone(object)

## S4 method for signature 'ANY'
is.stochasticallyMonotone(object)
```

Arguments

`object`	A markovchain object or a transition matrix.

Value

A boolean value.

`is.TimeReversible` *checks if ctmc object is time reversible*

Description

The function returns checks if provided function is time reversible

Usage

```
is.TimeReversible(ctmc)
```

Arguments

`ctmc`	a ctmc-class object

Value

Returns a boolean value stating whether ctmc object is time reversible

a boolean value as described above

Author(s)

Vandit Jain

References

INTRODUCTION TO STOCHASTIC PROCESSES WITH R, ROBERT P. DOBROW, Wiley

Examples

```
energyStates <- c("sigma", "sigma_star")
byRow <- TRUE
gen <- matrix(data = c(-3, 3,
                       1, -1), nrow = 2,
              byrow = byRow, dimnames = list(energyStates, energyStates))
molecularCTMC <- new("ctmc", states = energyStates,
                     byrow = byRow, generator = gen,
                     name = "Molecular Transition Model")
is.TimeReversible(molecularCTMC)
```

kemenyConstant *Kemeny's constant of a Markov chain*

Description

Computes Kemeny's constant for a finite, irreducible discrete-time Markov chain. It is the stationary-distribution-weighted mean hitting time of a randomly selected destination and is independent of the starting state.

Usage

```
kemenyConstant(object)

## S4 method for signature 'markovchain'
kemenyConstant(object)
```

Arguments

object	A markovchain object representing a finite, irreducible discrete-time Markov chain.

Details

For a row-stochastic transition matrix P, let π be its unique stationary distribution and define

$$Z = (I - P + \mathbf{1}\pi^T)^{-1}.$$

With hitting times defined by $T_j = \inf\{n \geq 0 : X_n = j\}$, so that $m_{jj} = 0$, the function returns

$$K = \sum_j \pi_j m_{ij} = \mathrm{tr}(Z) - 1.$$

The value does not depend on the starting state i.

Irreducibility is sufficient; aperiodicity is not required. Reducible chains can have multiple stationary distributions and are rejected.

Some references instead put the mean first-return time $m_{jj} = 1/\pi_j$ on the diagonal. Under that convention the corresponding stationary weighted sum is $K + 1$, not K. This function uses the zero-diagonal hitting-time convention, consistently with meanFirstPassageTime().

The implementation uses a dense LAPACK solve for the fundamental matrix Z. Its time complexity is $O(n^3)$ and its memory use is $O(n^2)$, as expected for a dense exact computation. It supports both row- and column-stochastic storage.

Value

A numeric scalar containing Kemeny's constant.

References

Kemeny, J. G. and Snell, J. L. (1960). *Finite Markov Chains*. D. Van Nostrand, Princeton, NJ.

See Also

`meanFirstPassageTime`, `steadyStates`, `is.irreducible`

Examples

```
statesNames <- c("a", "b")
mc <- new("markovchain",
  states = statesNames,
  transitionMatrix = matrix(c(0.7, 0.3, 0.1, 0.9),
    byrow = TRUE, nrow = 2,
    dimnames = list(statesNames, statesNames)))
kemenyConstant(mc)
```

---

kullback *Example from Kullback and Kupperman Tests for Contingency Tables*

Description

A list of two matrices representing raw transitions between two states

Usage

```
data(kullback)
```

Format

A list containing two 6x6 non - negative integer matrices

---

lazyChain *Build a lazy version of a Markov chain*

Description

Constructs the "lazy" chain associated with a `markovchain` object: at every step, stay put with probability `alpha` and otherwise take a step of the original chain.

Usage

```
lazyChain(object, alpha = 0.5)

## S4 method for signature 'markovchain'
lazyChain(object, alpha = 0.5)
```

Arguments

`object`	A `markovchain` object.
`alpha`	A single number in $[0, 1]$, the probability of staying in the current state at each step. The default, `0.5`, matches the usual textbook "lazy random walk" construction.

Details

For a transition matrix P (in whichever storage convention `object` already uses) and a laziness parameter $\alpha \in [0, 1]$, the lazy chain's transition matrix is

$$L = \alpha I + (1 - \alpha)P.$$

Laziness is a standard device for forcing aperiodicity without changing where the chain can go or its stationary distribution:

- L has the *same* stationary distribution as P (if $\pi P = \pi$ then $\pi L = \alpha\pi + (1 - \alpha)\pi P = \pi$), and the same communicating classes, since $L_{ij} > 0 \iff P_{ij} > 0$ for $i \neq j$.
- For $0 < \alpha < 1$, L is aperiodic even if P is periodic, because $L_{ii} = \alpha > 0$ for every state i rules out any period greater than 1. This is why `mixingTime`, which requires aperiodicity, is often applied to `lazyChain(object)` rather than to a periodic `object` directly (see `mixingTime`'s own documentation for why it rejects periodic chains outright rather than lazifying them automatically).
- Every non-trivial eigenvalue of L is $\alpha + (1 - \alpha)\lambda$ for the corresponding eigenvalue λ of P: laziness shrinks the whole non-trivial spectrum towards α, so `slem` and `impliedTimescales` generally get *worse* (mixing gets slower) as `alpha` increases towards 1.

$\alpha = 0$ returns P unchanged; $\alpha = 1$ returns the identity matrix (a chain that never moves).

Value

A new `markovchain` object with transition matrix $L = \alpha I + (1 - \alpha)P$, the same states, and the same row/column-stochastic storage convention (`byrow`) as `object`.

References

Levin, D. A. and Peres, Y. (2017). *Markov Chains and Mixing Times*, 2nd edition. American Mathematical Society.

See Also

`subchain`, `mixingTime`, `slem`

Examples

```
# A 2-cycle is periodic (period 2); its lazy version is aperiodic.
statesNames <- c("a", "b")
cycle2 <- new("markovchain", states = statesNames,
  transitionMatrix = matrix(c(0, 1, 1, 0), byrow = TRUE, nrow = 2,
```

```
                              dimnames = list(statesNames, statesNames)))
period(cycle2)
lazyCycle2 <- lazyChain(cycle2, alpha = 0.5)
period(lazyCycle2)
steadyStates(cycle2)
steadyStates(lazyCycle2) # unchanged by laziness
```

`lump` *Aggregate a Markov chain over a partition*

Description

Coarsens a Markov chain to a reduced state space. By default the function requires exact lumpability. With `force = TRUE`, it performs an approximate aggregation using stationary weights when available and arithmetic averages for macro-states with zero stationary mass.

Usage

```
lump(object, partition, force = FALSE)

## S4 method for signature 'markovchain'
lump(object, partition, force = FALSE)
```

Arguments

`object`	A `markovchain` object.
`partition`	A named list of character vectors defining macro-states.
`force`	If `FALSE`, stop unless the chain is exactly lumpable. If `TRUE`, return a weighted approximate lumping.

Value

A `markovchain` object on the macro-state space.

`markovchain-class` *Markov Chain class*

Description

The S4 class that describes `markovchain` objects.

Slots

`states` Name of the states. Must be the same of `colnames` and `rownames` of the transition matrix

`byrow` TRUE or FALSE indicating whether the supplied matrix is either stochastic by rows or by columns

`transitionMatrix` Square transition matrix

`name` Optional character name of the Markov chain

Creation of objects

Objects can be created by calls of the form `new("markovchain", states, byrow, transitionMatrix, ...)`.

Methods

* `signature(e1 = "markovchain", e2 = "markovchain")`: multiply two `markovchain` objects

* `signature(e1 = "markovchain", e2 = "matrix")`: markovchain by matrix multiplication

* `signature(e1 = "markovchain", e2 = "numeric")`: markovchain by numeric vector multiplication

* `signature(e1 = "matrix", e2 = "markovchain")`: matrix by markov chain

* `signature(e1 = "numeric", e2 = "markovchain")`: numeric vector by `markovchain` multiplication

[`signature(x = "markovchain", i = "ANY", j = "ANY", drop = "ANY")`: ...

^ `signature(e1 = "markovchain", e2 = "numeric")`: power of a `markovchain` object

== `signature(e1 = "markovchain", e2 = "markovchain")`: equality of two `markovchain` object

!= `signature(e1 = "markovchain", e2 = "markovchain")`: non-equality of two `markovchain` object

absorbingStates `signature(object = "markovchain")`: method to get absorbing states

canonicForm `signature(object = "markovchain")`: return a `markovchain` object into canonic form

coerce `signature(from = "markovchain", to = "data.frame")`: coerce method from markovchain to `data.frame`

conditionalDistribution `signature(object = "markovchain")`: returns the conditional probability of subsequent states given a state

coerce `signature(from = "data.frame", to = "markovchain")`: coerce method from `data.frame` to `markovchain`

coerce `signature(from = "table", to = "markovchain")`: coerce method from `table` to `markovchain`

coerce `signature(from = "msm", to = "markovchain")`: coerce method from `msm` to `markovchain`

coerce `signature(from = "msm.est", to = "markovchain")`: coerce method from `msm.est` (but only from a Probability Matrix) to `markovchain`

coerce `signature(from = "etm", to = "markovchain")`: coerce method from `etm` to `markovchain`

coerce `signature(from = "sparseMatrix", to = "markovchain")`: coerce method from `sparseMatrix` to `markovchain`

coerce `signature(from = "markovchain", to = "igraph")`: coercing to `igraph` objects

coerce `signature(from = "markovchain", to = "matrix")`: coercing to `matrix` objects

coerce `signature(from = "markovchain", to = "sparseMatrix")`: coercing to `sparseMatrix` objects

coerce `signature(from = "matrix", to = "markovchain")`: coercing to `markovchain` objects from `matrix` one

dim `signature(x = "markovchain")`: method to get the size

names `signature(x = "markovchain")`: method to get the names of states

names<- `signature(x = "markovchain", value = "character")`: method to set the names of states

initialize `signature(.Object = "markovchain")`: initialize method

plot `signature(x = "markovchain", y = "missing")`: plot method for `markovchain` objects

predict `signature(object = "markovchain")`: predict method. Starting from the last element of `newdata`, which must be a state of the chain, it returns the `n.ahead` following states, each being the most probable transition from the previous one; ties are broken at random.

print `signature(x = "markovchain")`: print method.

show `signature(object = "markovchain")`: show method.

sort `signature(x = "markovchain", decreasing=FALSE)`: sorting the transition matrix.

states `signature(object = "markovchain")`: returns the names of states (as `names`.

steadyStates `signature(object = "markovchain")`: method to get the steady vector.

summary `signature(object = "markovchain")`: method to summarize structure of the markov chain. `summary(object, details = TRUE)` also prints, after the usual output, the size and rank of the transition matrix, the number of communicating classes, irreducibility, period, regularity, whether the chain is absorbing, reversible, stochastically monotone and symmetric, and, when they are defined, the entropy rate, the SLEM, the spectral gap and Kemeny's constant; these values are also returned, invisibly, in the `details` element of the result.

transientStates `signature(object = "markovchain")`: method to get the transient states.

t `signature(x = "markovchain")`: transpose matrix

transitionProbability `signature(object = "markovchain")`: transition probability

Note

1. `markovchain` object are backed by S4 Classes.
2. Validation method is used to assess whether either columns or rows totals to one. Rounding is used up to `.Machine$double.eps * 100`. If state names are not properly defined for a probability `matrix`, coercing to `markovchain` object leads to overriding states name with artificial "s1", "s2", ... sequence. In addition, operator overloading has been applied for $+, *, ^, ==, !=$ operators.

Author(s)

Giorgio Spedicato

References

A First Course in Probability (8th Edition), Sheldon Ross, Prentice Hall 2010

See Also

markovchainSequence,markovchainFit

Examples

```
#show markovchain definition
showClass("markovchain")
#create a simple Markov chain
transMatr<-matrix(c(0.4,0.6,.3,.7),nrow=2,byrow=TRUE)
simpleMc<-new("markovchain", states=c("a","b"),
              transitionMatrix=transMatr,
              name="simpleMc")
#power
simpleMc^4
#some methods
steadyStates(simpleMc)
absorbingStates(simpleMc)
simpleMc[2,1]
t(simpleMc)
is.irreducible(simpleMc)
#conditional distributions
conditionalDistribution(simpleMc, "b")
#example for predict method
sequence<-c("a", "b", "a", "a", "a", "a", "b", "a", "b", "a", "b", "a", "a", "b", "b", "b", "a")
mcFit<-markovchainFit(data=sequence)
predict(mcFit$estimate, newdata="b",n.ahead=3)
#direct conversion
myMc<-as(transMatr, "markovchain")

#example of summary
summary(simpleMc)
summary(simpleMc, details = TRUE)
## Not run: plot(simpleMc)
```

---

markovchainFit *Function to fit a discrete Markov chain*

Description

Given a sequence of states arising from a stationary state, it fits the underlying Markov chain distribution using either MLE (also using a Laplacian smoother), bootstrap or by MAP (Bayesian) inference.

Usage

```
.markovchainFitRcpp(
  data,
  method = "mle",
  byrow = TRUE,
  nboot = 10L,
  laplacian = 0,
  name = "",
  parallel = FALSE,
  confidencelevel = 0.95,
  confint = TRUE,
  hyperparam = matrix(),
  sanitize = FALSE,
  possibleStates = character(),
  progress = FALSE
)

createSequenceMatrix(
  stringchar,
  toRowProbs = FALSE,
  sanitize = FALSE,
  possibleStates = character()
)

