Package ‘motifcounter’

May 18, 2024

Type  Package
Title  R package for analysing TFBSs in DNA sequences
Version  1.28.0
Date  2017
Author  Wolfgang Kopp [aut, cre]
Suggests  knitr, rmarkdown, testthat, MotifDb, seqLogo, prettydoc
Imports  Biostrings, methods
Depends  R(>= 3.0)
Maintainer  Wolfgang Kopp <wolfgang.kopp@mdc-berlin.de>
Description  ‘motifcounter’ provides motif matching, motif counting
and motif enrichment functionality based on position
frequency matrices.
The main features of the packages include the utilization
of higher-order background models and accounting
for self-overlapping motif matches when determining motif enrichment.
The background model allows to capture dinucleotide
(or higher-order nucleotide) composition adequately
which may reduced model biases and misleading results compared
to using simple GC background models.
When conducting a motif enrichment analysis
based on the motif match count, the package
relies on a compound Poisson distribution or alternatively
a combinatorial model. These distribution account for self-overlapping
motif structures as exemplified by repeat-like or palindromic motifs,
and allow to determine the p-value and fold-enrichment for
a set of observed motif matches.
License  GPL-2
biocViews  Transcription,MotifAnnotation,SequenceMatching,Software
RoxygenNote  6.0.1
VignetteBuilder  knitr
NeedsCompilation  yes
Collate 'background_wrapper.R' 'compoiss_wrapper.R'
    'combinatorial_wrapper.R' 'score_wrapper.R' 'count_wrapper.R'
    'enrichmentTest.R' 'foreground_wrapper.R' 'markovmodel.R'
    'motifcounter-package.R' 'observed_wrapper.R' 'option.R'
    'overlap.R' 'simulate_wrapper.R' 'wrapper.R'

git_url https://git.bioconductor.org/packages/motifcounter

git_branch RELEASE_3_19

Repository Bioconductor 3.19

Date/Publication 2024-05-17

Contents

motifcounter-package ........................................ 3
Background-class ............................................. 5
clumpSizeDist ................................................ 5
combinatorialDist ............................................ 7
compoundPoissonDist ......................................... 8
computeClumpStartProb ....................................... 9
generateDNAString ........................................... 11
generateDNAStringSet ........................................ 12
getAlpha ....................................................... 13
getBeta ......................................................... 13
getBeta3p ....................................................... 14
getBeta5p ....................................................... 14
getCounts ....................................................... 15
getGamma ....................................................... 15
getOrder ....................................................... 16
getSinglestranded ............................................ 16
getStation ...................................................... 17
getTrans ....................................................... 17
hitStrand ...................................................... 18
lenSequences .................................................. 19
markovModel ................................................... 19
motifAndBackgroundValid ..................................... 20
motifcounterOptions .......................................... 20
motifEnrichment ............................................... 22
motifHitProfile ............................................... 23
motifHits ...................................................... 24
motifValid ..................................................... 25
normalizeMotif ............................................... 26
numMotifHits .................................................. 27
Overlap-class ................................................ 28
probOverlapHit ............................................... 28
readBackground ............................................... 29
motifcounter-package

Description

The package provides functions for determining the positions of motif hits as well as motif hit enrichment for a given position frequency matrix (PFM) in a DNA sequence of interest. The following examples guides you through the main functions of the ‘motifcounter’ package.

Details

For an analysis with ‘motifcounter’, the user is required to provide 1) a PFM, 2) a DNA sequence which is used to estimate a background model (see link{readBackground}), 3) a DNA sequence of interest that shall be scanned for motif hits (can be the same as the one used for point 2), and 4) (optionally) a desired false positive probability of motif hits in random DNA sequences (see motifcounterOptions).

