Package 'spgs'

March 28, 2015

Type Package

Encoding latin1

Title Statistical Patterns in Genomic Sequences

Version 1.0

Date 2015-03-27

Author Andrew Hart [aut, cre], Servet Martínez [aut], Universidad de Chile [cph], INRIA-Chile [cph]

Maintainer Andrew Hart <ahart@dim.uchile.cl>

Depends R (>= 3.0)

Enhances seqinr

Description A collection of statistical hypothesis tests and other techniques for identifying certain spatial relationships/phenomena in DNA sequences. In particular, it provides tests and graphical methods for determining whether or not DNA sequences comply with Chargaff's second parity rule or exhibit purine-pyrimidine parity. In addition, there are functions for efficiently simulating discrete state space Markov chains and testing arbitrary symbolic sequences of symbols for the presence of first-order Markovianness.
Also, it has functions for counting words/k-mers (and cylinder patterns) in arbitrary symbolic sequences. Functions which take a DNA sequence as input can handle sequences stored as SeqFastadna objects from the 'seqinr' package.

License GPL (>= 2)

NeedsCompilation yes

Classification/MSC 62F03, 62G10, 62M07, 62P10, 92D20

Repository CRAN

Date/Publication 2015-03-28 07:40:29

R topics documented:

`spgs-package` *Statistical Patterns in Genomic Sequences*

Description

provides functions for exploring and testing statistical properties and patterns in DNA sequences.

Details

Package:	spgs
Type:	Package
License:	GPL (>= 2)

This package provides a range of statistical tests for various properties of DNA and/or other genomic sequences. There are eight groups of functions:

Testing for Chargaff's second parity rule in bacteria and other DNA sequences `chargaff0.test`, `chargaff1.test`, `chargaff2.test`, `chargaff.gibbs.test`, `oligoProfile`

Testing for purine-pyrimidine parity in viruses and other DNA sequences `ag.test`, `agct.test`

Testing for Bernoulli/Markov processes `markov.test`, `diid.test`

Independence tests `diffsign.test`, `turningpoint.test`, `rank.test`, `lb.test`

Tests for uniform distribution `ks.unif.test`, `chisq.unif.test`

Simulation of random vectors, stochastic matrices, Bernoulli processes and Markov chains `simulateMarkovChain`, `estimateMarkovChain`, `rstochvec`, `rstochmat`, `rcspr2mat`

Functions for obtaining the complement or reverse complement of a DNA sequence `complement`, `reverseComplement`

Functions for counting words/k-mers and cylinders in symbolic sequences `pair.counts`, `triple.counts`, `quadruple.counts`, `cylinder.counts`

The word/k-mer counting functions are general and can deal with arbitrary symbolic sequences, not only DNA sequences.

Functions which take a DNA sequence as input are able to work with sequences stored as `SeqFastadna` objects generated by the **seqinr** package.

Author(s)

Andrew Hart and Servet Martínez

Maintainer: Andrew Hart <ahart@dim.uchile.cl>

References

Hart, A.G. and Martínez, S. (2011) Statistical testing of Chargaff's second parity rule in bacterial genome sequences. *Stoch. Models* **27(2)**, 1–46.

Hart, A.G. and Martínez, S. (2014) Markovianness and Conditional Independence in Annotated Bacterial DNA. *Stat. Appl. Genet. Mol. Biol.* **13(6)**, 693-716. arXiv:1311.4411 [q-bio.QM].

Hart, A.G. and Martínez, S. (2012) A Gibbs approach to Chargaff's second parity rule. *J. Stat. Phys.* **146(2)**, 408-422. arXiv:1105.0685 [math.pr].

See Also

`seqinr`

`ag.test` *Test of Purine-Pyrimidine Parity Based on Purine Count*

Description

Performs a test proposed by Hart and Martínez (2011) for the equivalence of the relative frequencies of purines ($A + G$) and pyrimidines ($C + T$) in DNA sequences. It does this by checking whether or not the mononucleotide frequencies of a DNA sequence satisfy the relationship A+G=C+T.

Usage

```
ag.test(x, type=c("interval", "simplex"))
```

Arguments

`x`	either a vector containing the relative frequencies of each of the 4 nucleotides A, C, G, T, a character vector representing a DNA sequence in which each element contains a single nucleotide, or a DNA sequence stored using the SeqFastadna class from the **seqinr** package.
`type`	Specifies one of two possible tests to perform, both of which are based on the same test statistic, but assuming different forms of the Dirichlet distribution under the null. "'`simplex`"' assumes a Dirichlet(1,1,1,1) distribution on the 3-simplex while "'`interval`"' assumes a Dirichlet(1,1) (uniform) distribution on the unit interval. The default is "'`interval`"'.

Details

The first argument may be a character vector representing a DNA sequence, a DNA sequence represented using the SeqFastadna class from the **seqinr** package, or a vector containing the relative frequencies of the A, C, G and T nucleic acids.

Let A, C, G and T denote the relative frequencies of the nucleotide bases appearing in a DNA sequence. This function carries out a statistical hypothesis test that the relative frequencies satisfy the relation $A + G = C + T$, or that purines $\{A, G\}$ occur equally as often as pyrimidines $\{C, T\}$ in a DNA sequence. The relationship can be rewritten as $A - T = C - G$, from which it is easy to see that the property being tested is a generalisation of Chargaff's second parity rule for mononucleotides, which states that $A = T$ and $C = G$. The test is set up as follows:

H_0: $A + G \neq C + T$
H_1: $A + G = C + T$

If '`type`' is set to "'`simplex`"', the vector (A, C, G, T) is assumed to come from a Dirichlet(1,1,1,1) distribution on the 3-simplex under the null hypothesis. Otherwise, if '`type`' is set to "'`interval`"', it is assumed under the null hypothesis that $(A + G, C + T)$ ~ Dirichlet(1,1) or, in other words, $A + G$ and $C + T$ are uniformly distributed on the unit interval and satisfy $A + G + C + T = 1$.

In both cases, the test statistic is $\eta_V^* = |A + G - 0.5|$.

Value

A list with class "htest.ext" containing the following components:

`statistic`	the value of the test statistic.
`p.value`	the p-value of the test. Only included if 'no.p.value' is 'FALSE'.
`method`	a character string indicating what type of test was performed.
`data.name`	a character string giving the name of the data.
`estimate`	the probability vector used to derive the test statistic.
`stat.desc`	a brief description of the test statistic.
`null`	the null hypothesis (H_0) of the test.
`alternative`	the alternative hypothesis (H_1) of the test.

Author(s)

Andrew Hart and Servet Martínez

References

Hart, A.G. and Martínez, S. (2011) Statistical testing of Chargaff's second parity rule in bacterial genome sequences. *Stoch. Models* **27(2)**, 1–46.

See Also

`chargaff0.test`, `chargaff1.test`, `chargaff2.test`, `agct.test`, `chargaff.gibbs.test`

Examples

```
#Demonstration on real viral sequence
data(pieris)
ag.test(pieris)
ag.test(pieris, type="simplex")

#Simulate synthetic DNA sequence that does not exhibit Purine-Pyrimidine parity
trans.mat <- matrix(c(.4, .1, .4, .1, .2, .1, .6, .1, .4, .1, .3, .2, .1, .2, .4, .3),
ncol=4, byrow=TRUE)
seq <- simulateMarkovChain(500000, trans.mat, states=c("a", "c", "g", "t"))
ag.test(seq)
```

`agct.test` *Test of Purine-Pyrimidine Parity Based on Euclidean distance*

Description

Performs a test proposed by Hart and Martínez (2011) for the equivalence of the relative frequencies of purines ($A + G$) and pyrimidines ($C + T$) in DNA sequences. It does this by checking whether or not the mononucleotide frequencies of a DNA sequence satisfy the relationship A+G=C+T.

Usage

```
agct.test(x, alg=c("exact", "simulate", "lower", "Lower", "upper"), n)
```

Arguments

`x`	either a vector containing the relative frequencies of each of the 4 nucleotides A, C, G, T, a character vector representing a DNA sequence in which each element contains a single nucleotide, or a DNA sequence stored using the SeqFastadna class from the **seqinr** package.
`alg`	the algorithm for computing the p-value. If set to "'`simulate`"', the p-value is obtained via Monte Carlo simulation. If set to "'`lower`"', an analytic lower bound on the p-value is computed. If set to "'`upper`"', an analytic upper bound on the p-value is computed. "'`lower`"' and "'`upper`"' are based on formulae in Hart and Martínez (2011). a Tighter (though unpublished) lower bound on the p-value may be obtained by specifying "'`Lower`"'. If 'alg' is specified as "'`exact`"' (the default value), the p-value for the test is computed exactly.
`n`	The number of replications to use for Monte Carlo simulation. If computationally feasible, a value >= 10000000 is recommended.

Details

The first argument may be a character vector representing a DNA sequence, a DNA sequence represented using the SeqFastadna class from the **seqinr** package, or a vector containing the relative frequencies of the A, C, G and T nucleic acids.

Let A, C, G and T denote the relative frequencies of the nucleotide bases appearing in a DNA sequence. This function carries out a statistical hypothesis test that the relative frequencies satisfy the relation $A + G = C + T$, or that purines $\{A, G\}$ occur equally as often as pyrimidines $\{C, T\}$ in a DNA sequence. The relationship can be rewritten as $A - T = C - G$, from which it is easy to see that the property being tested is a generalisation of Chargaff's second parity rule for mononucleotides, which states that $A = T$ and $C = G$. The test is set up as follows:

H_0: $A + G \neq C + T$
H_1: $A + G = C + T$

The vector (A, C, G, T) is assumed to come from a Dirichlet(1,1,1,1) distribution on the 3-simplex under the null hypothesis.