markovchainFit(data, method = "mle", byrow = TRUE, nboot = 10L,
  laplacian = 0, name = "", parallel = FALSE, confidencelevel = 0.95,
  confint = TRUE, hyperparam = matrix(), sanitize = FALSE,
  possibleStates = character(), absorbingStates = character(),
  progress = FALSE, num.cores = NULL)
```

Arguments

data	It can be a character vector or a nxn matrix or a nxn data frame or a list
method	Method used to estimate the Markov chain. Either "mle", "map", "bootstrap" or "laplace". All four are available for a single sequence. For a list of sequences, "mle", "map" and "laplace" pool the transition counts over the sequences, while "bootstrap" is not available and raises an error explaining why.
byrow	For a character vector or a list of sequences, it tells whether the fitted transition matrix is stored by row (the default) or by column; the `byrow` slot of the returned chain records the choice. For matrix or data frame input it instead describes the input data – whether each observed trajectory is a row or a column of `data` – and the fitted chain is stored by row either way.

nboot Number of bootstrap replicates in case "bootstrap" is used.

laplacian Laplacian smoothing parameter, default zero. It is only used when "laplace" method is chosen.

name Optional character for name slot.

parallel Use parallel processing when performing Boostrap estimates.

confidencelevel

$$\alpha$$

level for conficence intervals width. Used only when method equal to "mle".

confint a boolean to decide whether to compute Confidence Interval or not.

hyperparam Hyperparameter matrix for the a priori distribution. If none is provided, default value of 1 is assigned to each parameter. This must be of size

$$kxk$$

where k is the number of states in the chain and the values should typically be non-negative integers.

sanitize how to deal with the states that have no observed outgoing transition, which is what possibleStates typically introduces. FALSE (the default) leaves their row at zero, which makes a transition matrix that is not stochastic. TRUE, or equivalently "uniform", puts 1 in every entry of such a row, so that the row becomes a uniform distribution over all the states: the unobserved states are then assumed to move to any state with equal probability, which is an assumption about the data and not a consequence of it. "absorbing" instead puts 1 on the diagonal only, making every unobserved state absorbing; this also gives a stochastic matrix, but adds no transition that was never observed (see #213).

possibleStates Possible states which are not present in the given sequence

progress Should a text progress bar be shown? It is only used by the "bootstrap" method, the other methods being fast; see Details.

stringchar It can be a

$$nxn$$

matrix or a character vector or a list

toRowProbs converts a sequence matrix into a probability matrix

absorbingStates

Character vector of states that are known a priori to be absorbing. The corresponding rows are set to the identity row after MLE fitting when byrow = TRUE; the corresponding columns are set to the identity column when byrow = FALSE. The argument is currently supported only for method = "mle".

num.cores Number of threads the parallel bootstrap path uses when method = "bootstrap" and parallel = TRUE. If NULL (the default) the thread count is read from getOption("RcppParallel.num / getOption("Ncpus") / OMP_NUM_THREADS / RCPP_PARALLEL_NUM_THREADS, falling back to min(2, cores) as CRAN policy requires. Ignored when parallel = FALSE.

Details

Disabling confint would lower the computation time on large datasets. If data or stringchar contain NAs, the related NA containing transitions will be ignored.

With progress = TRUE the "bootstrap" method shows a text progress bar (txtProgressBar, style 3) covering the simulation of the nboot bootstrap sequences and the estimation of a transition matrix from each of them. With parallel = TRUE the sequences are simulated in parallel threads, which cannot report progress, so the bar only covers the estimation step. The computation can be interrupted while the bar is shown.

When absorbingStates is supplied, the declared states must have no observed outgoing transitions. This allows terminal states in censored customer journeys to be represented as absorbing states without adding artificial observations.

sanitize = "absorbing" reaches the same result without naming the states: every state that has no observed outgoing transition, which is what possibleStates typically introduces, is made absorbing. Unlike absorbingStates, it works with every method, since it only replaces the uniform row that sanitize = TRUE would have produced. As with absorbingStates, only the estimate is constrained: any confidence bounds and standard errors keep the values the unconstrained fit assigned to those rows.

Value

A list containing an estimate, log-likelihood, and, when "bootstrap" method is used, a matrix of standards deviations and the bootstrap samples. When the "mle", "bootstrap" or "map" method is used, the lower and upper confidence bounds are returned along with the standard error. The "map" method also returns the expected value of the parameters with respect to the posterior distribution.

Note

This function has been rewritten in Rcpp. Bootstrap algorithm has been defined "heuristically". In addition, parallel facility is not complete, involving only a part of the bootstrap process. When data is either a data.frame or a matrix object, only MLE fit is currently available.

Author(s)

Giorgio Spedicato, Tae Seung Kang, Sai Bhargav Yalamanchi

References

A First Course in Probability (8th Edition), Sheldon Ross, Prentice Hall 2010

Inferring Markov Chains: Bayesian Estimation, Model Comparison, Entropy Rate, and Out-of-Class Modeling, Christopher C. Strelioff, James P. Crutchfield, Alfred Hubler, Santa Fe Institute

Yalamanchi SB, Spedicato GA (2015). Bayesian Inference of First Order Markov Chains. R package version 0.2.5

See Also

markovchainSequence, markovchainListFit

Examples

```
sequence <- c("a", "b", "a", "a", "a", "a", "b", "a", "b", "a", "b", "a", "a",
              "b", "b", "b", "a")
sequenceMatr <- createSequenceMatrix(sequence, sanitize = FALSE)
mcFitMLE <- markovchainFit(data = sequence)
mcFitBSP <- markovchainFit(data = sequence, method = "bootstrap", nboot = 5, name = "Bootstrap Mc")

na.sequence <- c("a", NA, "a", "b")
# There will be only a (a,b) transition
na.sequenceMatr <- createSequenceMatrix(na.sequence, sanitize = FALSE)
mcFitMLE <- markovchainFit(data = na.sequence)

# data can be a list of character vectors
sequences <- list(x = c("a", "b", "a"), y = c("b", "a", "b", "a", "c"))
mcFitMap <- markovchainFit(sequences, method = "map")
mcFitMle <- markovchainFit(sequences, method = "mle")
```

---

markovchainList-class *Non homogeneus discrete time Markov Chains class*

Description

A class to handle non homogeneous discrete Markov chains

Slots

`markovchains` Object of class `"list"`: a list of markovchains

`name` Object of class `"character"`: optional name of the class

Objects from the Class

A `markovchainlist` is a list of `markovchain` objects. They can be used to model non homogeneous discrete time Markov Chains, when transition probabilities (and possible states) change by time.

Methods

[[`signature(x = "markovchainList")`: extract the i-th `markovchain`

dim `signature(x = "markovchainList")`: number of `markovchain` underlying the matrix

predict `signature(object = "markovchainList")`: predict from a `markovchainList`

print `signature(x = "markovchainList")`: prints the list of markovchains

show `signature(object = "markovchainList")`: same as `print`

Note

The class consists in a list of `markovchain` objects. It is aimed at working with non homogeneous Markov chains.

Author(s)

Giorgio Spedicato

References

A First Course in Probability (8th Edition), Sheldon Ross, Prentice Hall 2010

See Also

markovchain

Examples

```
showClass("markovchainList")
#define a markovchainList
statesNames=c("a","b")

mcA<-new("markovchain",name="MCA",
         transitionMatrix=matrix(c(0.7,0.3,0.1,0.9),
                         byrow=TRUE, nrow=2,
                         dimnames=list(statesNames,statesNames))
        )

mcB<-new("markovchain", states=c("a","b","c"), name="MCB",
         transitionMatrix=matrix(c(0.2,0.5,0.3,0,1,0,0.1,0.8,0.1),
         nrow=3, byrow=TRUE))

mcC<-new("markovchain", states=c("a","b","c","d"), name="MCC",
         transitionMatrix=matrix(c(0.25,0.75,0,0,0.4,0.6,
                                   0,0,0,0,0.1,0.9,0,0,0.7,0.3),
                                 nrow=4, byrow=TRUE)
)
mcList<-new("markovchainList",markovchains=list(mcA, mcB, mcC),
           name="Non - homogeneous Markov Chain")
```

---

markovchainListFit *markovchainListFit*

Description

Given a data frame or a matrix (rows are observations, by cols the temporal sequence), it fits a non - homogeneous discrete time markov chain process (storing row). In particular a markovchainList of size = ncol - 1 is obtained estimating transitions from the n samples given by consecutive column pairs.

Usage

```
markovchainListFit(data, byrow = TRUE, laplacian = 0, name)
```

Arguments

data	Either a matrix or a data.frame or a list object.
byrow	Indicates whether distinc stochastic processes trajectiories are shown in distinct rows.
laplacian	Laplacian correction (default 0).
name	Optional name.

Details

If data contains NAs then the transitions containing NA will be ignored.

Value

A list containing two slots: estimate (the estimate) name

Examples

```
# using holson dataset
data(holson)
# fitting a single markovchain
singleMc <- markovchainFit(data = holson[,2:12])
# fitting a markovchainList
mclistFit <- markovchainListFit(data = holson[, 2:12], name = "holsonMcList")
```

---

markovchainSequence *Function to generate a sequence of states from homogeneous Markov chains.*

Description

Provided any markovchain object, it returns a sequence of states coming from the underlying stationary distribution.

Usage

```
markovchainSequence(
  n,
  markovchain,
  t0 = sample(markovchain@states, 1),
  include.t0 = FALSE,
  useRCpp = TRUE
)
```

Arguments

n	Sample size
markovchain	markovchain object
t0	The initial state
include.t0	Specify if the initial state shall be used
useRCpp	Boolean. Should RCpp fast implementation being used? Default is yes.

Details

A sequence of size n is sampled.

Value

A Character Vector

Author(s)

Giorgio Spedicato

References

A First Course in Probability (8th Edition), Sheldon Ross, Prentice Hall 2010

See Also

markovchainFit

Examples

```
# define the markovchain object
statesNames <- c("a", "b", "c")
mcB <- new("markovchain", states = statesNames,
   transitionMatrix = matrix(c(0.2, 0.5, 0.3, 0, 0.2, 0.8, 0.1, 0.8, 0.1),
   nrow = 3, byrow = TRUE, dimnames = list(statesNames, statesNames)))

# show the sequence
outs <- markovchainSequence(n = 100, markovchain = mcB, t0 = "a")
```

`meanAbsorptionTime` *Mean absorption time*

Description

Computes the expected number of steps to go from any of the transient states to any of the recurrent states. The Markov chain should have at least one transient state for this method to work

Usage

```
meanAbsorptionTime(object)
```

Arguments

`object`	the markovchain object

Value

A named vector with the expected number of steps to go from a transient state to any of the recurrent ones

Author(s)

Ignacio Cordón

References

C. M. Grinstead and J. L. Snell. Introduction to Probability. American Mathematical Soc., 2012.

Examples

```
m <- matrix(c(1/2, 1/2, 0,
              1/2, 1/2, 0,
                0, 1/2, 1/2), ncol = 3, byrow = TRUE)
mc <- new("markovchain", states = letters[1:3], transitionMatrix = m)
times <- meanAbsorptionTime(mc)
```

`meanFirstPassageTime` *Mean First Passage Time for irreducible Markov chains*

Description

Given an irreducible (ergodic) markovchain object, this function calculates the expected number of steps to reach other states

Usage

```
meanFirstPassageTime(object, destination)
```

Arguments

`object`	the markovchain object
`destination`	a character vector representing the states respect to which we want to compute the mean first passage time. Empty by default

Details

For an ergodic Markov chain it computes:

- If destination is empty, the average first time (in steps) that takes the Markov chain to go from initial state i to j. (i, j) represents that value in case the Markov chain is given row-wise, (j, i) in case it is given col-wise.
- If destination is not empty, the average time it takes us from the remaining states to reach the states in `destination`

Value

a Matrix of the same size with the average first passage times if destination is empty, a vector if destination is not

Author(s)

Toni Giorgino, Ignacio Cordón

References

C. M. Grinstead and J. L. Snell. Introduction to Probability. American Mathematical Soc., 2012.

Examples

```
m <- matrix(1 / 10 * c(6,3,1,
                       2,3,5,
                       4,1,5), ncol = 3, byrow = TRUE)
mc <- new("markovchain", states = c("s","c","r"), transitionMatrix = m)
meanFirstPassageTime(mc, "r")


# Grinstead and Snell's "Oz weather" worked out example
mOz <- matrix(c(2,1,1,
                2,0,2,
                1,1,2)/4, ncol = 3, byrow = TRUE)

mcOz <- new("markovchain", states = c("s", "c", "r"), transitionMatrix = mOz)
meanFirstPassageTime(mcOz)
```

`meanNumVisits` *Mean num of visits for markovchain, starting at each state*

Description

Given a markovchain object, this function calculates a matrix where the element (i, j) represents the expect number of visits to the state j if the chain starts at i (in a Markov chain by columns it would be the element (j, i) instead)

Usage

```
meanNumVisits(object)
```

Arguments

`object`	the markovchain-class object

Value

a matrix with the expect number of visits to each state

Author(s)

Ignacio Cordón

References

R. Vélez, T. Prieto, Procesos Estocásticos, Librería UNED, 2013

Examples

```
M <- markovchain:::zeros(5)
M[1,1] <- M[5,5] <- 1
M[2,1] <- M[2,3] <- 1/2
M[3,2] <- M[3,4] <- 1/2
M[4,2] <- M[4,5] <- 1/2

mc <- new("markovchain", transitionMatrix = M)
meanNumVisits(mc)
```

---

meanRecurrenceTime *Mean recurrence time*

Description

Computes the expected time to return to a recurrent state in case the Markov chain starts there

Usage

```
meanRecurrenceTime(object)
```

Arguments

object	the markovchain object

Value

For a Markov chain it outputs is a named vector with the expected time to first return to a state when the chain starts there. States present in the vector are only the recurrent ones. If the matrix is ergodic (i.e. irreducible), then all states are present in the output and order is the same as states order for the Markov chain

Author(s)

Ignacio Cordón

References

C. M. Grinstead and J. L. Snell. Introduction to Probability. American Mathematical Soc., 2012.

Examples

```
m <- matrix(1 / 10 * c(6,3,1,
                       2,3,5,
                       4,1,5), ncol = 3, byrow = TRUE)
mc <- new("markovchain", states = c("s","c","r"), transitionMatrix = m)
meanRecurrenceTime(mc)
```

`mergeWith` *Merge two Markov chains by convex combination of their transition matrices*

Description

Builds a new `markovchain` object whose transition matrix is a convex combination of the transition matrices of two existing chains defined on the same state space.

Usage

```
mergeWith(object, other, gamma = 0.5)

## S4 method for signature 'markovchain,markovchain'
mergeWith(object, other, gamma = 0.5)
```

Arguments

`object`	A `markovchain` object.
`other`	A second `markovchain` object, defined on the same set of state names as `object` (see Details for what "same" means here).
`gamma`	A single number in $[0, 1]$: the weight given to `other`. `gamma = 0` returns `object` unchanged (up to storage convention); `gamma = 1` returns `other` unchanged.

Details

For transition matrices P_1 (from `object`) and P_2 (from `other`), and a blending factor $\gamma \in [0, 1]$, the merged transition matrix is

$$P = (1 - \gamma)P_1 + \gamma P_2.$$

States are matched by name, not by position. `object` and `other` must have exactly the same set of state names (as sets – `other`'s states may be in a different order, or `other` may use a different storage convention (`byrow`), and both are handled correctly). Rows and columns of `other`'s transition matrix are realigned to `object`'s state order before combining, and both matrices are converted to row-stochastic form first if needed, so that P_1 and P_2 are always combined entry-for-entry between matching states rather than between matching matrix positions.

This is a deliberate difference from the na\"ive version of this operation, which combines two same-*size* transition matrices positionally and would silently produce a meaningless result if the two chains happened to list their states in a different order (or under a different `byrow` convention) despite describing the same states. Requiring identical state name sets, rather than merely identical size, catches that mismatch as an error instead of propagating it.

Because P_1 and P_2 are both row-stochastic with non-negative entries and $\gamma \in [0, 1]$, P is automatically row-stochastic with non-negative entries: no renormalization is needed (this is the same convexity argument used for `lazyChain`, of which `mergeWith(object, identity_chain, gamma)` is a special case when `other` is an identity chain on the same states).

This function does not require `object` or `other` to be irreducible: merging is meaningful for any two chains on the same state space, including reducible ones. Note, however, that the merged chain's stationary distribution (if any) is generally *not* a combination of π_1 and π_2 in any simple way; it must be recomputed from P directly.

The implementation performs no eigendecomposition and is $O(n^2)$ time and memory for two n-state chains, dominated by realigning `other`'s matrix to `object`'s state order.

Value

A new `markovchain` object, row-stochastic, on the common state names, with transition matrix $P = (1-\gamma)P_1 + \gamma P_2$.

See Also

`lazyChain`, `subchain`

Examples

```
statesNames <- c("a", "b")
mc1 <- new("markovchain", states = statesNames,
  transitionMatrix = matrix(c(0.9, 0.1, 0.1, 0.9), byrow = TRUE, nrow = 2,
                            dimnames = list(statesNames, statesNames)))
# mc2 lists the same two states in the opposite order.
mc2 <- new("markovchain", states = rev(statesNames),
  transitionMatrix = matrix(c(0.5, 0.5, 0.5, 0.5), byrow = TRUE, nrow = 2,
                            dimnames = list(rev(statesNames), rev(statesNames))))
merged <- mergeWith(mc1, mc2, gamma = 0.5)
merged
# States are matched by name: merged["a", "b"] combines mc1["a","b"] with
# mc2["a","b"], not with whatever happened to sit in the same matrix cell.
```

---

mixingTime *Mixing time of a Markov chain*

Description

Estimates the total-variation mixing time of a finite, irreducible, aperiodic (i.e. ergodic) discrete-time Markov chain: the smallest number of steps after which the chain's distribution is within `epsilon` of its stationary distribution, from every possible starting state.

Usage

```
mixingTime(object, epsilon = 0.25, maxIter = 10000L)