Package: motifcounter
Type: Package
Version: 1.0
Date: 2016-11-04
License: GPL-2

Author(s)

Wolfgang Kopp

Maintainer: Wolfgang Kopp <kopp@molgen.mpg.de>
Examples

# Load sequences
file = system.file("extdata", "seq.fasta", package = "motifcounter")
seqs = Biostrings::readDNAStringSet(file)

# Estimate an order-1 background model
order = 1
bg = readBackground(seqs, order)

# Load motif
motiffile = system.file("extdata", "x31.tab", package = "motifcounter")
motif = t(as.matrix(read.table(motiffile)))

# Normalize the motif
# Normalization is sometimes necessary to prevent zeros in
# the motif
motif = normalizeMotif(motif)

# Use subset of the sequences
seqs = seqs[1:10]

# Optionally, set the false positive probability
# alpha=0.001 # is also the default
# motifcounterOptions(alpha)

# Investigate the per-position and per-strand scores in a given sequence
scores = scoreSequence(seqs[[1]], motif, bg)

# Investigate the per-position and per-strand motif hits in a given sequence
hits = motifHits(seqs[[1]], motif, bg)

# Determine the average score profile across a set of sequences
scores = scoreProfile(seqs, motif, bg)

# Determine the average motif hit profile across a set of sequences
hits = motifHitProfile(seqs, motif, bg)

# Determine the empirical score distribution
scoreHistogram(seqs, motif, bg)

# Determine the theoretical score distribution in random sequences
scoreDist(motif, bg)

# Determine the motif hit enrichment in a set of DNA sequences
# 1. Use the compound Poisson approximation
# and scan only a single strand for motif hits
result = motifEnrichment(seqs, motif, bg,
                          singlestranded = TRUE, method = "compound")

# Determine the motif hit enrichment in a set of DNA sequences
# 2. Use the compound Poisson approximation
Background-class

Background class definition

Description

Objects of this class serve as a container that holds parameters for the Background model.

Details

A Background model is constructed via readBackground.

Slots

station  Stationary probabilities
trans   Transition probabilities
counts k-mer counts
order   Background model order

clumpSizeDist  Clump size distribution

Description

This function approximates the distribution of the clump sizes.

Usage

clampSizeDist(maxclump, overlap, method = "kopp")

Arguments

maxclump  Maximal clump size
overlap   An Overlap object.
method    String that defines which method shall be invoked: ‘pape’ or ‘kopp’ (see description). Default: method = ‘kopp’.
Details

The clump size distribution can be determined in two alternative ways:

1. A re-implemented version of the algorithm that was described in Pape et al. *Compound poisson approximation of the number of occurrences of a position frequency matrix (PFM) on both strands. 2008* can be invoked using `method='pape'`.

2. An improved approximation of the clump size distribution uses more appropriate statistical assumptions concerning overlapping motif hits and that can be used with order-d background models as well. The improved version is used by default with `method='kopp'`.

Value

List containing

- `dist` Distribution of the clump size

See Also

- `probOverlapHit`

Examples

```r
# Load sequences
seqfile = system.file("extdata", "seq.fasta", package = "motifcounter")
seqs = Biostrings::readDNAStringSet(seqfile)

# Load motif
motiffile = system.file("extdata", "x31.tab", package = "motifcounter")
motif = t(as.matrix(read.table(motiffile)))

# Load background model
bg = readBackground(seqs, 1)

# Use 100 individual sequences of length 150 bp each
seqlen = rep(150, 100)

# Compute overlapping probabilities
# for scanning the forward DNA strand only
op = motifcounter:::probOverlapHit(motif, bg, singlestranded = FALSE)

# Computes the compound Poisson distribution
dist = motifcounter:::clumpSizeDist(20, op)
```
combinatorialDist

Combinatorial model approximation of the number of motif hits

Description
This function approximates the distribution of the number of motif hits. To this end, it sums over all combinations of obtaining k hits in a random sequence of a given length using an efficient dynamic programming algorithm.

Usage
combinatorialDist(seqlen, overlap)

Arguments
- **seqlen**: Integer-valued vector that defines the lengths of the individual sequences. For a given DNAStringSet, this information can be retrieved using `numMotifHits`.
- **overlap**: An Overlap object.

Details
This function is an alternative to `compoundPoissonDist` which requires fixed-length sequences and currently only supports the computation of the distribution of the number of hits when both DNA strands are scanned for motif hits.

Value
List containing
- **dist**: Distribution of the number of hits

See Also
- `compoundPoissonDist`
- `numMotifHits`
- `probOverlapHit`

Examples
```r
# Load sequences
seqfile = system.file("extdata", "seq.fasta", package = "motifcounter")
seqs = Biostrings::readDNAStringSet(seqfile)

# Load motif
motiffile = system.file("extdata", "x31.tab", package = "motifcounter")
motif = t(as.matrix(read.table(motiffile)))

# Load background model
```
compoundPoissonDist

Description
This function approximates the distribution of the number of motif hits that emerges from a random DNA sequence of a given length.