The test statistic η_V is the Euclidean distance from the relative frequency vector (A, C, G, T) to the closest point in the square set $\theta_V = \{(x, y, 1/2 - x, 1/2 - y) : 0 <= x, y <= 1/2\}$, which divides the 3-simplex into two equal parts. η_V lies in the range $[0, \sqrt{3/8}]$.

Value

A list with class "htest.ext" containing the following components:

`statistic`	the value of the test statistic.
`p.value`	the p-value of the test.
`method`	a character string indicating what type of test was performed.
`data.name`	a character string giving the name of the data.
`estimate`	the probability vector used to derive the test statistic.
`stat.desc`	a brief description of the test statistic.
`null`	the null hypothesis (H_0) of the test.
`alternative`	the alternative hypothesis (H_1) of the test.

Note

`agct.test(x, alg="upper")` is equivalent to `ag.test(x,alg="simplex")` except that the p-value computed using the formula for '`alg="upper"`' is exact for the test statistic η_V^* used in `ag.test`, whereas it is merely an upper bound on the p-value for η_V.

Author(s)

Andrew Hart and Servet Martínez

References

Hart, A.G. and Martínez, S. (2011) Statistical testing of Chargaff's second parity rule in bacterial genome sequences. *Stoch. Models* **27(2)**, 1–46.

See Also

`chargaff0.test`, `chargaff1.test`, `chargaff2.test`, `ag.test`, `chargaff.gibbs.test`

Examples

```
#Demonstration on real viral sequence
data(pieris)
agct.test(pieris)

#Simulate synthetic DNA sequence that does not exhibit Purine-Pyrimidine parity
trans.mat <- matrix(c(.4, .1, .4, .1, .2, .1, .6, .1, .4, .1, .3, .2, .1, .2, .4, .3),
ncol=4, byrow=TRUE)
seq <- simulateMarkovChain(500000, trans.mat, states=c("a", "c", "g", "t"))
agct.test(seq)
```

`array2vector` *Convert Arrays and Tables to Vectors with Named Elements*

Description

Convert an array/table into an integer vector, preserving the names corresponding to each element in a sensible way. These functions differ from `as.vector` in that they name each element of the resulting vector by combining appropriate names from the various dimensions that together uniquely identify the position of each element in the original array/table.

Usage

```
array2vector(x, sep=".", sort=FALSE, rev=FALSE, ...)
table2vector(x, sep=".", sort=FALSE, rev=FALSE, ...)
```

Arguments

`x`	an array or table.
`sep`	a character string to be used to separate the name corresponding to each dimension when constructing the element names for the vector. The default value is `"."`.
`sort`	Should the elements in the resulting vector be sorted in lexicographic order according to the names they are assigned? The default is `FALSE`.
`rev`	For the purposes of sorting, should the names of the vector's elements be read from right-to-left, i.e. in reverse order? The default is `FALSE`.
`...`	Arguments to be passed from or to other functions.

Details

`table2vector` is merely a convenience alias for `array2vector`, which converts a multi-dimensional array or table to a vector using `as.vector`, but names each of the elements in the resulting vector according to the names contained in its `dimnames` attribute.

the name of each element is constructed by concatenating names (one from each `dimnames` member) seperated by the value specified in `sep`.

Note that dimensions of `x` which lack a corresponding vector of names in the `dimnames` attribute will be assigned a names vector of the form `1:d` where `d` is the dimension size specified in the corresponding entry of the `dim` attribute.

Value

An integer vector with names set as described in 'Details'.

Author(s)

Andrew Hart and Servet Martínez

See Also

pair.counts, triple.counts, quadruple.counts, cylinder.counts

Examples

```
a <- array(1:8, dim=rep(2,3), dimnames=list(c("a","b"), c("x","p"), c("v","u")))
array2vector(a)
array2vector(a, sep="")
array2vector(a, sep="", sort=TRUE)
array2vector(a, sep="", sort=TRUE, rev=TRUE)
array2vector(a, sep="", sort=TRUE, decreasing=TRUE)
```

---

chargaff.gibbs.test *Test of CSPR for Dinucleotides Under Gibbs Distribution*

Description

Performs a test of Chargaff's second parity rule (CSPR) for dinucleotides under a Gibbsian assumption on the DNA sequence, which was proposed in Hart and Martínez (2012).

Usage

```
chargaff.gibbs.test(x, maxLag=200)
```

Arguments

x	either a character vector representing a DNA sequence in which each element contains a single nucleotide, or a DNA sequence stored using the SeqFastadna class from the **seqinr** package.
maxLag	The maximum number of lags (cylinder lengths) to use in computing variances. the default value is '200'.

Details

This function performs a test of Chargaff's second parity rule for dinucleotides assuming the DNA sequence was generated by a Gibbs distribution. Under the null hypothesis, the test statistic η is asymptotically χ^2 on 5 degrees of freedom.

The test is set up as follows:

H_0: the sequence complies with CSPR for dinucleotides
H_1: the sequence does not comply with CSPR for dinucleotides

Value

A list with class "htest" containing the following components:

`statistic`	the value of the test statistic.
`p.value`	the p-value of the test.
`method`	a character string indicating what type of test was performed.
`data.name`	a character string giving the name of the data.
`FHat`	the 5-element vector $n\hat{F}$ used in calculating the test statistic.
`pairs`	the stochastic matrix of dinucleotide counts used to derive $n\hat{F}$.
`v`	The asymptotic covariance matrix of $n\hat{F}$.
`n`	the length of the DNA sequence.
`cutoff`	the actual number of lags used by the algorithm to calculate covariances.
`maxCutoff`	the value specified for the maxLag parameter when the test was performed.

Author(s)

Andrew Hart and Servet Martínez

References

Hart, A.G. and Martínez, S. (2012) A Gibbs approach to Chargaff's second parity rule. *J. Stat. Phys.* **146(2)**, 408-422.

See Also

`chargaff0.test`, `chargaff1.test`, `chargaff2.test`, `agct.test`, `ag.test`

Examples

```
#Demonstration on real bacterial sequence
data(nanoarchaeum)
chargaff.gibbs.test(nanoarchaeum)

#Simulate synthetic DNA sequence that does not satisfy Chargaff's second parity rule
trans.mat <- matrix(c(.4, .1, .4, .1, .2, .1, .6, .1, .4, .1, .3, .2, .1, .2, .4, .3),
ncol=4, byrow=TRUE)
seq <- simulateMarkovChain(500000, trans.mat, states=c("a", "c", "g", "t"))
chargaff.gibbs.test(seq)
```

`chargaff0.test` *Vector Test of Chargaff's Second Parity Rule (CSPR) for Mononucleotides*

Description

Performs the vector test of Chargaff's second parity rule (CSPR) for mononucleotides proposed in Hart and Martínez (2001).

Usage

```
chargaff0.test(x, alg=c("exact", "simulate", "lower", "upper", "Lower", "Upper"), n,
no.p.value=FALSE)
```

Arguments

`x`	either a vector containing the relative frequencies of each of the 4 nucleotides A, C, G, T, a character vector representing a DNA sequence in which each element contains a single nucleotide, or a DNA sequence stored using the SeqFastadna class from the **seqinr** package.
`alg`	the algorithm for computing the p-value. If set to "'`simulate`"', the p-value is obtained via Monte Carlo simulation. If set to "'`lower`"', an analytic lower bound on the p-value is computed. If set to "'`upper`"', an analytic upper bound on the p-value is computed. "'`lower`"' and "'`upper`"' are based on formulae in Hart and Martínez (2011). a Tighter (though unpublished) lower /upper bound on the p-value may be obtained by specifying "'`Lower`"'/"'`Upper`"'. If 'type' is specified as "'`exact`"' (the default value),the p-value for the test is computed exactly for small values of the test statistic and crudely approximated for large values. See the note below for further details.
`n`	The number of replications to use for Monte Carlo simulation. If computationally feasible, a value >= 10000000 is recommended.
`no.p.value`	If '`TRUE`', do not compute the p-value. The default is '`FALSE`'.

Details

The first argument may be a character vector representing a DNA sequence, a DNA sequence represented using the SeqFastadna class from the **seqinr** package, or a vector containing the relative frequencies of the A, C, G and T nucleic acids.

Letting A, C, G and T denote the relative frequencies of their corresponding nucleic acids, this function performs the following hypothesis test:

H_0: $A \neq T$ or $C \neq G$
H_1: $A = T$ and $C = G$

The vector (A, C, G, T) is assumed to come from a Dirichlet(1,1,1,1) distribution on the 3-simplex under the null hypothesis.

The test statistic is $\eta_0 = \sqrt{(A-T)^2/2 + (C-G)^2/2}$.

Value

A list with class "htest.ext" containing the following components:

`statistic`	the value of the test statistic.
`p.value`	the p-value of the test. Only included if no.p.value is FALSE.
`method`	a character string indicating what type of test was performed.
`data.name`	a character string giving the name of the data.
`estimate`	the probability vector used to derive the test statistic.
`stat.desc`	a brief description of the test statistic.
`null`	the null hypothesis (H_0) of the test.
`alternative`	the alternative hypothesis (H_1) of the test.

Note

Currently, regardless of the algorithm (`alg`) selected, the p-value or bound is only computed correctly when the test statistic is smaller than or equal to $\sqrt{2}/4$. A value of 1 is returned when the test statistic is greater than $\sqrt{2}/4$. This is not accurate, but shouldn't matter as it is well within the acceptance region of the null hypothesis.

The algebraically computed bounds on the p-value obtained when '`alg`' is set to either "'`lower`"'or "'`upper`"' are not as tight as those corresponding to "'`Lower`"' and "'`Upper`"', which should be generally preferred. However, "'`exact`"' or "'`simulate`"' should be employed for real- world analysis.