## S4 method for signature 'markovchain'
mixingTime(object, epsilon = 0.25, maxIter = 10000L)
```

Arguments

`object`	A `markovchain` object representing a finite, irreducible, aperiodic discrete-time Markov chain.
`epsilon`	A single number strictly between `0` and `1`: the total-variation threshold that counts as "mixed". The classical default `0.25` follows Levin and Peres (2017); it is a conventional choice, not a universal constant.
`maxIter`	A single positive integer: the largest t that will be tried before giving up. This is a safety limit, not a modelling parameter: it exists so that a chain which (numerically) mixes only extremely slowly reports a clear error instead of looping for an unbounded number of iterations.

Details

For a row-stochastic transition matrix P with stationary distribution π, define the worst-case total variation distance after t steps as

$$d(t) = \max_i \tfrac{1}{2} \sum_j |P^t_{ij} - \pi_j|.$$

The mixing time returned is

$$t_{\mathrm{mix}}(\varepsilon) = \min\{t \geq 1 : d(t) \leq \varepsilon\}.$$

Unlike `slem`, `spectralGap` and `impliedTimescales`, `mixingTime()` requires aperiodicity in addition to irreducibility. This is not an arbitrary restriction carried over from another implementation: for a periodic chain, $P^t(i, \cdot)$ never converges to π at all (it keeps cycling through a fixed set of distributions), so $d(t)$ does not go to zero and "the number of steps until $d(t) \leq \varepsilon$" is simply undefined for small enough ε. `slem()` and `spectralGap()` remain meaningful for periodic chains because they summarise the transition matrix's spectrum directly, without reference to a limit that may not exist.

The implementation repeatedly forms $P^{t+1} = P^t P$ and checks $d(t)$ after each multiplication, starting from $t = 1$, until the threshold is met or `maxIter` is reached. Its time complexity is $O(t_{\mathrm{mix}} \cdot n^3)$ and its memory use is $O(n^2)$ for a dense n-state transition matrix: this is a direct, easy-to-audit computation, not an asymptotically optimal one (a repeated-squaring scheme would reach a single large power of P faster, but would not let every intermediate t be checked against `epsilon` along the way). It supports both row- and column-stochastic storage.

Value

A single positive integer, the estimated mixing time $t_{\mathrm{mix}}(\varepsilon)$. For the trivial one-state chain, `0` is returned (it is its own stationary distribution).

References

Levin, D. A. and Peres, Y. (2017). *Markov Chains and Mixing Times*, 2nd edition. American Mathematical Society.

See Also

`slem`, `spectralGap`, `impliedTimescales`, `period`, `is.irreducible`

Examples

```
statesNames <- c("a", "b")
mc <- new("markovchain",
  states = statesNames,
  transitionMatrix = matrix(c(0.7, 0.3, 0.1, 0.9),
    byrow = TRUE, nrow = 2,
    dimnames = list(statesNames, statesNames)))
mixingTime(mc)
mixingTime(mc, epsilon = 0.01) # a tighter threshold needs more steps
```

---

multinomialConfidenceIntervals

A function to compute multinomial confidence intervals of DTMC

Description

Return estimated transition matrix assuming a Multinomial Distribution

Usage

```
multinomialConfidenceIntervals(
  transitionMatrix,
  countsTransitionMatrix,
  confidencelevel = 0.95
)
```

Arguments

`transitionMatrix`
: An estimated transition matrix.

`countsTransitionMatrix`
: Empirical (conts) transition matrix, on which the `transitionMatrix` was performed.

`confidencelevel`
: confidence interval level.

Value

Two matrices containing the confidence intervals.

References

Constructing two-sided simultaneous confidence intervals for multinomial proportions for small counts in a large number of cells. Journal of Statistical Software 5(6) (2000)

See Also

`markovchainFit`

Examples

```
seq<-c("a", "b", "a", "a", "a", "a", "b", "a", "b", "a", "b", "a", "a", "b", "b", "b", "a")
mcfit<-markovchainFit(data=seq,byrow=TRUE)
seqmat<-createSequenceMatrix(seq)
multinomialConfidenceIntervals(mcfit$estimate@transitionMatrix, seqmat, 0.95)
```

---

name *Method to retrieve name of markovchain object*

Description

This method returns the name of a markovchain object

Usage

```
name(object)

## S4 method for signature 'markovchain'
name(object)
```

Arguments

object	A markovchain object

Author(s)

Giorgio Spedicato, Deepak Yadav

Examples

```
statesNames <- c("a", "b", "c")
markovB <- new("markovchain", states = statesNames, transitionMatrix =
               matrix(c(0.2, 0.5, 0.3, 0, 1, 0, 0.1, 0.8, 0.1), nrow = 3,
               byrow = TRUE, dimnames=list(statesNames,statesNames)),
               name = "A markovchain Object"
)
name(markovB)
```

name<- *Method to set name of markovchain object*

Description

This method modifies the existing name of markovchain object

Usage

```
name(object) <- value

## S4 replacement method for signature 'markovchain'
name(object) <- value
```

Arguments

object	A markovchain object
value	New name of markovchain object

Author(s)

Giorgio Spedicato, Deepak Yadav

Examples

```
statesNames <- c("a", "b", "c")
markovB <- new("markovchain", states = statesNames, transitionMatrix =
                matrix(c(0.2, 0.5, 0.3, 0, 1, 0, 0.1, 0.8, 0.1), nrow = 3,
                byrow = TRUE, dimnames=list(statesNames,statesNames)),
                name = "A markovchain Object"
)
name(markovB) <- "dangerous mc"
```

names,markovchain-method

Returns the states for a Markov chain object

Description

Returns the states for a Markov chain object

Usage

```
## S4 method for signature 'markovchain'
names(x)
```

Arguments

x	object we want to return states for

`noofVisitsDist` *Expected fraction of the first N steps spent in each state*

Description

Given the initial state i, returns for every state j the expected fraction of the first `N` steps that the DTMC spends in j.

Usage

```
noofVisitsDist(markovchain,N,state)
```

Arguments

`markovchain`	a markovchain-class object
`N`	number of steps, a positive integer
`state`	the initial state

Details

The value for state j is

$$\frac{1}{N}\sum_{k=1}^{N}(P^k)_{ij} = \frac{E[V_j(N)]}{N},$$

where $V_j(N)$ is the number of visits to j at times $1, \ldots, N$ (the initial state, at time 0, is not counted). The values sum to one, and multiplied by `N` they give the expected numbers of visits. As `N` grows they converge to the stationary distribution for an irreducible chain.

Despite the name of the function, and the title of earlier versions of this page, the result is not the joint distribution of the numbers of visits $(V_1(N), \ldots, V_n(N))$, which the package does not compute (see issue #139).

Value

a named numeric vector with one element per state, summing to one.

Author(s)

Vandit Jain

Examples

```
transMatr<-matrix(c(0.4,0.6,.3,.7),nrow=2,byrow=TRUE)
simpleMc<-new("markovchain", states=c("a","b"),
              transitionMatrix=transMatr,
              name="simpleMc")
noofVisitsDist(simpleMc,5,"a")

# expected numbers of visits during the first 5 steps
5 * noofVisitsDist(simpleMc,5,"a")
```

`normalizedEntropyRate` *Normalized entropy rate of a Markov chain*

Description

The entropy rate of a finite irreducible Markov chain divided by the topological entropy of its graph, a measure of how random the chain is relative to the most random chain with the same possible transitions.

Usage

```
normalizedEntropyRate(object)

## S4 method for signature 'markovchain'
normalizedEntropyRate(object)
```

Arguments

`object`	A `markovchain` object representing a finite, irreducible discrete-time Markov chain.

Details

The ratio H/h_{top} does not depend on the logarithm base, so no `base` argument is needed. By the variational principle (`topologicalEntropy`) it never exceeds one; tiny excursions due to round-off are clipped to $[0, 1]$. When $h_{top} = 0$ the chain has a single possible path, hence entropy rate zero, and the ratio is defined here to be `0`, the convention used by PyDTMC's `entropy_rate_normalized`.

Irreducibility is required, as in `entropyRate`, which is what makes the entropy rate well defined.

Value

A numeric scalar in $[0, 1]$: `0` for deterministic dynamics, `1` when the entropy rate reaches the topological entropy.

See Also

`entropyRate`, `topologicalEntropy`

Examples

```
statesNames <- c("a", "b")
mc <- new("markovchain",
  states = statesNames,
  transitionMatrix = matrix(c(0.7, 0.3, 0.1, 0.9),
    byrow = TRUE, nrow = 2,
    dimnames = list(statesNames, statesNames)))
normalizedEntropyRate(mc)
```

---

ones *Returns an Identity matrix*

Description

Returns an Identity matrix

Usage

```
ones(n)
```

Arguments

n	size of the matrix

Value

a identity matrix

---

period *Various function to perform structural analysis of DTMC*

Description

These functions return absorbing and transient states of the markovchain objects.

Usage

```
period(object)

communicatingClasses(object)

recurrentClasses(object)

transientClasses(object)
```

```
transientStates(object)

recurrentStates(object)

absorbingStates(object)

canonicForm(object)
```

Arguments

object	A markovchain object.

Value

`period` returns a integer number corresponding to the periodicity of the Markov chain (if it is irreducible)

`absorbingStates` returns a character vector with the names of the absorbing states in the Markov chain

`communicatingClasses` returns a list in which each slot contains the names of the states that are in that communicating class

`recurrentClasses` analogously to `communicatingClasses`, but with recurrent classes

`transientClasses` analogously to `communicatingClasses`, but with transient classes

`transientStates` returns a character vector with all the transient states for the Markov chain

`recurrentStates` returns a character vector with all the recurrent states for the Markov chain

`canonicForm` returns the Markov chain reordered by a permutation of states so that we have blocks submatrices for each of the recurrent classes and a collection of rows in the end for the transient states

Author(s)

Giorgio Alfredo Spedicato, Ignacio Cordón

References

Feres, Matlab listing for markov chain.

See Also

markovchain

Examples

```
statesNames <- c("a", "b", "c")
mc <- new("markovchain", states = statesNames, transitionMatrix =
          matrix(c(0.2, 0.5, 0.3,
                   0,   1,   0,
                   0.1, 0.8, 0.1), nrow = 3, byrow = TRUE,
                 dimnames = list(statesNames, statesNames))
```

```
          )

communicatingClasses(mc)
recurrentClasses(mc)
recurrentClasses(mc)
absorbingStates(mc)
transientStates(mc)
recurrentStates(mc)
canonicForm(mc)

# periodicity analysis
A <- matrix(c(0, 1, 0, 0, 0.5, 0, 0.5, 0, 0, 0.5, 0, 0.5, 0, 0, 1, 0),
            nrow = 4, ncol = 4, byrow = TRUE)
mcA <- new("markovchain", states = c("a", "b", "c", "d"),
           transitionMatrix = A,
           name = "A")

is.irreducible(mcA) #true
period(mcA) #2

# periodicity analysis
B <- matrix(c(0, 0, 1/2, 1/4, 1/4, 0, 0,
                    0, 0, 1/3, 0, 2/3, 0, 0,
                    0, 0, 0, 0, 0, 1/3, 2/3,
                    0, 0, 0, 0, 0, 1/2, 1/2,
                    0, 0, 0, 0, 0, 3/4, 1/4,
                    1/2, 1/2, 0, 0, 0, 0, 0,
                    1/4, 3/4, 0, 0, 0, 0, 0), byrow = TRUE, ncol = 7)
mcB <- new("markovchain", transitionMatrix = B)
period(mcB)
```

`populationGeneticsModel` *Build a population-genetics Markov chain (Moran or Wright-Fisher)*

Description

Constructs the Markov chain tracking the number of copies of a mutant allele in a population of `n` individuals under mutation and viability selection, for either the Moran or the Wright-Fisher model of population genetics.

Usage

```
populationGeneticsModel(
  model = c("moran", "wright-fisher"),
  n,
  s = 0,
  u = 1e-09,
```

```
  v = 1e-09,
  states = NULL
)
```

Arguments

`model`	Either `"moran"` or `"wright-fisher"`, selecting which reproduction scheme generates the transition matrix.
`n`	A single integer of at least `2`: the (constant) population size. The chain has `n + 1` states, $0, 1, \ldots, n$, the possible mutant-allele counts.
`s`	A single finite number greater than `-1`: the selection coefficient. The mutant allele's fitness relative to the wild-type is $1 + s$ (`s = 0` is neutral drift, `s > 0` favours the mutant, `-1 < s < 0` disfavours it).
`u`	A single number in $[0, 1]$: the backward mutation rate, mutant to wild-type. Defaults to `1e-9`, matching the effectively mutation-free chain conventionally used for the neutral/absorbing case.
`v`	A single number in $[0, 1]$: the forward mutation rate, wild-type to mutant. Defaults to `1e-9`.
`states`	An optional character vector of `n + 1` state names. Defaults to `as.character(0:n)`.

Details

Moran model. At each step one individual is chosen to reproduce (with probability proportional to its type's relative fitness, mutant fitness $r = 1 + s$ against wild-type fitness 1) and one individual, chosen uniformly at random, dies and is replaced by the offspring, which mutates with probability `u` or `v` depending on the parent's type. From interior state i ($0 < i < n$), writing $r_i = (1 + s)i$ and $m_i = n - i$,

$$P_{i,i-1} = \frac{i}{n} \cdot \frac{r_i v + m_i(1 - u)}{r_i + m_i}, \qquad P_{i,i+1} = \frac{m_i}{n} \cdot \frac{r_i(1 - u) + m_i v}{r_i + m_i},$$

with P_{ii} the remainder.

Wright-Fisher model. Generations are discrete and non-overlapping: from interior state i, the mutant-allele frequency $k = i/n$ is first updated for mutation, $p = k(1 - u) + (1 - k)v$, then for viability selection, $p' = \min\big(p(1 + s)/(1 + ps),\, 1\big)$, and the next generation's mutant count is $\mathrm{Binomial}(n, p')$-distributed: $P_{ij} = \binom{n}{j} p'^{\,j}(1 - p')^{n-j}$.

Both constructions follow **PyDTMC**'s `population_genetics_model`, with one deliberate difference: this function uses mutant relative fitness $1 + s$ in *both* models (PyDTMC's Moran implementation instead uses $1 - s$, so that a positive `s` there favours the wild-type rather than the mutant, the opposite of its own Wright-Fisher convention and of the usual textbook one). `s = 0` (neutral drift) and the mutation-rate terms are unaffected by this choice and match PyDTMC exactly; away from `s = 0` the two packages' Moran chains differ by construction, by design, to keep `s`'s meaning consistent between `"moran"` and `"wright-fisher"` within this package. As `u, v` shrink to `0`, both reduce to the classical drift-only chains with absorbing loss/fixation states, going back to Wright (1931) and Moran (1958); see Ewens (2004) for a modern textbook treatment of both.

Value

A new, row-stochastic `markovchain` object on `n + 1` states. States `"0"` and `as.character(n)` (loss and fixation of the mutant allele) are absorbing, following the standard textbook treatment of both models; every interior state has a transition row determined by `model`, `s`, `u` and `v` as detailed below.

References

Wright, S. (1931). Evolution in Mendelian populations. *Genetics*, 16(2), 97-159.

Moran, P. A. P. (1958). Random processes in genetics. *Mathematical Proceedings of the Cambridge Philosophical Society*, 54(1), 60-71.

Ewens, W. J. (2004). *Mathematical Population Genetics I: Theoretical Introduction* (2nd ed.). Springer.

See Also

`birthDeath`, `gamblersRuin`

Examples

```
# Neutral drift (s = 0): absorption ("fixation") probability of the
# mutant allele from state i equals i/n, the classical result.
neutral <- populationGeneticsModel(model = "moran", n = 8, s = 0)
ap <- absorptionProbabilities(neutral)
ap["4", "8"] # close to 4/8 = 0.5

# Positive selection increases the fixation probability from any
# interior starting count.
favoured <- populationGeneticsModel(model = "wright-fisher", n = 8, s = 0.5)
absorptionProbabilities(favoured)["4", "8"]
```

---

predictHommc *Simulate a higher order multivariate markovchain*

Description

This function provides a prediction of states for a higher order multivariate markovchain object

Usage