Usage
compoundPoissonDist(seqlen, overlap, method = "kopp")

Arguments

  seqlen    Integer-valued vector that defines the lengths of the individual sequences. For a given DNAStringSet, this information can be retrieved using numMotifHits.

  overlap   An Overlap object.

  method    String that defines which method shall be invoked: 'pape' or 'kopp' (see description). Default: method = 'kopp'.

Details
The distribution can be determined in two alternative ways:

1. A re-implemented version of the algorithm that was described in Pape et al. Compound poisson approximation of the number of occurrences of a position frequency matrix (PFM) on both strands. 2008 can be invoked using method='pape'. The main purpose of this implementation concerns benchmarking an improved approximation. In contrast to the original model, this implementation can be used with general order-d Markov models.

2. We provide an improved compound Poisson approximation that uses more appropriate statistical assumptions concerning overlapping motif hits and that can be used with order-d background models as well. The improved version is used by default with method='kopp'. Note: Only method='kopp' supports the computation of the distribution of the number of motif hits w.r.t. scanning a single DNA strand (see probOverlapHit).
computeClumpStartProb

Value

List containing

\textbf{dist} Distribution of the number of hits

See Also

combinatorialDist
probOverlapHit
numMotifHits

Examples

\begin{verbatim}
# Load sequences
seqfile = system.file("extdata", "seq.fasta", package = "motifcounter")
seqs = Biostrings::readDNAStringSet(seqfile)

# Load motif
motiffile = system.file("extdata", "x31.tab", package = "motifcounter")
motif = t(as.matrix(read.table(motiffile)))

# Load background model
bg = readBackground(seqs, 1)

# Use 100 individual sequences of length 150 bp each
seqlen = rep(150, 100)

# Compute overlapping probabilities
# for scanning the forward DNA strand only
op = motifcounter:::probOverlapHit(motif, bg, singlestranded = TRUE)

# Computes the compound Poisson distribution
dist = motifcounter:::compoundPoissonDist(seqlen, op)
#plot(1:length(dist$dist)-1, dist$dist)

# Compute overlapping probabilities
# for scanning the forward DNA strand only
op = motifcounter:::probOverlapHit(motif, bg, singlestranded = FALSE)

# Computes the compound Poisson distribution
dist = motifcounter:::compoundPoissonDist(seqlen, op)
#plot(1:length(dist$dist)-1, dist$dist)
\end{verbatim}
Description

This function leverages a Markov model in order to determine the clump start probability. The computation depends on the selected false positive probability for calling motif matches 'alpha' and the pre-determined overlapping match probabilities 'beta'.

Usage

computeClumpStartProb(overlap)

Arguments

overlap An Overlap object.

Details

The general idea of the method relies on the fact that for the stationary distribution of the Markov model, motif matches must be observed with probability 'alpha'. Hence, the clump start probability 'tau' is optimized to achieve that goal.

The R interface is only used for the purpose of testing the correctness of the model.

Value

Clump start probability 'tau'

See Also

compoundPoissonDist
numMotifHits
probOverlapHit

Examples

# Load sequences
seqfile = system.file("extdata", "seq.fasta", package = "motifcounter")
seqs = Biostrings::readDNAStringSet(seqfile)

# Load motif
motiffile = system.file("extdata", "x31.tab", package = "motifcounter")
motif = t(as.matrix(read.table(motiffile)))

# Load background model
bg = readBackground(seqs, 1)

# Compute overlap probabilities
op = motifcounter:::probOverlapHit(motif, bg, singlestranded = FALSE)

# Computes the clump start probability
dist = motifcounter:::computeClumpStartProb(op)
generateDNAString

Description

This function generates a random DNAString of a given length by sampling from the background model.

Usage

generateDNAString(len, bg)

Arguments

len Integer length of the sequence
bg A Background object

Value

A DNAString object

See Also

generateDNAStringSet

Examples

# Load sequences
seqfile = system.file("extdata", "seq.fasta", package = "motifcounter")
seqs = Biostrings::readDNAStringSet(seqfile)

# Load background
bg = readBackground(seqs, 1)

# Generate a 1 kb random sequence
motifcounter:::generateDNAString(1000, bg)
generateDNAStringSet  Generate DNAStringSet

Description

This function generates a DNAStringSet-object of the given individual sequence lengths by sampling from the background model.