'`no.p.value`' suppresses computation of the p-value when it is set to '`TRUE`'. This may be useful wen using this function to help simulate the test statistic.

Author(s)

Andrew Hart and Servet Martínez

References

Hart, A.G. and Martínez, S. (2011) Statistical testing of Chargaff's second parity rule in bacterial genome sequences. *Stoch. Models* **27(2)**, 1–46.

See Also

`chargaff1.test`, `chargaff2.test`, `agct.test`, `ag.test`, `chargaff.gibbs.test`

Examples

```
#Demonstration on real bacterial sequence
data(nanoarchaeum)
chargaff0.test(nanoarchaeum)

#Simulate synthetic DNA sequence that does not satisfy Chargaff's second parity rule
trans.mat <- matrix(c(.4, .1, .4, .1, .2, .1, .6, .1, .4, .1, .3, .2, .1, .2, .4, .3),
ncol=4, byrow=TRUE)
```

```
seq <- simulateMarkovChain(500000, trans.mat, states=c("a", "c", "g", "t"))
chargaff0.test(seq)
```

---

chargaff1.test *Matrix Test of CSPR for Mononucleotides*

Description

Performs a test of Chargaff's second parity rule (CSPR) for mononucleotides on a genomic sequence using a 4X4 stochastic matrix estimated from the sequence. The test was proposed by Hart and Martínez (2011).

Usage

```
chargaff1.test(x, alg=c("table", "simulate", "upper"), n, no.p.value=FALSE)
```

Arguments

x	either a vector containing the relative frequencies of each of the 4 nucleotides A, C, G, T, a character vector representing a DNA sequence in which each element contains a single nucleotide, or a DNA sequence stored using the SeqFastadna class from the **seqinr** package.
alg	the algorithm for computing the p-value. If set to "'simulate"', the p-value is obtained via Monte Carlo simulation. If set to "'upper"', an analytic upper bound on the p-value is computed. "'upper"' are based on formulae in Hart and Martínez (2011). If 'type' is specified as "'table"' (the default value),the p-value for the test is obtained from a linear interpolation of a look-up table. See the note below for further details.
n	The number of replications to use for Monte Carlo simulation. If computationally feasible, a value >= 10000000 is recommended.
no.p.value	If 'TRUE', do not compute the p-value. The default is 'FALSE'.

Details

The first argument may be a character vector representing a DNA sequence, a DNA sequence represented using the SeqFastadna class from the **seqinr** package, or a vector containing the relative frequencies of the A, C, G and T nucleic acids.

This function performs a test of Chargaff's second parity rule for mononucleotides based on a 4X4 stochastic matrix P estimated from the empirical dinucleotide distribution of a genomic sequence . The a, b) entry of P gives the empirical probability (relative frequency) that a nucleotide a is followed by a nucleotide b in the sequence. The test is set up as follows:

H_0: the sequence (or matrix P) does not comply with CSPR for mononucleotides
H_1: the sequence (or matrix P) complies with CSPR for mononucleotides

Value

A list with class "htest.ext" containing the following components:

`statistic`	the value of the test statistic.
`p.value`	the p-value of the test. Only included if no.p.value is FALSE.
`method`	a character string indicating what type of test was performed.
`data.name`	a character string giving the name of the data.
`f`	the 2-element vector used in calculating the test statistic.
`estimate`	the stochastic matrix P used to derive the test statistic.
`stat.desc`	a brief description of the test statistic.
`null`	the null hypothesis (H_0) of the test.
`alternative`	the alternative hypothesis (H_1) of the test.

Note

Currently, the look-up table that is employed when '`alg`' is set to "'`table`"' does not provide an accurate p-value when the statistic is smaller than 0.00806. Care should be taken when adjusting p-values for multiple testing.

The algebraically computed upper bound on the p-value obtained when '`alg`' is set to "'`upper`"' is not very tight and not suitable for real- world applications.

'`no.p.value`' suppresses computation of the p-value when it is set to '`TRUE`'. This may be useful wen using this function to help simulate the test statistic.

Author(s)

Andrew Hart and Servet Martínez

References

Hart, A.G. and Martínez, S. (2011) Statistical testing of Chargaff's second parity rule in bacterial genome sequences. *Stoch. Models* **27(2)**, 1–46.

See Also

`chargaff0.test`, `chargaff2.test`, `agct.test`, `ag.test`, `chargaff.gibbs.test`

Examples

```
#Demonstration on real bacterial sequence
data(nanoarchaeum)
chargaff1.test(nanoarchaeum)

#Simulate synthetic DNA sequence that does not satisfy Chargaff's second parity rule
trans.mat <- matrix(c(.4, .1, .4, .1, .2, .1, .6, .1, .4, .1, .3, .2, .1, .2, .4, .3),
ncol=4, byrow=TRUE)
seq <- simulateMarkovChain(500000, trans.mat, states=c("a", "c", "g", "t"))
chargaff1.test(seq)
```

chargaff2.test *Matrix Test of CSPR for Dinucleotides*

Description

Performs the matrix test of Chargaff's second parity rule (CSPR) for dinucleotides proposed in Hart and Martínez (2011).

Usage

```
chargaff2.test(x, alg=c("table", "simulate", "upper"), n, no.p.value=FALSE)
```

Arguments

x	either a vector containing the relative frequencies of each of the 4 nucleotides A, C, G, T, a character vector representing a DNA sequence in which each element contains a single nucleotide, or a DNA sequence stored using the SeqFastadna class from the **seqinr** package.
alg	the algorithm for computing the p-value. If set to "'simulate"', the p-value is obtained via Monte Carlo simulation. If set to "'upper"', an analytic upper bound on the p-value is computed. "'upper"' are based on formulae in Hart and Martínez (2011). If 'type' is specified as "'table"' (the default value),the p-value for the test is obtained from a linear interpolation of a look-up table. See the note below for further details.
n	The number of replications to use for Monte Carlo simulation. If computationally feasible, a value >= 10000000 is recommended.
no.p.value	If 'TRUE', do not compute the p-value. The default is 'FALSE'.

Details

This function performs a test of Chargaff's second parity rule for dinucleotides based on a 4X4 stochastic matrix $\hat{P}$ estimated from the empirical dinucleotide distribution of a genomic sequence . The $a, b)$ entry of $\hat{P}$ gives the empirical probability (relative frequency) that a nucleotide a is followed by a nucleotide b in the sequence. The test is set up as follows:

H_0: the sequence (or matrix $\hat{P}$) does not comply with CSPR for dinucleotides
H_1: the sequence (or matrix $\hat{P}$) complies with CSPR for dinucleotides

Value

A list with class "htest.ext" containing the following components:

statistic	the value of the test statistic.
p.value	the p-value of the test. Only included if no.p.value is FALSE.
method	a character string indicating what type of test was performed.
data.name	a character string giving the name of the data.

`f`	the 5-element vector used in calculating the test statistic.
`estimate`	the stochastic matrix $\hat{P}$ used to derive the test statistic.
`stat.desc`	a brief description of the test statistic.
`null`	the null hypothesis (H_0) of the test.
`alternative`	the alternative hypothesis (H_1) of the test.

Note

Currently, the look-up table that is employed when 'alg' is set to "'table"' does not provide an accurate p-value when the statistic is smaller than 0.05899. Care should be taken when adjusting p-values for multiple testing.

The algebraically computed upper bound on the p-value obtained when 'alg' is set to "'upper"' is not very tight and not suitable for real- world applications.

'no.p.value' suppresses computation of the p-value when it is set to 'TRUE'. This may be useful wen using this function to help simulate the test statistic.

Author(s)

Andrew Hart and Servet Martínez

References

Hart, A.G. and Martínez, S. (2011) Statistical testing of Chargaff's second parity rule in bacterial genome sequences. *Stoch. Models* **27(2)**, 1–46.

See Also

chargaff0.test, chargaff1.test agct.test, ag.test, chargaff.gibbs.test

Examples

```
#Demonstration on real bacterial sequence
data(nanoarchaeum)
chargaff2.test(nanoarchaeum)

#Simulate synthetic DNA sequence that does not satisfy Chargaff's second parity rule
trans.mat <- matrix(c(.4, .1, .4, .1, .2, .1, .6, .1, .4, .1, .3, .2, .1, .2, .4, .3),
ncol=4, byrow=TRUE)
seq <- simulateMarkovChain(500000, trans.mat, states=c("a", "c", "g", "t"))
chargaff2.test(seq)
```

`chisq.unif.test` *Test of Uniformity Based on Pearsons's Chi-Squared test*

Description

Tests if a set of data points is uniformly distributed over a specified interval [a,b].

Usage

```
chisq.unif.test(x, bins=NULL, interval=c(0,1), min.bin.size=10, all.inside=TRUE,
rightmost.closed=TRUE, ...)
```

Arguments

`x`	A numeric vector of data values.
`bins`	If specified, the number of bins to use to discretise the interval. Otherwise, the number of bins will be chosen automatically.
`interval`	A two-element vector giving the support of the uniform distribution. The default is `c(0,1)`.
`min.bin.size`	The minimum number of data points to have in each bin. If bins cannot be chosen without violating this constraint, an error is generated. The default is 10. This parameter is ignored if '`bins`' is specified.
`all.inside`	Determines if data points outside the interval should be counted as belonging to the extremal bins. The default is '`TRUE`'.
`rightmost.closed`	Determines if data points that coinside with`interval[2]` are counted as belonging to the last bin. This parameter only has an effect if `all.inside` is set to `FALSE`. The default is `TRUE`.
`...`	Additional parameters to be passed to `chisq.test`.

Details

This function tests the fit of a set of data points to a uniform distribution on $[a, b]$ by partitioning $[a, b]$ into '`bins`' bins, counting how many points fall in each bin and then testing that the points are equally distributed among the bins using Pearson's chi-squared test.