```
predictHommc(hommc,t,init)
```

Arguments

hommc	a hommc-class object
t	no of iterations to predict
init	matrix of previous states size of which depends on hommc

Details

The user is required to provide a matrix of giving n previous coressponding every categorical sequence. Dimensions of the init are s X n, where s is number of categorical sequences and n is order of the homc. The last column of `init` holds the most recent state of each sequence.

At each step the next state of sequence j is drawn from $\sum_k \sum_h \lambda_{jkh} P_h^{(jk)} x_{t-h+1}^{(k)}$, where $x_{t-h+1}^{(k)}$ is the state of sequence k $h-1$ steps before the current one (Ching et al., 2008). The matrices are read by column (`P[to, from]`), as returned by `fitHighOrderMultivarMC`, or by row when `byrow = TRUE`, and all sequences are drawn from the same past before it is updated.

Value

The function returns a matrix of size s X t displaying t predicted states in each row coressponding to every categorical sequence.

Author(s)

Vandit Jain

`predictiveDistribution` *predictiveDistribution*

Description

The function computes the probability of observing a new data set, given a data set

Usage

```
predictiveDistribution(stringchar, newData, hyperparam = matrix())
```

Arguments

`stringchar`	This is the data using which the Bayesian inference is performed.
`newData`	This is the data whose predictive probability is computed.
`hyperparam`	This determines the shape of the prior distribution of the parameters. If none is provided, default value of 1 is assigned to each parameter. This must be of size kxk where k is the number of states in the chain and the values should typically be non-negative integers.

Details

The underlying method is Bayesian inference. The probability is computed by averaging the likelihood of the new data with respect to the posterior. Since the method assumes conjugate priors, the result can be represented in a closed form (see the vignette for more details), which is what is returned.

Value

The log of the probability is returned.

Author(s)

Sai Bhargav Yalamanchi

References

Inferring Markov Chains: Bayesian Estimation, Model Comparison, Entropy Rate, and Out-of-Class Modeling, Christopher C. Strelioff, James P. Crutchfield, Alfred Hubler, Santa Fe Institute

Yalamanchi SB, Spedicato GA (2015). Bayesian Inference of First Order Markov Chains. R package version 0.2.5

See Also

markovchainFit

Examples

```
sequence<- c("a", "b", "a", "a", "a", "a", "b", "a", "b", "a", "b", "a", "a",
             "b", "b", "b", "a")
hyperMatrix<-matrix(c(1, 2, 1, 4), nrow = 2,dimnames=list(c("a","b"),c("a","b")))
predProb <- predictiveDistribution(sequence[1:10], sequence[11:17], hyperparam =hyperMatrix )
hyperMatrix2<-hyperMatrix[c(2,1),c(2,1)]
predProb2 <- predictiveDistribution(sequence[1:10], sequence[11:17], hyperparam =hyperMatrix2 )
predProb2==predProb
```

---

preproglucacon *Preprogluccacon DNA protein bases sequences*

Description

Sequence of bases for preproglucacon DNA protein

Usage

```
data(preproglucacon)
```

Format

A data frame with 1572 observations on the following 2 variables.

`V1` a numeric vector, showing original coding

`preproglucacon` a character vector, showing initial of DNA bases (Adenine, Cytosine, Guanine, Thymine)

Source

Avery Henderson

References

Averuy Henderson, Fitting markov chain models on discrete time series such as DNA sequences

Examples

```
data(preproglucacon)
preproglucaconMc<-markovchainFit(data=preproglucacon$preproglucacon)
```

`priorDistribution` *priorDistribution*

Description

Function to evaluate the prior probability of a transition matrix. It is based on conjugate priors and therefore a Dirichlet distribution is used to model the transitions of each state.

Usage

```
priorDistribution(transMatr, hyperparam = matrix())
```

Arguments

`transMatr`	The transition matrix whose probability is the parameter of interest.
`hyperparam`	The hyperparam matrix (optional). If not provided, a default value of 1 is assumed for each and therefore the resulting probability distribution is uniform.

Details

The states (dimnames) of the transition matrix and the hyperparam may be in any order.

Value

The log of the probabilities for each state is returned in a numeric vector. Each number in the vector represents the probability (log) of having a probability transition vector as specified in corresponding the row of the transition matrix.

Note

This function can be used in conjunction with inferHyperparam. For example, if the user has a prior data set and a prior transition matrix, he can infer the hyperparameters using inferHyperparam and then compute the probability of their prior matrix using the inferred hyperparameters with priorDistribution.

Author(s)

Sai Bhargav Yalamanchi, Giorgio Spedicato

References

Yalamanchi SB, Spedicato GA (2015). Bayesian Inference of First Order Markov Chains. R package version 0.2.5

See Also

predictiveDistribution, inferHyperparam

Examples

```
priorDistribution(matrix(c(0.5, 0.5, 0.5, 0.5),
                  nrow = 2,
                  dimnames = list(c("a", "b"), c("a", "b"))),
                  matrix(c(2, 2, 2, 2),
                  nrow = 2,
                  dimnames = list(c("a", "b"), c("a", "b"))))
```

---

probabilityatT *Calculating probability from a ctmc object*

Description

This function returns the probability of every state at time t under different conditions

Usage

```
probabilityatT(C,t,x0,useRCpp)
```

Arguments

C	A CTMC S4 object
t	final time t
x0	initial state
useRCpp	logical whether to use RCpp implementation

Details

The initial state is not mandatory, In case it is not provided, function returns a matrix of transition function at time t else it returns vector of probaabilities of transition to different states if initial state was x0

Value

returns a vector or a matrix in case x0 is provided or not respectively.

Author(s)

Vandit Jain

References

INTRODUCTION TO STOCHASTIC PROCESSES WITH R, ROBERT P. DOBROW, Wiley

Examples

```
states <- c("a","b","c","d")
byRow <- TRUE
gen <- matrix(data = c(-1, 1/2, 1/2, 0, 1/4, -1/2, 0, 1/4, 1/6, 0, -1/3, 1/6, 0, 0, 0, 0),
nrow = 4,byrow = byRow, dimnames = list(states,states))
ctmc <- new("ctmc",states = states, byrow = byRow, generator = gen, name = "testctmc")
probabilityatT(ctmc,1,useRCpp = TRUE)
```

---

rain *Alofi island daily rainfall*

Description

Rainfall measured in Alofi Island

Usage

```
data(rain)
```

Format

A data frame with 1096 observations on the following 2 variables.

`V1` a numeric vector, showing original coding

`rain` a character vector, showing daily rainfall millilitres brackets

Source

Avery Henderson

References

Avery Henderson, Fitting markov chain models on discrete time series such as DNA sequences

Examples

```
data(rain)
rainMc<-markovchainFit(data=rain$rain)
```

`randomMarkovChain` *Random Markov chain*

Description

Generates a Markov chain whose transition probabilities are drawn at random, optionally with a given number of zero probabilities and with some probabilities fixed in advance.

Usage

```
randomMarkovChain(
  n,
  states = NULL,
  zeros = 0L,
  mask = NULL,
  byrow = TRUE,
  seed = NULL,
  name = "Random Markov chain"
)
```

Arguments

`n`	The number of states. It can be omitted when `states` is given.
`states`	An optional character vector of `n` state names. Defaults to `as.character(1:n)`.
`zeros`	The number of transition probabilities, among those not fixed by `mask`, that are set to zero. Every row whose probabilities are not all fixed keeps at least one positive free entry, which bounds `zeros` from above; a larger value is an error.
`mask`	An optional `n` x `n` matrix of fixed transition probabilities: `NA` marks the entries to draw at random, any other value (in $[0, 1]$) is kept as it is. In each row the fixed values must not sum to more than one. A row whose fixed values sum to one gets zero in its `NA` entries; in any other row the free entries share the remaining probability. With `byrow = FALSE` the mask is read by columns, like the transition matrix.
`byrow`	Whether the transition matrix of the result is stored by rows (the default) or by columns.
`seed`	An optional whole number. When given, the chain is generated after `set.seed(seed)` and the caller's random number stream is restored afterwards, so the result is reproducible without affecting later random draws; when `NULL` (the default), the current stream is used, so `set.seed()` beforehand works as usual.
`name`	The `name` slot of the result.

Details

The algorithm follows `MarkovChain.random()` of PyDTMC. In each row not completely fixed by `mask`, one free entry, chosen at random, is reserved to be positive. The `zeros` zero entries are then

chosen at random among the remaining free entries of the whole matrix, the other free entries are drawn from a uniform distribution on $(0, 1)$, and the free entries of each row are rescaled so that, together with the fixed ones, they sum to one. The rows are normalised uniforms, which is not the uniform distribution on the simplex; use `dirichletChain` or draw the rows yourself if that matters.

Value

A `markovchain` object with `n` states.

See Also

`dirichletChain`, `identityChain`

Examples

```
randomMarkovChain(4, seed = 1)
# 6 of the 16 transition probabilities are zero
sum(randomMarkovChain(4, zeros = 6, seed = 1)@transitionMatrix == 0)
# state "b" moves to "a" with probability 0.5; the rest is random
m <- matrix(NA, 3, 3)
m[2, 1] <- 0.5
randomMarkovChain(states = c("a", "b", "c"), mask = m, seed = 1)
```

`rctmc` *rctmc*

Description

The function generates random CTMC transitions as per the provided generator matrix.

Usage

```
rctmc(n, ctmc, initDist = numeric(), T = 0, include.T0 = TRUE,
  out.type = "list")
```

Arguments

`n`	The number of samples to generate.
`ctmc`	The CTMC S4 object.
`initDist`	The initial distribution of states.
`T`	The time up to which the simulation runs (all transitions after time T are not returned).
`include.T0`	Flag to determine if start state is to be included.
`out.type`	"list" or "df"

Details

In order to use the T0 argument, set n to Inf.

Value

Based on out.type, a list or a data frame is returned. The returned list has two elements - a character vector (states) and a numeric vector (indicating time of transitions). The data frame is similarly structured.

Author(s)

Sai Bhargav Yalamanchi

References

Introduction to Stochastic Processes with Applications in the Biosciences (2013), David F. Anderson, University of Wisconsin at Madison

See Also

generatorToTransitionMatrix,ctmc-class

Examples

```
energyStates <- c("sigma", "sigma_star")
byRow <- TRUE
gen <- matrix(data = c(-3, 3, 1, -1), nrow = 2,
              byrow = byRow, dimnames = list(energyStates, energyStates))
molecularCTMC <- new("ctmc", states = energyStates,
                     byrow = byRow, generator = gen,
                     name = "Molecular Transition Model")

statesDist <- c(0.8, 0.2)
rctmc(n = Inf, ctmc = molecularCTMC, T = 1)
rctmc(n = 5, ctmc = molecularCTMC, initDist = statesDist, include.T0 = FALSE)
```

---

redistribute *Evolution of a distribution over time*

Description

Propagates an initial distribution of the states of a discrete-time Markov chain forward in time and returns the whole trajectory of distributions.

Usage

```
redistribute(object, steps, initial = NULL, lastOnly = FALSE)

## S4 method for signature 'markovchain'
redistribute(object, steps, initial = NULL, lastOnly = FALSE)
```

Arguments

`object`	A `markovchain` object.
`steps`	A single non-negative whole number: the number of steps to propagate. With `steps = 0` only the initial distribution is returned.
`initial`	The initial distribution. Either `NULL` (default), for the uniform distribution over the states; a single state name, for a point mass on that state; or a numeric vector of non-negative probabilities summing to one. A named numeric vector is matched to the states by name, an unnamed one by position.
`lastOnly`	Logical. If `TRUE`, only the distribution after `steps` steps is returned. Defaults to `FALSE`.

Details

If μ_0 is the initial distribution (a row vector) and P the row-stochastic transition matrix, the distribution after t steps is

$$\mu_t = \mu_{t-1} P = \mu_0 P^t.$$

The implementation propagates the vector step by step, at a cost of $O(n^2)$ per step for a dense n-state chain, instead of forming P^t; this is what makes the whole trajectory available at no extra cost. Both row- and column-stochastic storage are supported. Each distribution is renormalized after every step to prevent round-off from accumulating over long horizons.

This mirrors PyDTMC's `redistribute()`, with the same defaults (uniform initial distribution, output including the initial one). For chains that converge, the rows approach the stationary distribution (see `steadyStates`); for periodic chains they do not, which is the expected behaviour and not an error.

Value

If `lastOnly = FALSE` (default), a numeric matrix with `steps + 1` rows and one column per state: row t (labelled `"t"`, from `"0"`) holds the distribution after t steps, so the first row is the initial distribution. Otherwise, a named numeric vector with the distribution after `steps` steps.

See Also

`steadyStates`, `mixingTime`, `autoplot.markovchain`

Examples

```
statesNames <- c("a", "b")
mc <- new("markovchain",
  states = statesNames,
  transitionMatrix = matrix(c(0.7, 0.3, 0.1, 0.9),
    byrow = TRUE, nrow = 2,
    dimnames = list(statesNames, statesNames)))
redistribute(mc, steps = 5, initial = "a")
redistribute(mc, steps = 50, initial = c(a = 0.2, b = 0.8), lastOnly = TRUE)
```

`relaxationTime` *Relaxation time of a Markov chain*

Description

The relaxation time $1/\text{gap}$ of a finite irreducible discrete-time Markov chain, the reciprocal of its spectral gap.

Usage

```
relaxationTime(object)

## S4 method for signature 'markovchain'
relaxationTime(object)
```

Arguments

`object`	A `markovchain` object representing a finite, irreducible discrete-time Markov chain.

Details

Following Levin and Peres (2017, Section 12.2) the relaxation time is $t_{rel} = 1/\gamma$, with $\gamma = 1 - \text{SLEM}$ the spectral gap of `spectralGap`. It is the quantity PyDTMC calls `relaxation_rate`, even if it is a time, not a rate. The two are the same number.

PyDTMC's `mixing_rate` is $-1/\log(\text{SLEM})$, i.e. the timescale of the slowest non-trivial mode, which is already returned as the first element (`"tau2"`) of `impliedTimescales`. It is therefore not repeated as a separate function.

Value

A positive numeric scalar, in number of steps: `Inf` for a periodic chain (spectral gap zero), `1` for the trivial one-state chain.

References

Levin, D. A. and Peres, Y. (2017). *Markov Chains and Mixing Times*, 2nd edition. American Mathematical Society.

See Also

`spectralGap`, `slem`, `impliedTimescales`, `mixingTime`

Examples

```
statesNames <- c("a", "b")
mc <- new("markovchain",
  states = statesNames,
  transitionMatrix = matrix(c(0.7, 0.3, 0.1, 0.9),
    byrow = TRUE, nrow = 2,
    dimnames = list(statesNames, statesNames)))
relaxationTime(mc)
```

`rmarkovchain` *Function to generate a sequence of states from homogeneous or non-homogeneous Markov chains.*

Description

Provided any `markovchain` or `markovchainList` objects, it returns a sequence of states coming from the underlying stationary distribution.

Usage

```
rmarkovchain(
  n,
  object,
  what = "data.frame",
  useRCpp = TRUE,
  parallel = FALSE,
  num.cores = NULL,
  ...
)
```

Arguments

`n`	Sample size
`object`	Either a `markovchain` or a `markovchainList` object
`what`	It specifies whether either a `data.frame` or a `matrix` (each rows represent a simulation) or a `list` is returned.
`useRCpp`	Boolean. Should RCpp fast implementation being used? Default is yes.
`parallel`	Boolean. Should parallel implementation being used? Default is yes.
`num.cores`	Number of threads used when `parallel = TRUE`. If `NULL` (the default) the thread count is read from `getOption("RcppParallel.numThreads")`, then `getOption("Ncpus")`, then `RCPP_PARALLEL_NUM_THREADS`, then `OMP_NUM_THREADS`, falling back to `min(2, cores)` as CRAN policy requires. Previous versions defaulted to `parallel::detectCores() - 1`, which could grab all available cores unexpectedly.
`...`	additional parameters passed to the internal sampler

Details

When a homogeneous process is assumed (markovchain object) a sequence is sampled of size n. When a non - homogeneous process is assumed, n samples are taken but the process is assumed to last from the begin to the end of the non-homogeneous markov process.

Value

Character Vector, data.frame, list or matrix

Note

Check the type of input

Author(s)

Giorgio Spedicato

References

A First Course in Probability (8th Edition), Sheldon Ross, Prentice Hall 2010

See Also

markovchainFit, markovchainSequence

Examples