Usage

generateDNAStringSet(seqlen, bg)

Arguments

seqlen  Integer-valued vector that defines the lengths of the individual sequences. For a given DNAStringSet, this information can be retrieved using numMotifHits.

bg  A Background object

Value

A DNAStringSet object

See Also

generateDNAStringSet

Examples

# Load sequences
seqfile = system.file("extdata", "seq.fasta", package = "motifcounter")
seqs = Biostrings::readDNAStringSet(seqfile)

# Load background
bg = readBackground(seqs, 1)

# Generate random sequences of various lengths
motifcounter:::generateDNAStringSet(10:50, bg)
getAlpha

Accessor to slot alpha

Description

Accessor to slot alpha

Usage

getAlpha(obj)

Arguments

obj An Overlap object

Value

alpha slot

gBeta

Accessor to slot beta

Description

Accessor to slot beta

Usage

gBeta(obj)

Arguments

obj An Overlap object

Value

beta slot
getBeta3p

### Description

Accessor to slot beta3p

### Usage

getBeta3p(obj)

#### Arguments

- **obj**
  
  An Overlap object

#### Value

beta3p slot

---

getBeta5p

### Description

Accessor to slot beta

### Usage

getBeta5p(obj)

#### Arguments

- **obj**

  An Overlap object

#### Value

beta5p slot
### getCounts

**Description**

Accessor to slot counts

**Usage**

getCounts(obj)

**Arguments**

- obj: A Background object

**Value**

counts slot

### getGamma

**Description**

Accessor to slot gamma

**Usage**

getGamma(obj)

**Arguments**

- obj: An Overlap object

**Value**

gamma slot
get0rder

Accessor to slot order

Description
Accessor to slot order

Usage
get0rder(obj)

Arguments
obj A Background object

Value
order slot

getSinglestranded

Accessor to slot singlestranded

Description
Accessor to slot singlestranded

Usage
getSinglestranded(obj)

Arguments
obj An Overlap object

Value
singlestranded slot
getStation

Accessor to slot station

Description
Accessor to slot station

Usage
getStation(obj)

Arguments
obj A Background object

Value
station slot

getTrans

Accessor to slot trans

Description
Accessor to slot trans

Usage
getTrans(obj)

Arguments
obj A Background object

Value
trans slot
hitStrand  

**Description**

This function computes the per-position motif matches in a given DNA strand.

**Usage**

hitStrand(seq, pfm, bg, threshold = NULL)

**Arguments**

- `seq`: A DNAString object
- `pfm`: An R matrix that represents a position frequency matrix
- `bg`: A Background object
- `threshold`: Score threshold for calling motif matches. If NULL, the threshold will determined from alpha.

**Details**

The function returns the per-position scores for the given strand. If the sequence is too short, it contains an empty vector.

**Value**

- `hits`: Vector of motif hits on the given strand

**Examples**

```r
# Load sequences
seqfile = system.file("extdata", "seq.fasta", package = "motifcounter")
seqs = Biostrings::readDNAStringSet(seqfile)

# Load background
bg = readBackground(seqs, 1)

# Load motif
motiffile = system.file("extdata", "x31.tab", package = "motifcounter")
motif = t(as.matrix(read.table(motiffile)))

# Compute the per-position and per-strand scores
motifcounter:::hitStrand(seqs[[1]], motif, bg)
```
**lenSequences**  
*Length of sequences in a given fasta file*

**Description**

The function returns a vector containing the lengths of each sequence contained in a set of sequences. Sequences containing 'N' or 'n' are skipped from the analysis and are set to length zero.

**Usage**

```
lenSequences(seqs)
```

**Arguments**

- `seqs` A DNAStringSet object

**Value**

A vector containing the lengths of each individual sequences

**Examples**

```
# Load sequences
file = system.file("extdata", "seq.fasta", package = "motifcounter")
seqs = Biostrings::readDNAStringSet(file)

# Retrieve sequence lengths
motifcounter:::lenSequences(seqs)
```

---

**markovModel**  
*Markov model for generating Y_1Y_2Y3 ...*

**Description**

This function implements the Markov model for producing motif matches. The function takes a state probability vector and uses the transition probabilities in order to obtain the state probability at the next time point. This function is used used to determine the stationary distribution of the states.