When '`bins`' is not specified, its value is selected using the following heuristic. Let n be the number of data points. If $n > 200$, then '`bins`' is set to 20. Otherwise, '`bins`' is set to $n/10$. Next,while there is a bin containing fewer than '`min.bin.size`' points in the resulting partition, '`bins`' is decremented by one. This process stops either when '`bins`' is equal to 1 or every bin contains at least '`min.bin.size`' points.

Value

A list with class "htest" containing the following components:

statistic	the value of the test statistic.
parameter	A vector containing the degrees of freedom of the chi-squared test (df), the left end of the interval (a) and the right end of the interval (b).
p.value	the p-value of the test.
method	a character string indicating what type of test was performed.
data.name	a character string giving the name of the data.
bins	The number of bins used for the test.
min.bin.size	The minimum bin size.
interval	The interval used for the test.

Warning

The arguments 'all.inside' and 'rightmost.closed' are included for experimentation and should be altered with caution.

Author(s)

Andrew Hart and Servet Martínez

See Also

chisq.test, findInterval, ks.unif.test

Examples

```
#Generate an IID uniform(0,1) sequence
u <- runif(1000)
chisq.unif.test(u)
```

---

complement *Complement of a DNA/RNA Sequence*

Description

Compute the complement of a DNA or RNA sequence.

Usage

```
## Default S3 method:
complement(x, content=c("dna", "rna"),
case=c("lower", "upper", "as is"), ...)
## S3 method for class 'SeqFastadna'
complement(x, ...)
## S3 method for class 'list'
complement(x, ...)
```

Arguments

x	A character vector, an object that can be coersed to a character vector or a list of objects that canbe be converted to character vectors. this argument can also be a SeqFastadna object provided by the **seqinr** package.
content	The content type of sequence(s). At present, supported types include "dna" and "rna". the default type is "dna".
case	Determines how symbols in x should be treated before translating them into their complements. "lower", the default behaviour, converts all symbols to lowercase while "upper" converts them to uppercase. "as is" allows the symbols to pass unchanged so that the case of each output symbol matches that of the corresponding input symbol.
...	Arguments to be passed from or to other functions.

Details

If x is a SeqFastadna object or a character vector in which each element is a single nucleobase, then it represents a single sequence and its complementary sequence will be returned in the same form.

On the other hand, if x is a vector of character strings, each of which represents a nucleic sequence, then the result will bea a character vector in which each element contains the complement of the corresponding element in x as a character string.

Value

According to the input x, a character vector, SeqFastadna object or list containing the complement(s) of the sequence(s) in x.

Author(s)

Andrew Hart and Servet Martínez

See Also

reverseComplement

Examples

```
complement("actg")
complement(c("t", "g", "a"))

#List of sequences
some.dna <- list("atgcgtcgttaa", c("g", "t", "g", "a", "a", "a"))
complement(some.dna)

#RNA sequence example
complement(c("a", "u", "g"), content="rna")

#Examples of lowercase, uppercase and as-is conversion
mixed.case <- c("t", "G", "g", "C", "a")
complement(mixed.case)
```

```
complement(mixed.case, case="upper")
complement(mixed.case, case="as is")
```

`cylinder.counts` *Count Cylinders (Fixed-Offset Patterns) in Character Vectors*

Description

Count fixed tuples of not necessarily adjacent symbols/elements in a character vector.

Usage

```
cylinder.counts(x, cylinder, case=c("lower", "upper", "as is"), circular=TRUE)
```

Arguments

`x`	a character vector or an object that can be coersed to a character vector.
`cylinder`	A vector of indices specifying the form of cylinders to count. See 'Details'.
`case`	determines how labels for the array should be generated: in lowercase, in uppercase or left as is, in which case labels such as "b" and "B" will be seen as distinct symbols and counted separately.
`circular`	Determines if the vector should be treated as circular or not. The default is `TRUE`, meaning that the start and end of the sequence will be joined together for the purpose of counting.

Details

`cylinder` represents a set of symbol patterns that one wishes to count in the sequence `x`. For example, if `cylinder` is `c(1,3,5)`, then this function will count occurrences of all patterns of the form '`u.v.w`', where '`u`', '`v`' and '`w`' can be any symbol present in `x` and `.` stands for a symbol whose value is not relevant to the pattern.

Suppose that `x` is a sequence of the nucleotides `a`, `c`, `g` and `t`. Then, `cylinder=1:2` will count the occurrences of all 16 dinucleotides: `aa`, `ac`, `ag`, `at`, `ca`, `cc`, ... In contrast, `cylinder=c(1,3)` will counts 16 sets of trinucleotides: `a.a`, `a.c`, `a.g`, `a.t`, `c.a`, `c.c`, `c.g`, ... the dot "`.`" stands for any nucleotide, so that `a.c` represents the set `aac`, `acc`, `agc`, `atg`. In both of these examples, a 4×4 array of counts will be produced, but in the first case the array will represent counts of dinucleotides, while in the second case it will represent counts of groups of trinucleotides.

If `circular` is `TRUE`, the vector `x` is treated as circular so that the some of all the counts in the resulting array is equal to the length of the vector and the sums across all dimentions of the array are equivalent, that is: writing
`counts <- cylinder.counts(x, cylinder=c(1,3,5))`
for some character sequence x, then
`apply(counts,1,sum)`, `apply(counts,2,sum)` and `apply(counts,3,sum)`
will all be identical.

On the other hand, if `circular` is `FALSE`, the sum of all the entries in the counts array will be less than the length of the vector and there will be a discrepancy between the sums over the various dimensions.

Value

An n-dimensional array of counts, where n is the length of `cylinder`.

Note

`table`is more efficient (by almost a factor of 2) at computing the counts of cylinders of length 1, whereas `cylinder.counts` is faster and uses less memory than for cylinders of length greater than 1.

Author(s)

Andrew Hart and Servet Martínez

See Also

`pair.counts`, `triple.counts`, `quadruple.counts`, `array2vector`, `table2vector`

Examples

```
#Generate an IID uniform DNA sequence
seq <- simulateMarkovChain(5000, matrix(0.25, 4, 4), states=c("a","c","g","t"))
cylinder.counts(seq, 1) #essentially the same as unclass(table(seq))
cylinder.counts(seq, 1:5) #counts of all 5-mers in the sequence

 #counts of all patterns of the form a.b where a and b represent
 #specific symbols and . denotes an arbitrary symbol.
 pat <- cylinder.counts(seq, c(1, 3))
#For example, pat["a","c"] gives the number of times that any of
#the following 4 words appears in the sequence:  aac, acc, agc, atc.
identical(cylinder.counts(seq, c(1,3)), apply(cylinder.counts(seq, 1:3), c(1, 3), sum))

##some relationships between cylinder.counts and other functionns
identical(cylinder.counts(seq, 1:2), pair.counts(seq))
identical(cylinder.counts(seq, 1:3), triple.counts(seq))
identical(cylinder.counts(seq, 1:4), quadruple.counts(seq))

#The following relationship means that counts on circular sequences are
#invariant under translationn
identical(cylinder.counts(seq, 1:6), cylinder.counts(seq, 10:15))

#Treating seq as non circular, most of the preceding relationships continue to hold
identical(cylinder.counts(seq, 1:2, circular=FALSE),
  pair.counts(seq, circular=FALSE))
identical(cylinder.counts(seq, 1:3, circular=FALSE),
triple.counts(seq, circular=FALSE))
identical(cylinder.counts(seq, 1:4, circular=FALSE),
  quadruple.counts(seq, circular=FALSE))
#The following relationship no longer holds; that is, non-circular counts
#are not invariant under translation.
identical(cylinder.counts(seq, 1:6, circular=FALSE),
  cylinder.counts(seq, 10:15, circular=FALSE))
```

`diffsign.test` *the Differents-Sign Test of Statistical Independence*

Description

Tests for a trend in a data series by comparing the number of positive differences between successive elements in the series to the number expected in an i.i.d. series.

Usage

```
diffsign.test(x)
```

Arguments

`x`	a numeric vector or univariate time series.

Details

Perform a test for trend based on the signs of successive differences in a data series. #this function counts the number of positive successive differences in the data, standardises #it to have mean 0 and variance 1 and asymptotically tests it against a standard normal distribution. the test statistic is:

D = (pd - mu)/sigma, where
pd is the number of positive differences in the data series,
mu = (n-1)/2,
sigma = sqrt((n+1)/12) and
n is the number of points in the data series.

The test is set up as follows:

H_0: the data series is i.i.d. (not trending)
H_1: the data series is not i.i.d. (trending)

Value

A list with class "htest" containing the following components:

`statistic`	the value of the test statistic.
`p.value`	the p-value of the test.
`method`	a character string indicating what type of test was performed.
`data.name`	a character string giving the name of the data.
`n`	the number of points in the data series.
`mu`	The expected number of positive differences that would be seen in an i.i.d. series.
`sigma`	The standard deviation of the number of positive differences that would be seen in an i.i.d. series.

Note

Missing values are not handled.

Points followed by a point having the exact same value are removed from the data series before computing the test statistic.

This test is useful for detecting linear trends in data series.

Author(s)

Andrew Hart and Servet Martínez

References

Brockwell, Peter J., Davis, Richard A. (2002) *Introduction to Time Series and Forecasting*. Springer Texts in Statistics, Springer-Verlag, New York.

See Also

turningpoint.test, rank.test, lb.test markov.test, diid.test,

Examples

```
#Generate an IID standard normal sequence
n <- rnorm(1000)
diffsign.test(n)
```

---

diid.disturbance *Construct feasible Random Noise Generating a Bernoulli Process*

Description

Produces a sequence of random noise which would generate an observed sequence of finite symbols provided that the sequence of symbols results from a Bernoulli process.