```
# define the markovchain object
statesNames <- c("a", "b", "c")
mcB <- new("markovchain", states = statesNames,
   transitionMatrix = matrix(c(0.2, 0.5, 0.3, 0, 0.2, 0.8, 0.1, 0.8, 0.1),
   nrow = 3, byrow = TRUE, dimnames = list(statesNames, statesNames)))

# show the sequence
outs <- rmarkovchain(n = 100, object = mcB, what = "list")


#define markovchainList object
statesNames <- c("a", "b", "c")
mcA <- new("markovchain", states = statesNames, transitionMatrix =
   matrix(c(0.2, 0.5, 0.3, 0, 0.2, 0.8, 0.1, 0.8, 0.1), nrow = 3,
   byrow = TRUE, dimnames = list(statesNames, statesNames)))
mcB <- new("markovchain", states = statesNames, transitionMatrix =
   matrix(c(0.2, 0.5, 0.3, 0, 0.2, 0.8, 0.1, 0.8, 0.1), nrow = 3,
   byrow = TRUE, dimnames = list(statesNames, statesNames)))
mcC <- new("markovchain", states = statesNames, transitionMatrix =
   matrix(c(0.2, 0.5, 0.3, 0, 0.2, 0.8, 0.1, 0.8, 0.1), nrow = 3,
   byrow = TRUE, dimnames = list(statesNames, statesNames)))
mclist <- new("markovchainList", markovchains = list(mcA, mcB, mcC))

# show the list of sequence
```

```
rmarkovchain(100, mclist, "list")
```

`rouwenhorst` *Discretize an AR(1) process into a Markov chain (Rouwenhorst's method)*

Description

Approximates the stationary first-order autoregressive process

$$y_t = (1-\rho)\alpha + \rho y_{t-1} + \varepsilon_t, \qquad \varepsilon_t \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma^2)$$

by a finite-state Markov chain, following Rouwenhorst (1995). Unlike `tauchen`, no grid-width parameter is needed and the method remains accurate for `rho` close to ± 1.

Usage

```
rouwenhorst(alpha, sigma, rho, size)
```

Arguments

`alpha`	A single finite number: the unconditional mean of the process.
`sigma`	A single finite positive number: the standard deviation of the innovation ε_t.
`rho`	A single number in $(-1, 1)$: the autocorrelation (persistence) of the process.
`size`	A single integer of at least 2: the number of grid points (states) of the discretized chain.

Details

The grid is $n =$ `size` evenly spaced points spanning $[\alpha - \psi,\ \alpha + \psi]$ with $\psi = \sigma_y\sqrt{n-1}$, where $\sigma_y = \sigma/\sqrt{1-\rho^2}$ is the process's unconditional standard deviation: this particular width (rather than a fixed multiple of σ_y as in `tauchen`) is what the method needs in order to match the AR(1)'s variance and first-order autocorrelation exactly at every `size`, including for `rho` near ± 1. The transition matrix is built recursively. Let $\theta = (1+\rho)/2$ and, for two states,

$$\Theta_2 = \begin{pmatrix} \theta & 1-\theta \\ 1-\theta & \theta \end{pmatrix}.$$

For m states ($2 < m \le n$), form the $m \times m$ matrix

$$\Theta_m = \theta \begin{pmatrix} \Theta_{m-1} & 0 \\ 0 & 0 \end{pmatrix} + (1-\theta) \begin{pmatrix} 0 & \Theta_{m-1} \\ 0 & 0 \end{pmatrix} + (1-\theta) \begin{pmatrix} 0 & 0 \\ \Theta_{m-1} & 0 \end{pmatrix} + \theta \begin{pmatrix} 0 & 0 \\ 0 & \Theta_{m-1} \end{pmatrix},$$

then divide every interior row (all but the first and last) by 2 to restore row-stochasticity, since those rows receive contributions from two of the four corner blocks above.

Rouwenhorst's method reproduces the AR(1)'s unconditional variance and lag-1 autocorrelation ρ exactly, for every `size` (Kopecky and Suen (2010) find it outperforms `tauchen` and several other methods across the persistence range typically seen in quarterly macroeconomic and actuarial time series, e.g. discretized short-rate or inflation processes for reserving and ALM work).

Value

A named list with two elements, `chain` and `states`, in the same form as returned by `tauchen`.

References

Rouwenhorst, K. G. (1995). Asset pricing implications of equilibrium business cycle models. In T. F. Cooley (Ed.), *Frontiers of Business Cycle Research*, 294-330. Princeton University Press.

Kopecky, K. A. and Suen, R. M. H. (2010). Finite state Markov-chain approximations to highly persistent processes. *Review of Economic Dynamics*, 13(3), 701-714.

See Also

`tauchen`

Examples

```
out <- rouwenhorst(alpha = 0, sigma = 1, rho = 0.9, size = 5)
out$states
pi <- as.numeric(steadyStates(out$chain))
sum(pi * (out$states - sum(pi * out$states))^2) # matches 1/(1-rho^2) closely
1 / (1 - 0.9^2)
```

---

sales *Sales Demand Sequences*

Description

Sales demand sequences of five products (A, B, C, D, E). Each row corresponds to a sequence. First row corresponds to Sequence A, Second row to Sequence B and so on.

Usage

```
data("sales")
```

Format

An object of class `matrix` (inherits from `array`) with 269 rows and 5 columns.

Details

The example can be used to fit High order multivariate markov chain.

Examples

```
data("sales")
# fitHighOrderMultivarMC(seqMat = sales, order = 2, Norm = 2)
```

`selectOrder` *Select the order of a Markov chain by information criteria*

Description

Fits by maximum likelihood fully parameterized Markov chains of order $0, 1, \ldots,$ `maxOrder` to an empirical sequence (or to a list of independent sequences), all on the same observations, and selects the order that minimizes the BIC or the AIC (Tong, 1975; Katz, 1981).

Usage

```
selectOrder(
  sequence,
  maxOrder = 3,
  criterion = c("BIC", "AIC"),
  start = NULL,
  parameters = c("full", "observed")
)
```

Arguments

`sequence`	An empirical sequence of states (a vector coercible to character, without missing values), or a list of such sequences.
`maxOrder`	The highest order considered, a non-negative integer.
`criterion`	The criterion used to select the order: `"BIC"` (default) or `"AIC"`.
`start`	Index of the first observation of each sequence entering the likelihood, at least `maxOrder + 1` (the default).
`parameters`	How the free parameters are counted: `"full"` (default) or `"observed"`, see Details.

Details

A Markov chain of order k on r states has a transition probability for each context (the k previous states) and next state; order 0 is independence. Its maximum likelihood estimates are the observed transition frequencies, and its log-likelihood is $\sum N(c, j) \log\{N(c, j)/N(c)\}$, where $N(c, j)$ counts context c followed by state j.

Information criteria are only comparable when every model is evaluated on the same observations. An order-k chain can only predict an observation from position $k + 1$ onwards, so all orders are evaluated on the observations from `start` (by default `maxOrder + 1`) to the end of each sequence, the earlier ones being only used as contexts. Berchtold and Raftery (2002), for instance, condition on the first 14 observations (`start = 15`). For a list of sequences the counts are pooled, the observations before `start` of each sequence are only used as contexts, and no context crosses from one sequence to the next.

With `parameters = "full"` (the default) an order-k chain has $r^k(r - 1)$ free parameters, as in Tong (1975) and Katz (1981). With `parameters = "observed"` only the probabilities that are not

estimated as zero are counted, that is the number of distinct next states minus one summed over the observed contexts; this is the convention of Berchtold and Raftery (2002), which is less penalizing when many transitions are never observed. In both cases BIC uses the number of observations entering the likelihood.

The table also reports the likelihood-ratio statistic of each order against the previous one, $G = 2(\ell_k - \ell_{k-1})$, with degrees of freedom equal to the difference in the number of parameters and an asymptotic chi-squared p-value (Anderson and Goodman, 1957). These tests are not adjusted for multiplicity, and they and the criteria become unreliable when the number of contexts r^k is not small compared with the number of observations: BIC is consistent for the order (Csiszar and Shields, 2000), whereas AIC tends to select too high an order in long sequences (Katz, 1981).

Value

A list with components

`order`	the order selected by `criterion`
`criterion`	the criterion used
`table`	a data frame with one row per order: `order`, `logLik`, `npar`, `AIC`, `BIC`, and the likelihood-ratio test against the previous order (`LR`, `df`, `p.value`; `NA` for order 0)
`nobs`	number of observations entering the likelihood
`start`, `parameters`	as used

References

Anderson, T. W. and Goodman, L. A. (1957). Statistical inference about Markov chains. The Annals of Mathematical Statistics, 28(1), 89-110.

Tong, H. (1975). Determination of the order of a Markov chain by Akaike's information criterion. Journal of Applied Probability, 12(3), 488-497.

Katz, R. W. (1981). On some criteria for estimating the order of a Markov chain. Technometrics, 23(3), 243-249.

Csiszar, I. and Shields, P. C. (2000). The consistency of the BIC Markov order estimator. The Annals of Statistics, 28(6), 1601-1619.

Berchtold, A. and Raftery, A. E. (2002). The mixture transition distribution model for high-order Markov chains and non-Gaussian time series. Statistical Science, 17(3), 328-356.

See Also

`assessOrder`, `verifyMarkovProperty`, `fitHigherOrder`, `fitMTD`, `higherOrderLogLik`

Examples

```
# Alofi rainfall: three states
data(rain)
selectOrder(rain$rain, maxOrder = 3)$table

# Koeberg wind directions with the conventions of Berchtold and Raftery (2002)
```

```
wind <- read.csv(system.file("extdata", "koeberg_wind.csv",
                             package = "markovchain"))$state
sel <- selectOrder(wind, maxOrder = 3, start = 15, parameters = "observed")
sel$order
sel$table
```

`sensitivity` *Sensitivity of the stationary distribution to a state's transition row*

Description

Computes, for a finite, irreducible discrete-time Markov chain, the coefficients needed to obtain the first-order change in the stationary distribution caused by an infinitesimal perturbation of one row of the transition matrix.

Usage

```
sensitivity(object, state)

## S4 method for signature 'markovchain'
sensitivity(object, state)
```

Arguments

`object`	A `markovchain` object representing a finite, irreducible discrete-time Markov chain.
`state`	A single state name (character) or state index (single positive integer), identifying the row k of the transition matrix to be perturbed.

Details

A single entry p_{kl} of a stochastic matrix cannot be perturbed on its own without leaving row k: some other entry (or entries) of that row must move to compensate, so that the row still sums to one. Any admissible perturbation of row k is therefore a direction vector $d \in \mathbb{R}^n$ with $\sum_l d_l = 0$, giving the perturbed matrix $P(\varepsilon) = P + \varepsilon\, e_k d^{\mathsf{T}}$ for small ε. This function returns the $n \times n$ matrix S such that, for *every* such d and every state j,

$$\left.\frac{d\pi_j}{d\varepsilon}\right|_{\varepsilon=0} = \sum_l d_l\, S_{lj} = \left(d^{\mathsf{T}} S\right)_j .$$

Closed form. Let $Z = (I - P + \mathbf{1}\pi^{\mathsf{T}})^{-1}$ be the fundamental matrix already used by `kemenyConstant`. Then

$$S_{lj} = \pi_k \left(Z_{lj} - \pi_j\right).$$

This particular centering (subtracting π_j, the same constant for every row l) is what makes S usable directly with *any* zero-sum direction d, because $\sum_l d_l \pi_j = \pi_j \sum_l d_l = 0$ drops out of the sum above – adding any other per-column constant to S would give the same directional derivatives, but

this one has the convenient side effect that `sensitivity(object,state)[state, ]` is the sensitivity of "leaving row `state` unchanged", which is informative on its own (it need not be zero: the *direction* $d = e_{\text{state}}$ is generally not itself a valid zero-sum perturbation by itself, only differences of rows are).

The common two-state case. The usual textbook question – "increase $p_{k,\text{to}}$ by ε, decrease $p_{k,\text{from}}$ by ε, how does π move?" – is answered by taking the difference of two rows of S:

$$\left.\frac{d\pi}{d\varepsilon}\right|_{\varepsilon=0} = S_{\text{to},\cdot} - S_{\text{from},\cdot}.$$

See the second example below, which checks this against a direct finite-difference recomputation of the stationary distribution.

Only irreducibility is required, not aperiodicity: Z and π are well defined for any irreducible chain regardless of periodicity.

The implementation calls `steadyStates` once and then solves one dense linear system for Z; both are $O(n^3)$ time and $O(n^2)$ memory for a dense n-state transition matrix, the same cost as `kemenyConstant`. It supports both row- and column-stochastic storage; S is always returned with rows/columns indexed by state name in the chain's own state order.

Value

An $n \times n$ numeric matrix S, with both dimensions named after `states(object)`. Row l of S corresponds to the perturbation direction "increase p_{kl}" (paired with a compensating decrease elsewhere in row k); column j corresponds to the affected stationary probability π_j. See Details for how to read individual entries.

References

Schweitzer, P. J. (1968). Perturbation theory and finite Markov chains. *Journal of Applied Probability*, 5(2), 401-413.

Meyer, C. D. (1980). The condition of a finite Markov chain and perturbation bounds for the limiting probabilities. *SIAM Journal on Algebraic and Discrete Methods*, 1(3), 273-283.

Cho, G. E. and Meyer, C. D. (2001). Comparison of perturbation bounds for the stationary distribution of a Markov chain. *Linear Algebra and its Applications*, 335(1-3), 137-150.

See Also

`kemenyConstant`, `steadyStates`, `is.irreducible`

Examples

```
statesNames <- c("a", "b", "c")
mc <- new("markovchain", states = statesNames,
  transitionMatrix = matrix(c(0.5, 0.3, 0.2,
                              0.2, 0.6, 0.2,
                              0.1, 0.1, 0.8), byrow = TRUE, nrow = 3,
                            dimnames = list(statesNames, statesNames)))
S <- sensitivity(mc, "a")
S
```

```
# Check against a finite-difference recomputation of the stationary
# distribution: increase p("a"->"c") and decrease p("a"->"b") by eps.
eps <- 1e-6
P2 <- mc@transitionMatrix
P2["a", "c"] <- P2["a", "c"] + eps
P2["a", "b"] <- P2["a", "b"] - eps
mc2 <- new("markovchain", states = statesNames, transitionMatrix = P2)
(steadyStates(mc2) - steadyStates(mc)) / eps   # finite difference
S["c", ] - S["b", ]                            # closed-form prediction
```

`show,hommc-method` *Function to display the details of hommc object*

Description

This is a convenience function to display the slots of hommc object in proper format

Usage

```
## S4 method for signature 'hommc'
show(object)
```

Arguments

`object`	An object of class hommc

`slem` *Second largest eigenvalue modulus (SLEM) of a Markov chain*

Description

Computes the second largest eigenvalue modulus (SLEM) of a finite, irreducible discrete-time Markov chain.

Usage

```
slem(object)

## S4 method for signature 'markovchain'
slem(object)
```

Arguments

`object`	A `markovchain` object representing a finite, irreducible discrete-time Markov chain.

Details

For a row-stochastic transition matrix P, let $1 = \lambda_1, \lambda_2, \ldots, \lambda_n$ be its eigenvalues. By the Perron-Frobenius theorem an irreducible chain has $\lambda_1 = 1$ with algebraic multiplicity one, and $|\lambda_k| \leq 1$ for every k. The SLEM is

$$\mathrm{SLEM} = \max_{k>1} |\lambda_k|.$$

Only irreducibility is required, not aperiodicity. If the chain is periodic, at least one non-trivial eigenvalue also has modulus one (e.g. $\lambda = -1$ for a 2-cycle), so `slem()` correctly returns `1` rather than rejecting the chain: a periodic chain genuinely does not contract towards its stationary distribution, which `SLEM = 1` reflects.

Repeated or complex non-trivial eigenvalues are handled through their modulus `Mod()`, so complex-conjugate pairs contribute the same value and ties do not need to be broken.

The implementation calls `eigen()` with `only.values = TRUE`, so it never computes eigenvectors. Its time complexity is $O(n^3)$ and its memory use is $O(n^2)$ for a dense n-state transition matrix. It supports both row- and column-stochastic storage.

Value

A numeric scalar in $[0, 1]$ containing the SLEM. For the trivial one-state chain, `0` is returned.

References

Levin, D. A. and Peres, Y. (2017). *Markov Chains and Mixing Times*, 2nd edition. American Mathematical Society.

See Also

`spectralGap`, `impliedTimescales`, `is.irreducible`, `period`

Examples

```
statesNames <- c("a", "b")
mc <- new("markovchain",
  states = statesNames,
  transitionMatrix = matrix(c(0.7, 0.3, 0.1, 0.9),
    byrow = TRUE, nrow = 2,
    dimnames = list(statesNames, statesNames)))
slem(mc)
```

spectralGap *Spectral gap of a Markov chain*

Description

Computes the spectral gap of a finite, irreducible discrete-time Markov chain, a lightweight diagnostic of its convergence and mixing behaviour.

Usage

```
spectralGap(object)

## S4 method for signature 'markovchain'
spectralGap(object)
```

Arguments

object	A markovchain object representing a finite, irreducible discrete-time Markov chain.

Details

The spectral gap is defined from the second largest eigenvalue modulus (SLEM, see slem) as

$$\text{gap} = 1 - \text{SLEM}.$$

As with slem, only irreducibility is required. A periodic chain has SLEM = 1 and therefore spectral gap 0: this is the mathematically correct value, not an error condition, since a periodic chain never contracts towards its stationary distribution. A larger spectral gap indicates faster convergence to stationarity; see impliedTimescales for the timescale associated with each non-trivial eigenvalue individually, of which the SLEM gives the slowest (dominant) one.

Value

A numeric scalar in $[0, 1]$ containing the spectral gap. For the trivial one-state chain, 1 is returned.

References

Levin, D. A. and Peres, Y. (2017). *Markov Chains and Mixing Times*, 2nd edition. American Mathematical Society.

See Also

slem, impliedTimescales, is.irreducible

Examples

```
statesNames <- c("a", "b")
mc <- new("markovchain",
  states = statesNames,
  transitionMatrix = matrix(c(0.7, 0.3, 0.1, 0.9),
    byrow = TRUE, nrow = 2,
    dimnames = list(statesNames, statesNames)))
spectralGap(mc)
```

---

states *Defined states of a transition matrix*

Description

This method returns the states of a transition matrix.

Usage

```
states(object)

## S4 method for signature 'markovchain'
states(object)
```

Arguments

object	A discrete markovchain object

Value

The character vector corresponding to states slot.

Author(s)

Giorgio Spedicato

References

A First Course in Probability (8th Edition), Sheldon Ross, Prentice Hall 2010

See Also

markovchain

Examples

```
statesNames <- c("a", "b", "c")
markovB <- new("markovchain", states = statesNames, transitionMatrix =
                matrix(c(0.2, 0.5, 0.3, 0, 1, 0, 0.1, 0.8, 0.1), nrow = 3,
                byrow = TRUE, dimnames=list(statesNames,statesNames)),
                name = "A markovchain Object"
)
states(markovB)
names(markovB)
```

---

steadyStates *Stationary states of a* markovchain *object*

Description

This method returns the stationary vector in matricial form of a markovchain object.

Usage

```
steadyStates(object)
```

Arguments

object	A discrete markovchain object

Value

A matrix corresponding to the stationary states

Note

The steady states are identified starting from which eigenvectors correspond to identity eigenvalues and then normalizing them to sum up to unity. When negative values are found in the matrix, the eigenvalues extraction is performed on the recurrent classes submatrix.

Author(s)

Giorgio Spedicato

References

A First Course in Probability (8th Edition), Sheldon Ross, Prentice Hall 2010

See Also

markovchain

Examples

```
statesNames <- c("a", "b", "c")
markovB <- new("markovchain", states = statesNames, transitionMatrix =
                matrix(c(0.2, 0.5, 0.3, 0, 1, 0, 0.1, 0.8, 0.1), nrow = 3,
                byrow = TRUE, dimnames=list(statesNames,statesNames)),
               name = "A markovchain Object"
)
steadyStates(markovB)
```

`subchain` *Restrict a Markov chain to a subset of states*

Description

Restricts a `markovchain` object to a chosen subset of its states, either as a raw (generally non-stochastic) principal submatrix, or as a properly renormalized Markov chain describing behaviour conditional on staying inside the subset.

Usage

```
subchain(object, states, method = c("submatrix", "renormalize"))

## S4 method for signature 'markovchain'
subchain(object, states, method = c("submatrix", "renormalize"))
```

Arguments

`object`	A `markovchain` object.
`states`	A character vector of at least one state name from `states(object)`, with no duplicates: the subset to restrict to.
`method`	Either `"submatrix"` or `"renormalize"` (can be abbreviated). See Details: the two options answer genuinely different questions and are not interchangeable.

Details

The two methods are deliberately named after two different, standard constructions, so that the choice – and its consequences – is explicit rather than implied:

`"submatrix"` Simply the entries of P with both indices restricted to `states`, with no adjustment. Its rows generally sum to *less than* one, because probability mass that originally went to states outside the subset is dropped, not redistributed. This is the "Q" block used, e.g., when building the fundamental matrix of an absorbing chain (see `fundamentalMatrix`): a useful building block for other computations, but not itself a transition matrix of any Markov chain, which is why it is returned as a plain matrix.

`"renormalize"` Each retained row is divided by its own sum, so the result is row-stochastic and can be wrapped in a `markovchain` object. This is the chain of successive positions of `object`, *conditioned on the event that it never leaves* `states` (sometimes called the chain "watched on" `states`, or its taboo probabilities; see Norris (1997), Section 3.3). It requires every state in `states` to have strictly positive probability of transitioning within the subset (otherwise that conditioning event has probability zero from that state, and the row cannot be renormalized); an error is raised naming any state that fails this, rather than silently producing a row of `NaN`.

Neither method requires `object` to be irreducible: restricting to a subset of states is meaningful for any chain, and is often used precisely to study one communicating class in isolation.

The implementation performs no eigendecomposition; it is $O(k^2)$ time and memory for a subset of size k, after an $O(n^2)$ extraction from the full n-state matrix.

Value

If `method = "submatrix"`: a plain numeric matrix (*not* a `markovchain` object, since its rows generally do not sum to one), the principal submatrix of the transition matrix restricted to `states`, always returned in row-stochastic orientation regardless of `object`'s own storage convention.

If `method = "renormalize"`: a new `markovchain` object on exactly the states in `states`, row-stochastic, describing the chain conditional on never leaving that subset.

References

Norris, J. R. (1998). *Markov Chains*. Cambridge University Press.

See Also

`lazyChain`, `fundamentalMatrix`, `canonicForm`

Examples

```
statesNames <- c("a", "b", "c")
mc <- new("markovchain", states = statesNames,
  transitionMatrix = matrix(c(0.5, 0.3, 0.2,
                              0.2, 0.6, 0.2,
                              0.1, 0.1, 0.8), byrow = TRUE, nrow = 3,
                            dimnames = list(statesNames, statesNames)))

# Raw submatrix: rows no longer sum to 1, mass has "leaked" to "c".
subchain(mc, c("a", "b"), method = "submatrix")
rowSums(subchain(mc, c("a", "b"), method = "submatrix"))

# Renormalized: a genuine markovchain, conditional on staying in {a, b}.
watched <- subchain(mc, c("a", "b"), method = "renormalize")
watched
rowSums(watched@transitionMatrix)
```

`tauchen` *Discretize an AR(1) process into a Markov chain (Tauchen's method)*

Description

Approximates the stationary first-order autoregressive process

$$y_t = (1-\rho)\alpha + \rho y_{t-1} + \varepsilon_t, \qquad \varepsilon_t \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma^2)$$

by a finite-state Markov chain on an evenly spaced grid, following Tauchen (1986).

Usage

```
tauchen(alpha, sigma, rho, size, k = 3)
```

Arguments

`alpha`	A single finite number: the unconditional mean of the process.
`sigma`	A single finite positive number: the standard deviation of the innovation ε_t.
`rho`	A single number in $(-1, 1)$: the autocorrelation (persistence) of the process.
`size`	A single integer of at least 2: the number of grid points (states) of the discretized chain.
`k`	A single positive number, the half-width of the grid in units of the process's unconditional standard deviation $\sigma_y = \sigma/\sqrt{1-\rho^2}$. The default, 3, follows Tauchen (1986)'s own recommendation and covers the great majority of the stationary distribution's mass for typical `rho`.

Details

The grid is $n =$ `size` evenly spaced points $y_1 < \cdots < y_n$ spanning $[\alpha - k\sigma_y,\ \alpha + k\sigma_y]$, with half-spacing $w = (y_n - y_1)/(2(n-1))$. Writing Φ for the standard normal CDF, the transition probabilities from grid point y_i are

$$P_{i1} = \Phi\left(\frac{y_1 - (1-\rho)\alpha - \rho y_i + w}{\sigma}\right),$$

$$P_{in} = 1 - \Phi\left(\frac{y_n - (1-\rho)\alpha - \rho y_i - w}{\sigma}\right),$$

$$P_{ij} = \Phi\left(\frac{y_j - (1-\rho)\alpha - \rho y_i + w}{\sigma}\right) - \Phi\left(\frac{y_j - (1-\rho)\alpha - \rho y_i - w}{\sigma}\right), \quad 1 < j < n,$$

i.e. the probability that y_t (a normal draw centred at the AR(1) conditional mean) lands in the half-open bin around y_j, with the two end bins extended to $\pm\infty$ so that rows sum to exactly 1.

Tauchen's method is simple and fast ($O(n^2)$ normal CDF evaluations) but the grid width is fixed by `k` regardless of `size`: for a coarse grid (small `size`) it under-resolves the bulk of the distribution, and for `rho` close to ± 1 the true unconditional variance is large and sensitive to `k`. See `rouwenhorst` for an alternative that tends to match the persistence of near-unit-root processes more accurately and needs no arbitrary grid-width parameter.

Value

A named list with two elements:

`chain` The discretized `markovchain` object, with state names equal to the grid values of y formatted to 4 significant digits.

`states` The numeric grid of y-values themselves, in the same order as `chain`'s states. Returning the actual levels alongside the chain, rather than only generic state labels `"1"`, `"2"`, ..., is deliberate: the whole point of discretizing an AR(1) process is usually to do further numeric work with the levels (e.g. plugging them into a pricing formula), and re-deriving the grid from `alpha`, `sigma`, `rho` and `k` a second time by hand is both extra work and a place for an off-by-one or rounding mismatch to creep in.

References

Tauchen, G. (1986). Finite state markov-chain approximations to univariate and vector autoregressions. *Economics Letters*, 20(2), 177-181.

See Also

`rouwenhorst`

Examples

```
out <- tauchen(alpha = 0, sigma = 1, rho = 0.9, size = 5)
out$states
out$chain
# The chain's own stationary variance should be close to the AR(1)'s
# theoretical unconditional variance sigma^2 / (1 - rho^2).
pi <- as.numeric(steadyStates(out$chain))
sum(pi * (out$states - sum(pi * out$states))^2)
1 / (1 - 0.9^2)
```

`timeCorrelations` *Time correlations and time relaxations of observed sequences*

Description

`timeCorrelations` computes the time autocorrelation of an observed sequence of states, or the time cross-correlation of two sequences, at stationarity. `timeRelaxations` computes how the expected value of the observable defined by a sequence evolves from a given initial distribution. They correspond to `time_correlations()` and `time_relaxations()` of PyDTMC.

Usage

```
timeCorrelations(object, sequence1, sequence2 = NULL, timePoints = 1)

## S4 method for signature 'markovchain'
timeCorrelations(object, sequence1, sequence2 = NULL, timePoints = 1)

timeRelaxations(object, sequence, initial = NULL, timePoints = 1)

## S4 method for signature 'markovchain'
timeRelaxations(object, sequence, initial = NULL, timePoints = 1)
```

Arguments

`object`	A `markovchain` object.
`sequence1`, `sequence`	A sequence of states of the chain (character vector or factor).
`sequence2`	An optional second sequence of states. If `NULL` (the default), `sequence1` is used, which gives the autocorrelation.
`timePoints`	A vector of non-negative whole numbers, the lags at which the quantities are computed.
`initial`	The initial distribution: `NULL` (uniform, the default), a single state, or a numeric probability vector, as in `redistribute`.

Details

A sequence defines an observable f on the states: f_j is the number of times state j occurs in it. With f from `sequence1`, g from `sequence2`, transition matrix P and stationary distribution π,

$$\text{timeCorrelations}(t) = \sum_i \pi_i f_i (P^t g)_i = E_\pi[f(X_0) g(X_t)],$$

and, with initial distribution μ,

$$\text{timeRelaxations}(t) = \mu P^t f = E_\mu[f(X_t)].$$

For an ergodic chain, both converge as t grows, to $E_\pi[f]E_\pi[g]$ and to $E_\pi[f]$ respectively, at a speed governed by the second largest eigenvalue modulus (`slem`).

The powers of P are applied by repeated multiplication, and by repeated squaring for long lags, never through an eigendecomposition. PyDTMC 9.0.0 switches to an eigendecomposition as soon as a lag exceeds the number of states; since its left and right eigenvectors are not biorthonormal when P has complex eigenvalues, it then returns wrong values at every lag for such chains (the tests of this function include one, whose correct values were checked with `numpy`).

`timeCorrelations` needs a unique stationary distribution, i.e. exactly one recurrent class, and stops otherwise (PyDTMC returns `None`). `timeRelaxations` is defined for every chain; unlike PyDTMC, it does not require a unique stationary distribution.

Value

A numeric vector with one value per element of `timePoints`, named after them.

References

Noe, F., Doose, S., Daidone, I., Loellmann, M., Sauer, M., Chodera, J. D. and Smith, J. C. (2011). Dynamical fingerprints for probing individual relaxation processes in biomolecular dynamics with simulations and kinetic experiments. *Proceedings of the National Academy of Sciences*, 108(12), 4822-4827.

See Also

redistribute, slem, relaxationTime

Examples

```
statesNames <- c("a", "b", "c")
mc <- new("markovchain", states = statesNames,
  transitionMatrix = matrix(c(0.5, 0.5, 0, 0.2, 0.3, 0.5, 0.1, 0.1, 0.8),
    byrow = TRUE, nrow = 3, dimnames = list(statesNames, statesNames)))
x <- c("a", "b", "c", "c", "c", "a")
timeCorrelations(mc, x, timePoints = 0:5)
timeRelaxations(mc, x, initial = "a", timePoints = c(0, 1, 10, 100))
```

---

tm_abs *Single Year Corporate Credit Rating Transitions*

Description

Matrix of Standard and Poor's Global Corporate Rating Transition Frequencies 2000 (NR Removed)

Usage

```
data(tm_abs)
```

Format

The format is: num [1:8, 1:8] 17 2 0 0 0 0 0 0 1 455 ... - attr(*, "dimnames")=List of 2 ..$: chr [1:8] "AAA" "AA" "A" "BBB" ... ..$: chr [1:8] "AAA" "AA" "A" "BBB" ...

References

European Securities and Markets Authority, 2016 https://cerep.esma.europa.eu/cerep-web/statistics/transitionMatrice.xhtml

Examples