Usage

```
diid.disturbance(x, random = TRUE,  estimates = FALSE)
```

Arguments

x	A sequence of finite symbols represented as a character vector.
random	This can be a logical value or a number in the range 0-1. If 'TRUE', random noise will be generated. If 'FALSE', the constant value 0.5 will be used as the noise source. If a value in the range 0-1 is specified, that value will be used as a constant noise source. the default value is 'TRUE'.
estimates	A logical value specifying if the distribution estimated for the Bernoulli process should be included in the return.

Value

If 'estimates' is 'TRUE', returns a list containing the following components:

disturbance	the sequence of random noise as a numeric vector.
stat.dist	The stationary distribution estimated from x.

Otherwise, if 'estimate' is 'FALSE', returns the sequence of random noise as a numeric vector.

Author(s)

Andrew Hart and Servet Martínez

See Also

markov.test, diid.test, diid.disturbance

---

diid.test *A Test for a Bernoulli Scheme (IID Sequence)*

Description

Tests whether or not a data series constitutes a Bernoulli scheme, that is, an independent and identically distributed (IID) sequence of symbols, by inferring the sequence of IID U(0,1) random noise that might have generated it.

Usage

```
diid.test(x, type = c("lb", "ks"), method = "holm", lag = 20, ...)
```

Arguments

x	the data series as a vector.
type	the procedures to use to test whether or not the noise series is independently and identically distributed on the unit interval. See 'Details'.
method	the correction method to be used for adjusting the p-values. It is identical to the method argument of the p.adjust function, which is called to adjust the p-values.
lag	the number of lags to use when applying the Ljung-Box (portmanteau) test (lb.test).
...	parameters to pass on to functions that can be subsequently called.

Details

This function tests if a symbolic sequence is a Bernoulli scheme, that is, independently and identically distributed (IID). It does this by reverse- engineering the sequence to obtain a sample of the kind of output from a pseudo- random number generator that would have produced the observed sequence if it had been generated by simulating an IID sequence. The sample output is then tested to see if it is an independent and identically distributed siequence of uniform numbers in the range 0-1. this involves the application of at least two tests, one for independence and another for uniformity over the unit interval. One concludes that the sequence is IID if the sample output passes the tests (that is, all null hypotheses are accepted) and not IID otherwise.

The test is set up as follows:

H_0: the sequence is IID
H_1: the sequence is not IID

To simplify the use of the test, correction for multiple testing is carried out, which yields a single adjusted p- value. If this p-value is less than the significance level established for the test procedure, the null hypothesis of Markovianness is rejected. Otherwise, the null hypothesis should be accepted.

To correctly apply the test, use the `type` argument to specify at least one test of independence and one test of uniformity from the options displayed in the following table.

Category	Function	Test
Uniformity	`ks.unif.test`	Kolmogorov-Smirnov test for uniform$(0,1)$ data
	`chisq.unif.test`	Pearson's chi-squared test for discrete uniform data,
Independence	`lb.test`	Ljung-Box Q test for uncorrelated data
	`diffsign.test`	signed difference test of independence
	`turningpoint.test`	turning point test of independence
	`rank.test`	rank test of independence

If `type` is not specified, `lb.test` and `ks.unif.test` are used by default.

As this procedure performs multiple tests in order to assess if the sequence is IID, it is necessary to adjust the p-values for multiple testing. By default, the Holm-Bonferroni method (`holm`) is used to correct for multiple testing, but this can be overridden via the `method` argument. The adjusted p-values are displayed when the result of the test is printed.

The smallest adjusted p-value constitutes the overall p-value for the test. If this p-value is less than the significance level fixed for the test procedure, the null hypothesis of the sequence beingIID is rejected. Otherwise, the null hypothesis should be accepted.

Value

A list with class "multiplehtest" containing the following components:

`method`	the character string "Composite test for a Bernoulli process".
`statistics`	the values of the test statistic for all the tests.
`parameters`	parameters for all the tests. Exactly one parameter is recorded for each test, for example, `df` for `lb.test`. Any additional parameters are not saved, for example, the `a` and `b` parameters of `chisq.unif.test`.

`p.values`	p-values of all the tests.
`methods`	a vector of character strings indicating what type of tests were performed.
`adjusted.p.values`	the adjusted p-values.
`data.name`	a character string giving the name of the data.
`adjust.method`	indicates which correction method was used to adjust the p-values for multiple testing.
`estimate`	the transition matrix estimated to fit a first-order Markov chain to the data and used to generate the infered random disturbance

.

Note

Sometimes, a warning message advising that ties should not be present for the Kolmogorov-Smirnov test can arise when analysing long sequences. If you do receive this warning, it means that the results of the Kolmogorov-Smirnov test (`ks.unif.test`) should not be trusted. In this case, Pearson's chi-squared test (`chisq.unif.test`) should be used instead of the Kolmogorov-Smirnov test.

Author(s)

Andrew Hart and Servet Martínez

References

Although This test procedure is unpublished, it is derived by making appropriate modifications to the test for first-order Markovianness described in the following two references.

Hart, A.G. and Martínez, S. (2011) Statistical testing of Chargaff's second parity rule in bacterial genome sequences. *Stoch. Models* **27(2)**, 1–46.

Hart, A.G. and Martínez, S. (2014) Markovianness and Conditional Independence in Annotated Bacterial DNA. *Stat. Appl. Genet. Mol. Biol.* **13(6)**, 693-716. arXiv:1311.4411 [q-bio.QM].

See Also

`markov.test`, `ks.unif.test`, `chisq.unif.test`, `diffsign.test`, `turningpoint.test`, `rank.test`, `lb.test`

Examples

```
#Generate an IID uniform DNA sequence
seq <- simulateMarkovChain(5000, matrix(0.25, 4, 4), states=c("a","c","g","t"))
diid.test(seq)
```

`disambiguate` *Disambiguate a Nucleic Sequence*

Description

Make a DNA/RNA sequence unambiguous by stripping out all symbols that do **not** uniquely specify nucleic acids. In other words, remove all symbols other than `a`'s, `c`'s, `g`'s, `t`'s or `u`'s from the sequence.

Usage

```
## Default S3 method:
disambiguate(x, case=c("lower", "upper", "as is"), ...)
## S3 method for class 'SeqFastadna'
disambiguate(x, ...)
## S3 method for class 'list'
disambiguate(x, ...)
```

Arguments

`x`	A character vector, an object that can be coersed to a character vector or a list of objects that canbe be converted to character vectors. this argument can also be a `SeqFastadna` object provided by the **seqinr** package.
`case`	Determines how symbols in `x` should be treated before translating them into their complements. "`lower`", the default behaviour, converts all symbols to lowercase while "`upper`" converts them to uppercase. "`as is`" allows the symbols to pass unchanged so that the case of each output symbol matches that of the corresponding input symbol.
`...`	Arguments to be passed from or to other functions.

Details

If `x` is a SeqFastadna object or a character vector in which each element is a single nucleobase, then it represents a single sequence. It will be made unambiguous and returned in the same form.

On the other hand, if `x` is a vector of character strings, each of which represents a nucleic sequence, then the result will bea a character vector in which each element contains the unambiguous sequence corresponding to the element in `x` as a character string.

Value

According to the input `x`, a character vector, SeqFastadna object or list containing the completely unambiguous sequence(s) in `x`.

Author(s)

Andrew Hart and Servet Martínez

`estimateMarkovChain` *Fit a first-Order Markov Chain to a Sequence of Finite Symbols*

Description

Estimates the transition matrix and stationary distribution of a first-order Markov chain from an observed sequence of symbols.

Usage

```
estimateMarkovChain(x, circular=TRUE)
```

Arguments

`x`	The sequence of observed symbols as a character vector. purposes
`circular`	Should the sequence be treated as circular for the purpose of estimation? The default is 'TRUE'.

Value

A list with class '`FiniteStateMarkovChain`' having the following components:

`trans.mat`	The stochastic transition matrix estimated from x.
`stat.dist`	The stationary distribution estimated from x.
`states`	the state labels

Author(s)

Andrew Hart and Servet Martínez

See Also

`markov.test`, `markov.disturbance`, `simulateMarkovChain`

Examples

```
#Obtain a random 3 x 3 stochastic matrix with rows and columns labelled "A", "B", "C"
mat <- rstochmat(3, labels=c("A", "B", "C"))
mat

#Simulate a Markov chain of length 500 using mat as the transition matrix
seq <- simulateMarkovChain(500, mat)

#Estimate mat and the stationary distribution for the Markov chain which generated seq
estimateMarkovChain(seq)
```

`ks.unif.test` *Using* `ks.test` *to test for Uniformity on the Unit Interval*

Description

Uses `ks.test` to test that a data vector is uniform on the unit interval. `ks.unif.test(x)` is merely convenient shorthand for `ks.test(x,punif)`.

Usage

```
ks.unif.test(x)
```

Arguments

`x`	a numeric vector or univariate time series.

Value

A list with class "htest" containing the following components:

`statistic`	the value of the test statistic.
`p.value`	the p-value of the test.
`method`	a character string indicating what type of test was performed.
`data.name`	a character string giving the name of the data.

Author(s)

Andrew Hart and Servet Martínez

See Also

`chisq.unif.test`, `markov.test`, `diid.test`

Examples

```
#Generate an IID uniform(0,1) sequence
u <- runif(1000)
ks.unif.test(u)
```

lb.test *The Ljung-Box Test for Uncorrelated Data*

Description

This function is a convenient wrapper for using `Box.test` to perform the Ljung- Box Q test of uncorrelated data without having to specify '`type`'. In other words, `lb.test(x, ...)` is equivalent to `Box.test(x, type="Ljung-Box", ...)`.