```
data(tm_abs)
```

`toBoundedChain` *Apply a boundary condition to a Markov chain's first and last state*

Description

Replaces the transition rows of a `markovchain` object's first and last state (in `states(object)` order) with an absorbing, reflecting, or semi-reflecting rule, leaving every other row unchanged.

Usage

```
toBoundedChain(object, boundaryCondition)

## S4 method for signature 'markovchain'
toBoundedChain(object, boundaryCondition)
```

Arguments

`object`	A `markovchain` object with at least 2 states.
`boundaryCondition`	Either: • the string `"absorbing"`: the first and last state each become absorbing ($P_{11} = 1$, $P_{nn} = 1$); • the string `"reflecting"`: the first state moves to the second with certainty and the last state moves to the second-to-last with certainty ($P_{12} = 1$, $P_{n,n-1} = 1$); • a single number $\beta \in [0, 1]$, the *semi-reflecting* case: the first state stays with probability $1 - \beta$ and moves to the second state with probability β ($P_{11} = 1 - \beta$, $P_{12} = \beta$), and symmetrically the last state stays with probability $1 - \beta$ and moves to the second-to-last with probability β. $\beta = 0$ is the absorbing case and $\beta = 1$ is the reflecting case.

Details

This function assumes – as is standard for a boundary condition – that `states(object)` is meaningfully ordered along a line, first state to last state, as it would be e.g. for `birthDeath` or any other chain built to represent a bounded random walk. It does not check this (there is no general way to check it from the transition matrix alone) and applies the same first/last-row replacement regardless of `object`'s actual structure; only the two boundary rows are ever touched, so applying it to a chain whose states are not linearly ordered simply reinterprets whichever states happen to be listed first and last.

Unlike `gamblersRuin`, which is absorbing at both ends by construction and cannot be un-done, `toBoundedChain()` can be applied to any existing chain and with any of the three conditions, including reflecting or semi-reflecting ones that `gamblersRuin()` does not offer directly.

The implementation touches only 2 of the n rows and is $O(n)$ time and memory beyond copying the transition matrix.

Value

A new markovchain object, row-stochastic, on the same states as object, identical to object except in its first and last transition rows.

See Also

birthDeath, gamblersRuin

Examples

```
bd <- birthDeath(p = c(0.3, 0.4, 0.5), q = c(0.2, 0.3, 0.1))

absorbed <- toBoundedChain(bd, "absorbing")
absorbed@transitionMatrix[1, ]
absorbed@transitionMatrix[4, ]

reflected <- toBoundedChain(bd, "reflecting")
reflected@transitionMatrix[1, ]

semiReflected <- toBoundedChain(bd, 0.25)
semiReflected@transitionMatrix[1, ]
```

---

toDictionary *Represent a Markov chain as a plain R list*

Description

Converts a markovchain object to a plain, self-describing R list: the same information toFile writes to disk, kept in memory. fromDictionary reverses the conversion.

Usage

```
toDictionary(object)

## S4 method for signature 'markovchain'
toDictionary(object)

fromDictionary(d)
```

Arguments

object	A markovchain object.
d	A list as returned by toDictionary. d$transitionMatrix may also be a plain n x n matrix (with or without dimnames matching d$states) for convenience when building a dictionary by hand rather than from an existing markovchain object. When it is a plain matrix, d$byrow is honored exactly as new("markovchain", ...) honors its own byrow argument: the matrix is stored as given, with d$byrow

only documenting whether it is row- or column-stochastic (so a column-stochastic matrix round-trips by setting `d$byrow = FALSE`, with no transposition performed here). When `d$transitionMatrix` is a nested list (as `toDictionary` produces), the nesting itself is always keyed `[[from]][[to]]` – i.e. row-stochastic – so it is always reconstructed with `byrow = TRUE`, regardless of `d$byrow`.

Details

Unlike PyDTMC's own `to_dictionary()`/`from_dictionary()`, which represent a chain as a flat mapping from every `(from_state, to_state)` pair to its probability – n^2 entries with no state grouping – this nests the representation by source state, which is both more compact to read and directly round-trips through R's own list-of-lists idiom without any special tuple-key handling.

Value

A named list with four elements:

`name` The chain's `name`, as a single string (possibly empty).

`states` A character vector of state names, in order.

`byrow` Always `TRUE`: the list always stores the chain row-stochastically, regardless of `object`'s own storage convention, so that the representation is unambiguous without also having to interpret this flag.

`transitionMatrix` A named list of named lists: `transitionMatrix[[i]][[j]]` is the probability of moving from state `i` to state `j`. This is deliberately not a plain matrix, so that the structure serializes to JSON or YAML (via `toFile`) as a self-describing object keyed by state name, rather than a bare array whose meaning depends on remembering a row/column order.

`fromDictionary` returns a `markovchain` object.

See Also

`toFile`, `fromFile`

Examples

```
statesNames <- c("a", "b")
mc <- new("markovchain", states = statesNames,
          transitionMatrix = matrix(c(0.7, 0.3, 0.4, 0.6), byrow = TRUE,
                                    nrow = 2, dimnames = list(statesNames, statesNames)))
d <- toDictionary(mc)
d$transitionMatrix$a$b # 0.3: probability of moving from "a" to "b"

identical(fromDictionary(d)@transitionMatrix, mc@transitionMatrix)
```

toDot *Export the transition graph as Graphviz DOT or Mermaid text*

Description

toDot writes the transition graph of a Markov chain in the DOT language of Graphviz; toMermaid writes it as a Mermaid flowchart. The states are the nodes and every transition with positive probability is an edge labelled with its probability.

Usage

```
toDot(object, file = NULL, digits = 3, minProbability = 0, direction = "LR")

## S4 method for signature 'markovchain'
toDot(object, file = NULL, digits = 3, minProbability = 0, direction = "LR")

toMermaid(
  object,
  file = NULL,
  digits = 3,
  minProbability = 0,
  direction = "LR"
)

## S4 method for signature 'markovchain'
toMermaid(
  object,
  file = NULL,
  digits = 3,
  minProbability = 0,
  direction = "LR"
)
```

Arguments

object	A markovchain object.
file	An optional file path. If given, the text is also written there (UTF-8) and returned invisibly.
digits	The number of significant digits of the probabilities shown on the edges.
minProbability	Transitions with probability not above this value are left out; the default, 0, keeps every possible transition.
direction	The layout direction: "LR" (left to right, the default), "TB" (top to bottom), "RL" or "BT".

Details

The edges follow the outgoing distribution of each state whatever the storage of the transition matrix (`byrow`), so a chain and its column-stochastic copy give the same text. State names are quoted and escaped, so they may contain spaces, quotes or other punctuation. In the Mermaid output the nodes get the identifiers `s1`, `s2`, ..., with the state names as labels, because Mermaid identifiers cannot contain arbitrary characters.

The DOT text can be rendered with Graphviz (for instance `dot -Tpng chain.dot -o chain.png`) or with `DiagrammeR::grViz()`; the Mermaid text can be pasted into any Markdown renderer that supports Mermaid diagrams, or rendered with `DiagrammeR::mermaid()`.

Value

The text, as a single character string (invisibly when `file` is given).

See Also

`toFile`, `plot,markovchain,missing-method`

Examples

```
statesNames <- c("a", "b", "c")
mc <- new("markovchain", states = statesNames, name = "Example",
  transitionMatrix = matrix(c(0.5, 0.5, 0, 0.2, 0.3, 0.5, 0, 0, 1),
    byrow = TRUE, nrow = 3, dimnames = list(statesNames, statesNames)))
cat(toDot(mc), "\n")
cat(toMermaid(mc, direction = "TB"), "\n")
```

---

toFile *Write or read a Markov chain to or from a file*

Description

Writes a `markovchain` object to a JSON, YAML, CSV or XML file, or reads one back, using the same representation as `toDictionary`.

Usage

```
toFile(object, file, format = NULL)

## S4 method for signature 'markovchain'
toFile(object, file, format = NULL)

fromFile(file, format = NULL)
```

Arguments

object	A markovchain object (for toFile).
file	A single file path to write to or read from. If format is not supplied, it is inferred from the file extension (.json, .yaml/.yml, .csv or .xml).
format	One of "json", "yaml", "csv" or "xml". The default, NULL, infers the format from file's extension.

Details

The JSON and YAML formats store exactly what toDictionary returns (name, state names, and the transition probabilities nested by source state), and round-trip a chain exactly, including full numeric precision (toFile writes both JSON and YAML with 17 significant digits, enough to recover every double exactly).

The CSV format only stores the transition matrix itself, as a table of probabilities with the state names as both the header row and the first column – there is no natural place in a CSV file for the chain's name, so it is not preserved by toFile(..., format = "csv") and fromFile always returns an unnamed chain for a .csv file. This is the same limitation PyDTMC's own CSV format has.

The XML format is the one of PyDTMC, so files can be exchanged with it in both directions: a root element MarkovChain with one Item element per transition, whose attributes are state_from, state_to and probability. All n^2 transitions are written, zeros included, and probabilities use 17 significant digits, so the round trip is exact. The name of the chain is stored as an attribute of the root element, which PyDTMC ignores when reading. When reading, the states are taken in the order in which their self transitions (state_from equal to state_to) appear, as PyDTMC does, and the name is restored if the attribute is present. Writing XML uses only base R; reading it requires the **xml2** package.

Writing JSON requires the **jsonlite** package, and writing YAML requires the **yaml** package; both are only in Suggests, and an informative error is raised if the relevant package is not installed. Reading has the same requirements for the format being read. CSV uses only base R and has no extra package dependency.

Value

toFile returns file, invisibly. fromFile returns a markovchain object.

See Also

toDictionary, fromDictionary

Examples

```
## Not run:
statesNames <- c("a", "b")
mc <- new("markovchain", states = statesNames,
          transitionMatrix = matrix(c(0.7, 0.3, 0.4, 0.6), byrow = TRUE,
                                    nrow = 2, dimnames = list(statesNames, statesNames)))
toFile(mc, "chain.json")
identical(fromFile("chain.json")@transitionMatrix, mc@transitionMatrix)
```

```
## End(Not run)
```

`toNthOrder` *Return the n-step transition chain*

Description

Returns the `markovchain` object whose transition matrix is $P^{\texttt{order}}$: from any state, its one-step transition probabilities are the original chain's `order`-step transition probabilities.

Usage

```
toNthOrder(object, order)
```

Arguments

`object`	A `markovchain` object.
`order`	A single integer of at least 2.

Details

This is a thin, discoverability-only wrapper around `object ^ order` (see `^,markovchain,numeric-method`), provided under this name because **PyDTMC**'s equivalent method is called `to_nth_order()`. It exists so that the operation is easy to find by that name; it introduces no new computation; the underlying `^` method is already $O(n^3 \log(\texttt{order}))$ via repeated squaring (`expm::%^%`), not a naive `order`-fold product, so there is nothing to improve on algorithmically here.

Value

A new `markovchain` object on the same states as `object`, with transition matrix $P^{\texttt{order}}$.

See Also

`toBoundedChain`, `lazyChain`

Examples

```
mc <- new("markovchain", states = c("a", "b"),
          transitionMatrix = matrix(c(0.9, 0.1, 0.3, 0.7), byrow = TRUE, nrow = 2))
identical(unclass(toNthOrder(mc, 5)@transitionMatrix), unclass((mc ^ 5)@transitionMatrix))
```

`topologicalEntropy` *Topological entropy of a Markov chain*

Description

Computes the topological entropy of the graph of a discrete-time Markov chain: the exponential growth rate of the number of distinct admissible paths, ignoring their probabilities.

Usage

```
topologicalEntropy(object, base = 2)

## S4 method for signature 'markovchain'
topologicalEntropy(object, base = 2)
```

Arguments

`object`	A `markovchain` object.
`base`	A finite numeric scalar strictly greater than one. The default, `2`, returns bits, consistently with `entropyRate`.

Details

If A is the 0/1 adjacency matrix with $A_{ij} = 1$ exactly when $p_{ij} > 0$, the topological entropy is

$$h_{top} = \log_b \rho(A),$$

where $\rho(A)$ is the spectral radius (Perron root) of A.

Only the pattern of positive entries matters: the value depends on which transitions are possible, not on how likely they are. It is the upper bound of the entropy rate over all the Markov chains sharing that graph (the variational principle, see Parry, 1964), so `entropyRate(object) <= topologicalEntropy(object)` for an irreducible chain. The bound is attained by the maximal-entropy (Parry) chain on the same graph, and also, for instance, by a chain whose every row is uniform over a common number of successors. A chain that is a single cycle (deterministic dynamics) has `topologicalEntropy = 0`.

No irreducibility is needed: for a reducible chain the result is the largest value over its communicating classes. A probability that is positive but numerically tiny counts as a transition, exactly as in `is.irreducible`.

The cost is one eigenvalue computation, $O(n^3)$ time and $O(n^2)$ memory for a dense chain. It mirrors PyDTMC's `topological_entropy`, which uses the natural logarithm.

Value

A non-negative numeric scalar in units determined by `base`.

References

Parry, W. (1964). Intrinsic Markov chains. *Transactions of the American Mathematical Society*, 112, 55-66.

Cover, T. M. and Thomas, J. A. (2006). *Elements of Information Theory*, 2nd edition. Wiley.

See Also

entropyRate, normalizedEntropyRate

Examples

```
statesNames <- c("a", "b")
mc <- new("markovchain",
  states = statesNames,
  transitionMatrix = matrix(c(0.7, 0.3, 0.1, 0.9),
    byrow = TRUE, nrow = 2,
    dimnames = list(statesNames, statesNames)))
topologicalEntropy(mc)
```

---

transition2Generator *Return the generator matrix for a corresponding transition matrix*

Description

Calculate the generator matrix for a corresponding transition matrix

Usage

```
transition2Generator(P, t = 1, method = "logarithm")
```

Arguments

P	transition matrix between time 0 and t
t	time of observation
method	"logarithm" returns the Matrix logarithm of the transition matrix

Value

A matrix that represent the generator of P

See Also

rctmc

Examples

```
mymatr <- matrix(c(.4, .6, .1, .9), nrow = 2, byrow = TRUE)
Q <- transition2Generator(P = mymatr)
expm::expm(Q)
```

`transitionProbability` *Function to get the transition probabilities from initial to subsequent states.*

Description

This is a convenience function to get transition probabilities.

Usage

```
transitionProbability(object, t0, t1)

## S4 method for signature 'markovchain'
transitionProbability(object, t0, t1)
```

Arguments

`object`	A `markovchain` object.
`t0`	Initial state.
`t1`	Subsequent state.

Value

Numeric Vector

Author(s)

Giorgio Spedicato

References

A First Course in Probability (8th Edition), Sheldon Ross, Prentice Hall 2010

See Also

`markovchain`

Examples

```
statesNames <- c("a", "b", "c")
markovB <- new("markovchain", states = statesNames, transitionMatrix =
                matrix(c(0.2, 0.5, 0.3, 0, 1, 0, 0.1, 0.8, 0.1), nrow = 3,
                byrow = TRUE, dimnames=list(statesNames,statesNames)),
               name = "A markovchain Object"
)
transitionProbability(markovB,"b", "c")
```

---

urnModel *Build an Ehrenfest urn model Markov chain*

Description

Constructs the Ehrenfest diffusion model: `balls` balls are split between two urns, A and B; at each step, one of the `balls` balls is chosen uniformly at random and moved to the other urn. The chain tracks the number of balls in urn A.

Usage