Usage

```
lb.test(x, ...)
```

Arguments

`x`	a numeric vector or univariate time series.
`...`	parameters to pass to `Box.test`.

Value

A list with class "htest" containing the following components:

`statistic`	the value of the test statistic.
`parameter`	the degrees of freedom of the approximate chi-squared distribution of the test statistic (taking `fitdf` into account).
`p.value`	the p-value of the test.
`method`	a character string indicating what type of test was performed.
`data.name`	a character string giving the name of the data.

Author(s)

Andrew Hart and Servet Martínez

See Also

`Box.test`, `markov.test`, `diid.test` `diffsign.test`, `turningpoint.test`, `rank.test`

Examples

```
#Generate an IID standard normal sequence
n <- rnorm(1000)
lb.test(n)
```

`markov.disturbance` *Construct feasible Random Noise Generating a First-Order Markov Chain*

Description

Produces a sequence of random noise which would generate an observed sequence of finite symbols provided that the sequence of symbols results from a first-order Markov chain.

Usage

```
markov.disturbance(x, chain = NULL, random = TRUE, bandwidth = 1,
estimates = is.null(chain))
```

Arguments

`x`	A sequence of finite symbols represented as a character vector.
`chain`	A list containing two named components which specify a first-order Markov chain. The '`trans.mat`' component holds the stochastic transition matrix for the chain while the '`stat.dist`' component holds the stationary distribution of the chain. If not specified, '`chain`' is estimated from '`x`' using `estimateMarkovChain`.
`random`	This can be a logical value or a number in the range 0-1. If '`TRUE`', random noise will be generated. If '`FALSE`', the constant value 0.5 will be used as the noise source. If a value in the range 0-1 is specified, that value will be used as a constant noise source. the default value is '`TRUE`'.
`bandwidth`	This value, which should be in the range 0-1, specifies the maximum peak-to-peak bandwidth of the random noise generated. The default value is 1.
`estimates`	A logical value specifying if the Markov chain estimates should be included in the return.

Value

If '`estimates`' is '`TRUE`', returns a list containing the following components:

`disturbance`	the sequence of random noise as a numeric vector.
`trans.mat`	The stochastic transition matrix estimated from x, if '`chain`' is NULL; otherwise a copy of '`trans.mat`' component of '`chain`'.
`stat.dist`	The stationary distribution estimated from x, if '`chain`' is NULL; otherwise a copy of the '`stat.dist`' component of '`chain`'.

Otherwise, if '`estimate`' is '`FALSE`', returns the sequence of random noise as a numeric vector.

Author(s)

Andrew Hart and Servet Martínez

See Also

markov.test, diid.test, diid.disturbance

---

markov.test *A Test for First-Order Markovianness*

Description

Performs a test for first-order Markovianness of a data series by inferring the sequence of i.i.d. U(0,1) random noise that might have generated it.

Usage

```
markov.test(x, type = c("lb", "ks"), method = "holm", lag = 20, ...)
```

Arguments

x	the data series as a vector.
type	the procedures to use to test whether or not the disturbance series is independently and identically distributed on the unit interval. See 'Details'.
method	the correction method to be used for adjusting the p-values. It is identical to the method argument of the p.adjust function, which is called to adjust the p-values.
lag	the number of lags to use when applying the Ljung-Box (portmanteau) test (lb.test).
...	parameters to pass on to functions that can be subsequently called.

Details

This function tests a symbolic sequence for first-order Markovianness (also known as the Markov property). It does this by reverse-engineering the sequence to obtain a sample of the kind of output from a pseudo-random number generator that would have produced the observed sequence if it had been generated by simulating a Markov chain .The sample output is then tested to see if it is an independent and identically distributed siequence of uniform numbers in the range 0-1. this involves the application of at least two tests, one for independence and another for uniformity over the unit interval. One concludes that the sequence is Markovian if the sample output passes the tests (that is, all null hypotheses are accepted) and non-Markovian otherwise.

The test is set up as follows:

H_0: the sequence is first-order Markov
H_1: the sequence is not first-order Markov

To simplify the use of the test, correction for multiple testing is carried out, which yields a single adjusted p- value. If this p-value is less than the significance level established for the test procedure, the null hypothesis of Markovianness is rejected. Otherwise, the null hypothesis should be accepted.

To correctly apply the test, use the type argument to specify at least one test of independence and one test of uniformity from the options displayed in the following table.

Category	Function	Test
Uniformity	ks.unif.test	Kolmogorov-Smirnov test for uniform$(0,1)$ data
	chisq.unif.test	Pearson's chi-squared test for discrete uniform data,
Independence	lb.test	Ljung-Box Q test for uncorrelated data
	diffsign.test	signed difference test of independence
	turningpoint.test	turning point test of independence
	rank.test	rank test of independence

If type is not specified, lb.test and ks.unif.test are used by default.

As this procedure performs multiple tests in order to assess if the sequence has a Markovian dependence structure, it is necessary to adjust the p-values for multiple testing. By default, the Holm-Bonferroni method (holm) is used to correct for multiple testing, but this can be overridden via the method argument. The adjusted p-values are displayed when the result of the test is printed.

The smallest adjusted p-value constitutes the overall p-value for the test. If this p-value is less than the significance level fixed for the test procedure, the null hypothesis of first-order Markovianness is rejected. Otherwise, the null hypothesis should be accepted.

Value

A list with class "multiplehtest" containing the following components:

method	the character string "Composite test for a first-order (finite state) Markov chain".
statistics	the values of the test statistic for all the tests.
parameters	parameters for all the tests. Exactly one parameter is recorded for each test, for example, df for lb.test. Any additional parameters are not saved, for example, the a and b parameters of chisq.unif.test.
p.values	p-values of all the tests.
methods	a vector of character strings indicating what type of tests were performed.
adjusted.p.values	the adjusted p-values.
data.name	a character string giving the name of the data.
adjust.method	indicates which correction method was used to adjust the p-values for multiple testing.
estimate	the transition matrix estimated to fit a first-order Markov chain to the data and used to generate the infered random disturbance.

Note

Sometimes, a warning message advising that ties should not be present for the Kolmogorov-Smirnov test can arise when analysing long sequences. If you do receive this warning, it means that the results of the Kolmogorov-Smirnov test (ks.unif.test) should not be trusted. In this case, Pearson's chi-squared test (chisq.unif.test) should be used instead of the Kolmogorov-Smirnov test.

Author(s)

Andrew Hart and Servet Martínez

References

Hart, A.G. and Martínez, S. (2011) Statistical testing of Chargaff's second parity rule in bacterial genome sequences. *Stoch. Models* **27(2)**, 1–46.

Hart, A.G. and Martínez, S. (2014) Markovianness and Conditional Independence in Annotated Bacterial DNA. *Stat. Appl. Genet. Mol. Biol.* **13(6)**, 693-716. arXiv:1311.4411 [q-bio.QM].

See Also

diid.test, ks.unif.test, chisq.unif.test, diffsign.test, turningpoint.test, rank.test, lb.test

Examples

```
#Generate an IID uniform DNA sequence
seq <- simulateMarkovChain(5000, matrix(0.25, 4, 4), states=c("a","c","g","t"))
markov.test(seq)
```

Nanoarchaeum equitans Kin4-M Chromosome

DNA sequence for the Nanoarchaeum equitans Kin4-M Chromosome

Description

This data set contains the DNA sequence for the chromosome of the *Nanoarchaeum equitans* Kin4-M bacteria. The Accession number for this sequence is NC_005213.1.

Usage

```
nanoarchaeum
```

Format

a SeqFastadna object.

Source

The NCBI ftp server at ftp://ftp.ncbi.nlm.nih.gov in the /genomes/bacteria directory.

See Also

pieris

Examples

```
data(nanoarchaeum)
pair.counts(nanoarchaeum)
```

oligoProfile *Oligo Profiles and Oligo Profile Correlation Plots of Nucleotide Sequences*

Description

Construct a k-mer oligo profile of a nucleotide sequence and print such a profile or its reverse complement. There is also a plot function for producing plots of the profile or its reverse complement and for comparing primary and complementary strand profiles.

Usage

```
oligoProfile(x, k, content=c("dna", "rna"),
case=c("lower", "upper", "as is"), circular=TRUE, disambiguate=TRUE,
plot=TRUE, ...)
## S3 method for class 'OligoProfile'
plot(x, which=1L, units=c("percentage", "count", "proportion"),
 main=NULL, xlab=NULL, ylab=NULL, ...)
## S3 method for class 'OligoProfile'
print(x, which=1L, units=c("percentage", "count", "proportion"),
digits=switch(units, percentage=3L, count=NULL, proportion=3L), ...)
```

Arguments

x	a character vector or an object that can be coersed to a character vector.
k	the k-mer profile to produce.
content	The content type ("dna" or "rna") of the input sequence. oligoProfile can often detect this automatically based on the presence/absence of t's or u's, but if neither is present, the content argument is consulted. The default value is "dna".
case	determines how labels for the array should be generated: in lowercase, in uppercase or left as is, in which case labels such as "b" and "B" will be seen as distinct symbols and counted separately.
circular	Determines if the vector should be treated as circular or not. The default is TRUE, meaning that the start and end of the sequence will be joined together for the purpose of counting.
disambiguate	if set to the default of true, makes the input sequence unambiguous before generating the profile. Otherwise, ambiguous symbols are treated like any other symbols and k-mer counts including them will be computed.
plot	should a plot of the profile be produced? The default is TRUE.
which	For print, specifies whether to display the profile for the sequence used to generate the OligoProfile object (1) or the profile of its reverse complement (2).

	For the plot method, which determines what should be plotted. Values 1 and 2 cause the profile for the original sequence (primary strand) or its reverse complement (complementary strand) to be plotted, respectively. Specifying which=3 will plot a comparison of the two profiles which can be used to assess compliance with Chargaff's second parity rule. the which argument may also be specified when calling oligoProfile, in which case it will be passed on to the plot method if the plot argument is set to TRUE.
units	The oligo profiles can be scaled according to three different units for presentation on plots: "percentage", "count" or "proportion". The default is "percentage".
main	The title of the plot. See plot.default. If not specified, an appropriate title is automatically generated.
xlab	a label for the x-axis of the plot. See plot.default. If not specified, an appropriate label is automatically generated.
ylab	a label for the y-axis of the plot. See plot.default. If not specified, an appropriate label is automatically generated.
digits	The number of significant digits to print. The default is 0L when units is set to "count" and 3L otherwise.
...	arguments to be passed from or to other functions

Details

This function returns the oligo profile for a sequence in an OligoProfile object, which is printed on screen if the plot parameter is FALSE. An oligo profile is simply the counts of all k-mers in a sequence for some specified value of k.

By default, oligoProfile produces a plot of the oligo profile expressed in terms of percentages. The plot argument determines if the plot should be generated or not and plotting parameters such as main, sub, etc., may be passed as arguments to the function when plot is TRUE.

The plot method, either called directly or indirectly via the oligoProfile function, can produce either the oligo profile of x (which = 1), the oligo profile of its reverse complement (which = 2), or an interstrand k-mer correlation plot comparing the k-oligo profile ofx with that of its reverse complement (which = 3). Such

Correlation plots effectively show the relationship between k-mers on the primary and complementary strands in a DNA duplex and can be used to assess compliance with CSPR. More precisely, one would conclude that a genomic sequence complies with CSPR if all the plotted points lie on a diagonal line running from the bottom-left corner to the top-right corner of the graph.

Value

A list with class "OligoProfile" containing the following components:

name	a name to identify the source of the profile.
wordLength	the value of k used to derive the k-mer profile.
content	indicates if the profile pertains to a DNA or RNA sequence.
case	indicates how the case of letters was processed before producing the profile.

circular	indicates whether or not the sequence was considered circular for the purpose of producing the profile.
disambiguate	indicates if the sequence was made unambiguous before producing the profile.
profile	a vector containing the raw counts (frequencies) of all k-mers.

Author(s)

Andrew Hart and Servet Martínez

References

Albrecht-Buehler, G. (2006) Asymptotically increasing compliance of genomes with Chargaff's second parity rules through inversions and inverted transpositions. *PNAS* **103(47)**, 17828–17833.

See Also

pair.counts, triple.counts, quadruple.counts, cylinder.counts, array2vector, table2vector, disambiguate

Examples

```
data(nanoarchaeum)
#Get the 3-oligo profile of Nanoarchaeum without plotting it
nano.prof <- oligoProfile(nanoarchaeum, 3, plot=FALSE)
nano.prof #print oligo profile as percentages
print(nano.prof, units="count") #print oligo profile as counts
plot(nano.prof) #oligo profile plotted as percentages
plot(nano.prof, units="count") #plot it as counts

#plot the 2-oligo profile of Nanoarchaeum as proportions
oligoProfile(nanoarchaeum, k=3, units="proportion")
```

---

pair.counts *Count Pairs in Character Vectors*

Description

Count pairs of adjacent symbols/elements in a character vector.

Usage

```
pair.counts(x, case=c("lower", "upper", "as is"), circular=TRUE)
```

Arguments

x	a character vector or an object that can be coersed to a character vector.
case	determines how labels for the array should be generated: in 'lower' case, in ' upper' case or 'as is', in which case labels such as 'b' and 'B' will be considered as distinct elements and counted separately.
circular	Determines if the vector should be treated as circular or not. The default is TRUE, meaning that the start and end of the sequence will be joined together for the purpose of counting.

Details

When circular is TRUE, the vector is treated as circular so that the some of all the counts in the resulting matrix is equal to the length of the vector and the row and column sums are equivalent. When circular is FALSE, the sum of all the entries in the counts matrix will be one less than the length of the vector and there will be a discrepancy between the row and column sums.

Value

A matrix of counts. The row and column labels correspond to the first and second element of each pair, respectively.

Author(s)

Andrew Hart and Servet Martínez

See Also

triple.counts, quadruple.counts, cylinder.counts, array2vector, table2vector

---

Pieris Rapae Granulovirus Genome

DNA sequence for the Pieris Rapae Granulovirus Genome

Description

This data set contains the DNA sequence for the *Pieris rapae* granulovirus genome. The Accession number for this sequence is NC_013797.1.

Usage

```
pieris
```

Format

a SeqFastadna object.

Source

The NCBI ftp server at ftp://ftp.ncbi.nlm.nih.gov in the /genomes/viruses directory.

See Also

nanoarchaeum

Examples

```
data(pieris)
pair.counts(pieris)
```

---

quadruple.counts *Count Quadruplets in Character Vectors*

Description

Count 4-tuples of adjacent symbols/elements in a character vector.

Usage

```
quadruple.counts(x, case=c("lower", "upper", "as is"), circular=TRUE)
```

Arguments

x	a character vector or an object that can be coersed to a character vector.
case	determines how labels for the array should be generated: in 'lower' case, in ' upper' case or 'as is', in which case labels such as 'b' and 'B' will be counted as distinct elements and counted separately.
circular	Determines if the vector should be treated as circular or not. The default is TRUE, meaning that the start and end of the sequence will be joined together for the purpose of counting.

Details

If circular is TRUE, the vector is treated as circular so that the some of all the counts in the resulting array is equal to the length of the vector and the sums across all dimentions of the array are equivalent, that is: if we write
`q <- quadruple.counts(x)`
for some character sequence x, then
`apply(q,1,sum)`, `apply(q,2,sum)`, `apply(q,3,sum)` and `apply(q,4,sum)`
are all identical.

On the other hand, if circular is FALSE, the sum of all the entries in the counts array will be two less than the length of the vector and there will be a discrepancy between the sums over the various dimensions.

Value

A 4-dimensional array of counts. The labels of the i-th dimension correspond to the i-th item in each tuple, where i is either 1, 2, 3 or 4.

Author(s)

Andrew Hart and Servet Martínez

See Also

`pair.counts`, `triple.counts`, `cylinder.counts`, `array2vector`, `table2vector`

`rank.test` *The Rank Test of Statistical Independence*

Description

Test for a trend in a data series by comparing the number of increasing pairs in the series with the number expected in an i.i.d. series.

Usage

```
rank.test(x)
```

Arguments

`x`	a numeric vector or univariate time series.

Details

Perform a test for trend based on the number of increasing ordered pairs in a data series. Consider pairs of the form (x(i), x(j)), where i<j. An increasing pair is any such pair for which x_i<x_j. This function counts the number of increasing pairs in the data, standardises it to have mean 0 and variance 1 and asymptotically tests it against a standard normal distribution. the test statistic is:

R = (pairs-mu)/sigma, where
pairs is the number of increasing pairs in the data,
mu = n*(n-1)/4,
sigma = sqrt(n*(n-1)*(2*n+5)/72) and
n is the number of data points in the series.

The test is set up as follows:

H_0: the data series is i.i.d. (not trending)
H_1: the data series is not i.i.d. (trending)

Value

A list with class "htest" containing the following components:

`statistic`	the value of the test statistic.
`p.value`	the p-value of the test.
`method`	a character string indicating what type of test was performed.
`data.name`	a character string giving the name of the data.
`pairs`	the number of increasing pairs counted in the data series.
`n`	the number of points in the data series.
`mu`	The expected number of increasing pairs that would be seen in an i.i.d. series.
`sigma`	The standard deviation of the number of increasing pairs that would be seen in an i.i.d. series.

Warning

If the spgs shared object was successfully compiled with support for a 64-bit unsigned integer type, then the following line should yield the value 0:

```
rank.test(1:92683)$pairs-2^32-55607
```

if not, then the package is only using 32-bit integer arithmetic for computing the rank test statistic and this will restrict `rank.test` to analysing series whose length is at most 92682. In this case, attempting to apply `rank.test` to a series longer than 92682 will result in a warning about an integer overflow having occurred and the results of the test should **not** be trusted.

Note

Missing values are not handled.

Points followed by a point having the exact same value are removed from the data series before computing the test statistic.

This test is useful for detecting linear trends in data series.

Author(s)

Andrew Hart and Servet Martínez

References

Brockwell, Peter J., Davis, Richard A. (2002) *Introduction to Time Series and Forecasting*. Springer Texts in Statistics, Springer-Verlag, New York.

See Also

`diffsign.test`, `turningpoint.test`, `lb.test`, `markov.test`, `diid.test`

Examples

```
#Generate an IID standard normal sequence
n <- rnorm(1000)
rank.test(n)
```

`rcspr2mat` *Random Stochastic Matrices Complying with Chargaff's Second Parity Rule for Dinucleotides*

Description

Randomly generate a 4 X 4 stochastic matrix that satisfies Chargaff's second parity rule for dinucleotides.

Usage

```
rcspr2mat(labels=c("a", "c", "g", "t"))
```

Arguments

`labels`	a vector of labels for the rows and columns of the matrix. By default, this is set to the set of four nucleotides a, c, g and t.

Details

This function randomly generates Stochastic matrices of the form

$$\begin{pmatrix} b_1 & b_2 & b_3 & 1-(b_1+b_2+b_3) \\ zb_6 & b_4 & 1-(zb_6+b_4+zb_3) & zb_3 \\ zb_5 & 1-(zb_5+b_4+zb_2) & b_4 & zb_2 \\ 1-(b_5+b_6+b_1) & b_5 & b_6 & b_1 \end{pmatrix}$$

where $b_1, \ldots, b_6$ are values in the interval (0,1) and z is a positive number.