```
urnModel(balls, states = NULL)
```

Arguments

`balls`	A single positive integer, the total number of balls. The chain has `balls` + 1 states, $0, 1, \ldots,$ `balls` (the possible counts of balls in urn A).
`states`	An optional character vector of `balls` + 1 state names, in increasing order of ball count. Defaults to `as.character(0:balls)`.

Details

The Ehrenfest model is the classical example of a chain whose equilibrium behaviour matches thermodynamic intuition despite every individual transition being fully reversible: its stationary distribution is $\mathrm{Binomial}(\texttt{balls}, 1/2)$ (each ball is, at equilibrium, independently in urn A or B with probability $1/2$), sharply concentrated around $\texttt{balls}/2$ for large `balls` even though the chain only ever moves one ball at a time and is reflecting, not absorbing, at the boundaries. It is irreducible and reversible for every `balls`, but periodic with period 2 (the parity of the ball count in urn A alternates every step): pass the result through `lazyChain` first if an aperiodic chain is needed, e.g. for `mixingTime`.

Value

A new, row-stochastic `markovchain` object with `balls` + 1 states. From state i ($0 < i <$ `balls`),

$$P_{i,i-1} = i/\texttt{balls}, \qquad P_{i,i+1} = 1 - i/\texttt{balls},$$

the probability that the ball moved was one of the i currently in urn A (decreasing A's count) versus one of the $\texttt{balls} - i$ currently in urn B (increasing it). States 0 and `balls` (all balls in one urn) are *reflecting*: the next ball moved must come from the only non-empty urn, so $P_{0,1} = P_{\texttt{balls},\texttt{balls}-1} = 1$ exactly.

References

Ehrenfest, P. and Ehrenfest, T. (1907). Uber zwei bekannte Einwande gegen das Boltzmannsche H-Theorem. *Physikalische Zeitschrift*, 8, 311-314.

See Also

birthDeath, lazyChain

Examples

```
ehrenfest <- urnModel(balls = 4)
ehrenfest
steadyStates(ehrenfest) # approximately Binomial(4, 0.5): 1/16 6/16 ...
dbinom(0:4, 4, 0.5)
```

---

verifyMarkovProperty *Test the first-order Markov property of an empirical sequence*

Description

Tests the null hypothesis that the conditional distribution of the next state depends only on the current state, against a second-order alternative.

Tests whether the sequence is compatible with a first-order Markov chain against a second-order alternative, by testing independence of past and future states conditional on the present state. Degrees of freedom are summed, present-state by present-state, over only the past and future states actually observed with that present state, mirroring the other functions documented on this page. Factors and numeric sequences are compared as character strings.

Tests whether transition probabilities are constant across consecutive blocks. Structural zeros can be supplied explicitly through a logical transition matrix.

Usage

```
verifyMarkovProperty(
  sequence,
  method = c("G", "Pearson", "simulation"),
  B = 9999,
  seed = NULL,
  verbose = TRUE
)

assessOrder(sequence, verbose = TRUE)

verifyEmpiricalToTheoretical(
  data,
  object,
  method = c("G", "Pearson", "simulation"),
```

```
  B = 9999,
  seed = NULL,
  verbose = TRUE
)

verifyHomogeneity(
  inputList,
  method = c("G", "Pearson", "simulation"),
  B = 9999,
  seed = NULL,
  verbose = TRUE
)

assessStationarity(sequence, nblocks, structural.zeros = NULL, verbose = TRUE)
```

Arguments

sequence	An empirical sequence.
method	Test statistic: ‘"G"‘, ‘"Pearson"‘, or ‘"simulation"‘.
B	Number of Monte Carlo replicates for simulation.
seed	Optional random seed.
verbose	Should test results be printed?
data	An empirical sequence or a matrix of transition counts.
object	A ‘markovchain‘ object specifying theoretical probabilities.
inputList	A list whose elements are empirical sequences or matrices.
nblocks	Number of blocks, at least two.
structural.zeros	Optional logical matrix marking impossible transitions.

Value

An ‘htest‘ object with additional package-specific components.

An ‘htest‘ object.

An ‘htest‘ object with observed and expected counts.

An ‘htest‘ object with pooled and individual transition counts.

An ‘htest‘ object.

References

Anderson, T. W. and Goodman, L. A. (1957). Statistical inference about Markov chains. *The Annals of Mathematical Statistics*, 28(1), 89–110.

Kullback, S., Kupperman, M. and Ku, H. H. (1962). Tests for Contingency Tables and Markov Chains. *Technometrics*, 4(4), 573–608.

Anderson, T. W. and Goodman, L. A. (1957). Statistical inference about Markov chains. *The Annals of Mathematical Statistics*, 28(1), 89–110.

See Also

Other statisticalTests: `assessIndependence()`

`zeros` *Matrix to create zeros*

Description

Matrix to create zeros

Usage

```
zeros(n)
```

Arguments

`n`	size of the matrix

Value

a square matrix of zeros

Index

!=,markovchain,markovchain-method
(markovchain-class), 62
∗ **classes**
ctmc-class, 20
markovchain-class, 62
markovchainList-class, 69
∗ **datasets**
blanden, 14
craigsendi, 19
holson, 43
kullback, 60
preproglucacon, 91
rain, 94
sales, 103
tm_abs, 118
∗ **package**
markovchain-package, 4
∗ **statisticalTests**
assessIndependence, 9
verifyMarkovProperty, 130
*,markovchain,markovchain-method
(markovchain-class), 62
*,markovchain,matrix-method
(markovchain-class), 62
*,markovchain,numeric-method
(markovchain-class), 62
*,matrix,markovchain-method
(markovchain-class), 62
*,numeric,markovchain-method
(markovchain-class), 62
.markovchainFitRcpp (markovchainFit), 65
==,markovchain,markovchain-method
(markovchain-class), 62
[,markovchain,ANY,ANY,ANY-method
(markovchain-class), 62
[[,markovchainList-method
(markovchainList-class), 69
%^%, *125*
^,markovchain,numeric-method
(markovchain-class), 62

absorbingStates, *36*
absorbingStates (period), 85
absorbingStates,markovchain-method
(markovchain-class), 62
absorptionProbabilities, 5, *36*, *37*
absorptionProbabilities,markovchain-method
(markovchain-class), 62
aggregateStates, 6
aggregateStates,markovchain-method
(aggregateStates), 6
assessIndependence, 9
assessIndependence(), *132*
assessOrder, *10*, *105*
assessOrder (verifyMarkovProperty), 130
assessStationarity, *10*
assessStationarity
(verifyMarkovProperty), 130
autoLump, *7*, *8*, 10
autoLump,markovchain-method (autoLump),
10
autoplot.markovchain, 11, *98*

birthDeath, 12, *37*, *89*, *119*, *120*, *130*
blanden, 14

canonicForm, *114*
canonicForm (period), 85
canonicForm,markovchain-method
(markovchain-class), 62
closestReversible, *8*, 15
closestReversible,markovchain-method
(closestReversible), 15
coerce,data.frame,markovchain-method
(markovchain-class), 62
coerce,etm,markovchain-method
(markovchain-class), 62
coerce,markovchain,data.frame-method
(markovchain-class), 62

coerce,markovchain,igraph-method
(markovchain-class), 62
coerce,markovchain,matrix-method
(markovchain-class), 62
coerce,markovchain,sparseMatrix-method
(markovchain-class), 62
coerce,Matrix,markovchain-method
(markovchain-class), 62
coerce,matrix,markovchain-method
(markovchain-class), 62
coerce,msm,markovchain-method
(markovchain-class), 62
coerce,msm.est,markovchain-method
(markovchain-class), 62
coerce,sparseMatrix,markovchain-method
(markovchain-class), 62
coerce,table,markovchain-method
(markovchain-class), 62
committorAB, 17
communicatingClasses (period), 85
communicatingClasses,markovchain-method
(markovchain-class), 62
conditionalDistribution, 18, *28*
conditionalDistribution,markovchain-method
(markovchain-class), 62
craigsendi, 19
createSequenceMatrix (markovchainFit),
65
ctmc-class, 20
ctmcFit, 21

dim,ctmc-method (ctmc-class), 20
dim,markovchain-method
(markovchain-class), 62
dim,markovchainList-method
(markovchainList-class), 69
dirichletChain, 22, *96*

entropyRate, 23, *84*, *126, 127*
entropyRate,markovchain-method
(entropyRate), 23
expectedRewards, 25
expectedRewardsBeforeHittingA, 26
ExpectedTime, 26

firstPassage, 27, *29*
firstPassageMultiple, 28
fitHigherOrder, 29, *32–34*, *38–41*, *105*
fitHighOrderMultivarMC, 31, *90*
fitMTD, *30*, 32, *39–41*, *105*
freq2Generator, 34
fromDictionary, *124*
fromDictionary (toDictionary), 120
fromFile, *121*
fromFile (toFile), 123
fundamentalMatrix, 35, *113, 114*

gamblersRuin, *13*, 36, *89*, *119, 120*
generatorToTransitionMatrix, *20*, 37, *97*

higherOrderLogLik, *30*, *33, 34*, 38, *41*, *105*
HigherOrderMarkovChain-class, 40
higherOrderPredict, *34*, 40, *40*
higherOrderSimulate
(higherOrderPredict), 40
hittingProbabilities, 42
hittingProbabilities,markovchain-method
(markovchain-class), 62
holson, 43
hommc (hommc-class), 44
hommc-class, 44

ictmc (ictmc-class), 45
ictmc-class, 45
identityChain, 46, *96*
impliedTimescales, 47, *61*, *79*, *99*, *109, 110*
impliedTimescales,markovchain-method
(impliedTimescales), 47
impreciseProbabilityatT, 48
inferHyperparam, 49, *93*
initialize,ctmc_method (ctmc-class), 20
initialize,markovchain-method
(markovchain-class), 62
is.accessible, 50
is.accessible,markovchain,character,character-method
(markovchain-class), 62
is.accessible,markovchain,missing,missing-method
(markovchain-class), 62
is.CTMCirreducible, 52
is.irreducible, *17*, *24*, *48*, 53, *55, 56*, *60*,
79, *107*, *109, 110*, *126*
is.irreducible,markovchain-method
(markovchain-class), 62
is.lumpable, *7, 8*, 54
is.lumpable,markovchain-method
(is.lumpable), 54
is.regular, 54

is.regular,markovchain-method
(markovchain-class), 62
is.reversible, *17*, 55
is.reversible,markovchain-method
(is.reversible), 55
is.stochasticallyMonotone, 57
is.stochasticallyMonotone,ANY-method
(is.stochasticallyMonotone), 57
is.stochasticallyMonotone,markovchain-method
(is.stochasticallyMonotone), 57
is.stochasticallyMonotone,matrix-method
(is.stochasticallyMonotone), 57
is.TimeReversible, 58

kemenyConstant, 59, *106, 107*
kemenyConstant,markovchain-method
(kemenyConstant), 59
kullback, 60

lazyChain, *8*, *46*, 60, *77, 78*, *114*, *125*, *129, 130*
lazyChain,markovchain-method
(lazyChain), 60
lump, *7, 8*, 62
lump,markovchain-method (lump), 62

markovchain, *19*, *70*, *86*, *111, 112*, *128*
markovchain (markovchain-package), 4
markovchain-class, 62
markovchain-package, 4
markovchainFit, *34*, *50*, 65, *65*, *72*, *91*, *101*
markovchainList-class, 69
markovchainListFit, *68*, 70
markovchainSequence, *65*, *68*, 71, *101*
meanAbsorptionTime, *36*, 73
meanAbsorptionTime,markovchain-method
(markovchain-class), 62
meanFirstPassageTime, *60*, 74
meanFirstPassageTime,markovchain,character-method
(markovchain-class), 62
meanFirstPassageTime,markovchain,missing-method
(markovchain-class), 62
meanNumVisits, 75
meanNumVisits,markovchain-method
(markovchain-class), 62
meanRecurrenceTime, 76
meanRecurrenceTime,markovchain-method
(markovchain-class), 62
mergeWith, 77
mergeWith,markovchain,markovchain-method
(mergeWith), 77
mixingTime, *56*, *61*, 78, *98, 99*, *129*
mixingTime,markovchain-method
(mixingTime), 78
multinomialConfidenceIntervals, 80

name, 81
name,markovchain-method (name), 81
name<-, 82
name<-,markovchain-method (name<-), 82
names,markovchain-method, 82
names<-,markovchain-method
(markovchain-class), 62
new, *120*
noofVisitsDist, 83
normalizedEntropyRate, 84, *127*
normalizedEntropyRate,markovchain-method
(normalizedEntropyRate), 84

ones, 85

period, *79*, 85, *109*
plot,ctmc,missing-method (ctmc-class),
20
plot,markovchain,missing-method
(markovchain-class), 62
populationGeneticsModel, 87
predict,markovchain-method
(markovchain-class), 62
predict,markovchainList-method
(markovchainList-class), 69
predictHommc, 89
predictiveDistribution, *50*, 90, *93*
preproglucacon, 91
print,markovchain-method
(markovchain-class), 62
print,markovchainList-method
(markovchainList-class), 69
priorDistribution, 92
probabilityatT, 93

rain, 94
randomMarkovChain, *22, 23*, *46*, 95
rctmc, *20*, *22*, *38*, 96, *127*
recurrentClasses (period), 85
recurrentClasses,markovchain-method
(markovchain-class), 62
recurrentStates (period), 85

recurrentStates,markovchain-method (markovchain-class), 62
redistribute, *11*, 97, *117, 118*
redistribute,markovchain-method (redistribute), 97
relaxationTime, 99, *118*
relaxationTime,markovchain-method (relaxationTime), 99
rmarkovchain, *41*, 100
rouwenhorst, 102, *115, 116*

sales, 103
selectOrder, 104
sensitivity, 106
sensitivity,markovchain-method (sensitivity), 106
seq2freqProb (fitHigherOrder), 29
seq2matHigh, *39*
seq2matHigh (fitHigherOrder), 29
set.seed, *33*
show,hommc-method, 108
show,markovchain-method (markovchain-class), 62
show,markovchainList-method (markovchainList-class), 69
slem, *47, 48*, *56*, *61*, *79*, *99*, 108, *110*, *117, 118*
slem,markovchain-method (slem), 108
sort,markovchain-method (markovchain-class), 62
spectralGap, *47, 48*, *56*, *79*, *99*, *109*, 110
spectralGap,markovchain-method (spectralGap), 110
states, 111
states,ctmc-method (ctmc-class), 20
states,markovchain-method (states), 111
steadyStates, *15–17*, *24*, *56*, *60*, *98*, *107*, 112
steadyStates,ctmc-method (ctmc-class), 20
steadyStates,markovchain-method (markovchain-class), 62
subchain, *61*, *78*, 113
subchain,markovchain-method (subchain), 113
summary, *53*
summary,markovchain-method (markovchain-class), 62

t,markovchain-method (markovchain-class), 62
tauchen, *102, 103*, 115
timeCorrelations, 116
timeCorrelations,markovchain-method (timeCorrelations), 116
timeRelaxations (timeCorrelations), 116
timeRelaxations,markovchain-method (timeCorrelations), 116
tm_abs, 118
toBoundedChain, *13*, *37*, 119, *125*
toBoundedChain,markovchain-method (toBoundedChain), 119
toDictionary, 120, *123, 124*
toDictionary,markovchain-method (toDictionary), 120
toDot, 122
toDot,markovchain-method (toDot), 122
toFile, *120, 121*, 123, *123*
toFile,markovchain-method (toFile), 123
toMermaid (toDot), 122
toMermaid,markovchain-method (toDot), 122
toNthOrder, 125
topologicalEntropy, *84*, 126
topologicalEntropy,markovchain-method (topologicalEntropy), 126
transientClasses (period), 85
transientClasses,markovchain-method (markovchain-class), 62
transientStates, *36*
transientStates (period), 85
transientStates,markovchain-method (markovchain-class), 62
transition2Generator, 127
transitionProbability, 128
transitionProbability,markovchain-method (transitionProbability), 128
txtProgressBar, *68*

urnModel, *13*, 129

verifyEmpiricalToTheoretical (verifyMarkovProperty), 130
verifyHomogeneity (verifyMarkovProperty), 130
verifyMarkovProperty, *10*, *105*, 130
verifyMarkovProperty(), *10*

zeros, 132