Such matrices characterize sequences of DNA that comply with Chargaff's second parity rule for dinucleotides. See the reference for further information.

Value

A 4 X 4 stochastic matrix satisfying Chargaff's second parity rule. The rows and columns are labelled according to labels.

Note

This function is only intended for obtaining samples of matrices complying with CSPR. It doe snot sample uniformly from the set of all such matrices and hence is not appropriate for simulation experiments requiring uniformly drawn samples.

Author(s)

Andrew Hart and Servet Martínez

References

Hart, A.G. and Martínez, S. (2011) Statistical testing of Chargaff's second parity rule in bacterial genome sequences. *Stoch. Models* **27(2)**, 1–46.

See Also

rstochmat

---

reverseComplement *Reverse Complement of a DNA/RNA Sequence*

Description

Compute the reverse complement of a DNA or RNA sequence.

Usage

```
## Default S3 method:
reverseComplement(x, content=c("dna", "rna"), case=c("lower", "upper", "as is"), ...)
## S3 method for class 'SeqFastadna'
reverseComplement(x, ...)
## S3 method for class 'list'
reverseComplement(x, ...)
```

Arguments

x	A character vector, an object that can be coersed to a character vector or a list of objects that canbe be converted to character vectors. this argument can also be a SeqFastadna object provided by the **seqinr** package.
content	The content type of sequence(s). At present, supported types include "dna" and "rna". the default type is "dna".
case	Determines how symbols in x should be treated before translating them into their complements. "lower", the default behaviour, converts all symbols to lowercase while "upper" converts them to uppercase. "as is" allows the symbols to pass unchanged so that the case of each output symbol matches that of the corresponding input symbol.
...	Arguments to be passed from or to other functions.

Details

If x is a SeqFastadna object or a character vector in which each element is a single nucleobase, then it represents a single sequence and its reverse- complementary sequence will be returned in the same form.

On the other hand, if x is a vector of character strings, each of which represents a nucleic sequence, then the result will bea a character vector in which each element contains the reverse complement of the corresponding element in x as a character string.

Value

According to the input x, a character vector, SeqFastadna object or list containing the reverse complement(s) of the sequence(s) in x.

Author(s)

Andrew Hart and Servet Martínez

See Also

complement

Examples

```
reverseComplement("actg")
reverseComplement(c("t", "g", "a"))

#List of sequences
some.dna <- list("atgcgtcgttaa", c("g", "t", "g", "a", "a", "a"))
reverseComplement(some.dna)

#RNA sequence example
reverseComplement(c("a", "u", "g"), content="rna")

#Examples of lowercase, uppercase and as-is conversion
mixed.case <- c("t", "G", "g", "C", "a")
reverseComplement(mixed.case)
reverseComplement(mixed.case, case="upper")
reverseComplement(mixed.case, case="as is")
```

---

rstochmat *Random Generation of Stochastic Matrices*

Description

Randomly generates stochastic matrices.

Usage

```
rstochmat(n, labels)
```

Arguments

n	the dimension of the matrix. If n is not specified, it is inferred from the lenth of 'labels'.
labels	a vector of labels for the rows and columns of the matrix. If 'labels' is not specified, 'n' must be specified and the value as.character(1:n) is assumed.

Details

Stochastic matrices are non-negative matrices whose rows all sum to unity. This function uniformly generates samples from the set of $n \times n$ stochastic matrices.

At least one of the arguments must be specified. The missing argument is infered from the other.

Value

An $n \times n$ stochastic matrix with rows and columns labelled according to 'labels'.

Author(s)

Andrew Hart and Servet Martínez

See Also

rcspr2mat, estimateMarkovChain, simulateMarkovChain

Examples

```
rstochmat(4)
rstochmat(3, c("a", "b", "c"))
rstochmat(labels=c("r", "R"))
```

---

rstochvec *Random Generation of Stochastic (Probability) Vectors*

Description

Randomly generate probability vectors, that is, non-negative vectors whose elements sum to unity.

Usage

```
rstochvec(n, labels)
```

Arguments

n	the length of the vector. If n is not specified, it is inferred from the lenth of 'labels'.
labels	a vector of labels for the elements of the vector. If 'labels' is not specified, n must be specified and the value as.character(1:n) is assumed.

Details

Stochastic (or probability) vectors are non-negative vectors that sum to unity. This function uniformly generates samples from the set of probability vector sof length n.

At least one of the arguments must be specified. The missing argument is infered from the other.

Value

A probability vector of length n with elements named according to '`labels`'.

Author(s)

Andrew Hart and Servet Martínez

Examples

```
rstochvec(4)
rstochvec(3, c("a", "b", "c"))
rstochvec(labels=c("r", "R"))
```

`simulateMarkovChain` *Simulate a first-Order Markov Chain*

Description

Simulates a first-order Markov chain.

Usage

```
simulateMarkovChain(n, trans.mat, init.dist=NULL, states=colnames(trans.mat))
```

Arguments

`n`	the length of the sample path to simulate.
`trans.mat`	The transition matrix of the Markov chain to simulate.
`init.dist`	The initial distribution to use for starting the simulation. If it is not specified, the stationary distribution of the Markov chain will be computed from trans.mat and used to start the simulation in the steady state.
`states`	This argument can be used to override the labels on the transition matrix (if any) and name the states output on the sample path.

Details

'trans.mat' must be a stochastic matrix. It must either have both row and column names, in which case they must agree, or no row and column names at all. The row/column names will be used to label the states visited by the Markov chain in the sample path simulated. If 'states' is specified, it will be used to label the states of the Markov chain instead of the row/column names of 'trans.mat', in which the length of 'states' must agree with the dimension of 'trans.mat'. If 'trans.mat' has no row/column names and 'states' is not specified, then the states of the Markov chain will be labelled $1, \ldots, n$, where n is the dimension of 'trans.mat'.

Value

A vector of length n containing a realisation of the specified Markov chain.

Author(s)

Andrew Hart and Servet Martínez

See Also

estimateMarkovChain, rstochmat, rcspr2mat

Examples

```
simulateMarkovChain(50, matrix(c(.8, .2, .2, .8), ncol=2))
simulateMarkovChain(50, rstochmat(3), states=c("yes", "no", "maybe"))
```

---

triple.counts *Count Triplets in Character Vectors*

Description

Count triples of adjacent symbols/elements in a character vector.

Usage

```
triple.counts(x, case=c("lower", "upper", "as is"), circular=TRUE)
```

Arguments

x	a character vector or an object that can be coersed to a character vector.
case	determines how labels for the array should be generated: in 'lower' case, in ' upper' case or 'as is', in which case labels such as 'b' and 'B' will be counted as distinct elements and counted separately.
circular	Determines if the vector should be treated as circular or not. The default is TRUE, meaning that the start and end of the sequence will be joined together for the purpose of counting.

Details

If `circular` is `TRUE`, the vector is treated as circular so that the some of all the counts in the resulting array is equal to the length of the vector and the sums across all dimentions of the array are equivalent, that is: if we write
`t <- triple.counts(x)`
for some character sequence x, then
`apply(t,1,sum)`, `apply(t,2,sum)` and `apply(t,3,sum)`
are all identical.

On the other hand, if `circular` is `FALSE`, the sum of all the entries in the counts array will be two less than the length of the vector and there will be a discrepancy between the sums over the various dimensions.

Value

A 3-dimensional array of counts. The labels of the i-th dimension correspond to the i-th element of each triple, where i is either 1, 2 or 3.

Author(s)

Andrew Hart and Servet Martínez

See Also

`pair.counts`, `quadruple.counts`, `cylinder.counts`, `array2vector`, `table2vector`

`turningpoint.test` *The Turning Point Test of Statistical Independence*

Description

Perform a test of statistical independence of a data series by comparing the number of turning points present in the series with the number of turning points expected to be present in an i.i.d. series.

Usage

```
turningpoint.test(x)
```

Arguments

`x`	a numeric vector or univariate time series.

Details

If the data is x[1], x[2], ..., x[n], then there is a turning point at the point i if either x[i-1]<x[i] and x[i]>x[i+1], or x[i-1]>x[i] and x[i]<x[i+1]. this function counts the number of turning points in the data, standardises it to have mean 0 and variance 1 and asymptotically tests it against a standard normal distribution. The test statistic is

T = (tp-mu)/sigma, where
tp is the number of turning points present in the series,
mu = 2*(n-2)/3,
sigma = sqrt((16*n-29)/90) and
n is the number of data points in the series.

The test is set up as follows:

H_0: the data series is i.i.d. (not trending)
H_1: the data series is not i.i.d. (trending)

Value

A list with class "htest" containing the following components:

`statistic`	the value of the test statistic.
`p.value`	the p-value of the test.
`method`	a character string indicating what type of test was performed.
`data.name`	a character string giving the name of the data.
`n`	the number of points in the data series.
`mu`	The expected number of turning points that would be seen in an i.i.d. series.
`sigma`	The standard deviation of the number of turning points that would be seen in an i.i.d. series.

Note

Missing values are not handled.

Points followed by a point having the exact same value are removed from the data series before computing the test statistic.

This test is useful for detecting cyclic/periodic trends in data series.

Author(s)

Andrew Hart and Servet Martínez

References

Brockwell, Peter J., Davis, Richard A. (2002) *Introduction to Time Series and Forecasting*. Springer Texts in Statistics, Springer-Verlag, New York.

Bienaymé, Irénée-Jules (1874). Sur une question de probabilités. *Bull. Math. Soc. Fr.* **2**, 153-154.

See Also

diffsign.test, rank.test, lb.test, markov.test, diid.test

Examples

```
#Generate an IID standard normal sequence
n <- rnorm(1000)
turningpoint.test(n)
```

Index