**Usage**

```
markovModel(overlap, nsteps = 1)
```

**Arguments**

- `overlap` An Overlap object.
- `nsteps` Number of state transitions to perform
# motifAndBackgroundValid

**Check validity of PFM with background**

This function checks if the PFM x background combination is valid. The function throws an error if this is not the case.

## Usage

```r
motifAndBackgroundValid(pfm, bg)
```
motifcounterOptions

Set parameters for the enrichment analysis

Description
This function sets some global parameters for the 'motifcounter' package.

Usage

motifcounterOptions(alpha = 0.001, gran = 0.1, ncores = 1)

Arguments

alpha  Numeric False positive probability for calling motif hits by chance. Default: alpha = 0.001
gran  Numeric score granularity which is used for discretizing the score range. Default: gran = 0.1
ncores  Integer number of cores used for parallel processing, if openMP is available. Default: ncores = 1

Arguments

pfm  An R matrix that represents a position frequency matrix
bg  A Background object

Value
None

Examples

# Load sequences
seqfile = system.file("extdata", "seq.fasta", package = "motifcounter")
seqs = Biostrings::readDNAStringSet(seqfile)

# Load background
bg = readBackground(seqs, 1)

# Load motif
motiffile = system.file("extdata", "x1.tab", package = "motifcounter")
motif = t(as.matrix(read.table(motiffile)))

# Check validity
motifcounter:::motifAndBackgroundValid(motif, bg)
Details

alpha=0.001 amounts to calling one motif hit per strand by chance in a sequence of length 1000 bp. Decreasing gran will increase number of discrete bins that represent the real-valued score range. This will yield more accurate score distribution due to less discretization noise, however, it incurs an increase of the computational burden.

Value

None

Examples

# Prescribe motifcounter Options
motifcounterOptions(alpha = 0.001, gran = 0.1, ncores = 1)

motifEnrichment

Enrichment of motif hits

Description

This function determines whether a given motif is enriched in a given DNA sequences.

Usage

motifEnrichment(seqs, pfm, bg, singlestranded = FALSE, method = "compound")

Arguments

seqs A DNASTringSet or DNASTring object
pfm An R matrix that represents a position frequency matrix
bg A Background object
singlestranded Boolean that indicates whether a single strand or both strands shall be scanned for motif hits. Default: singlestranded = FALSE.
method String that defines whether to use the 'compound' Poisson approximation' or the 'combinatorial' model. Default: method='compound'.

Details

Enrichment is tested by comparing the observed number of motif hits against a theoretical distribution of the number of motif hits in random DNA sequences. Optionally, the theoretical distribution of the number of motif hits can be evaluated by either a 'compound Poisson model' or the 'combinatorial model'. Additionally, the enrichment test can be conducted with respect to scanning only the forward strand or both strands of the DNA sequences. The latter option is only available for the 'compound Poisson model'.
motifHitProfile

Value

List that contains

- **pvalue**  P-value for the enrichment test
- **fold**  Fold-enrichment with respect to the expected number of hits

See Also

- compoundPoissonDist, combinatorialDist

Examples

```r
# Load sequences
seqfile = system.file("extdata", "seq.fasta", package = "motifcounter")
seqs = Biostrings::readDNAStringSet(seqfile)

# Load background
bg = readBackground(seqs, 1)

# Load motif
motiffile = system.file("extdata", "x31.tab", package = "motifcounter")
motif = t(as.matrix(read.table(motiffile)))

# 1 ) Motif enrichment test w.r.t. scanning a *single* DNA strand
# based on the 'Compound Poisson model'
result = motifEnrichment(seqs, motif, bg,
                           singlestranded = TRUE, method = "compound")

# 2 ) Motif enrichment test w.r.t. scanning *both* DNA strand
# based on the 'Compound Poisson model'
result = motifEnrichment(seqs, motif, bg, method = "compound")

# 3 ) Motif enrichment test w.r.t. scanning *both* DNA strand
# based on the *combinatorial model*
result = motifEnrichment(seqs, motif, bg, singlestranded = FALSE,
                          method = "combinatorial")
```

description

This function computes the per-position average motif hit profile across a set of fixed-length DNA sequences. It can be used to reveal positional constraints of TFBSs.
motifHitProfile(seqs, pfm, bg)

Arguments

seqs A DNAStringSet or DNAString object
pfm An R matrix that represents a position frequency matrix
bg A Background object

Value

List containing

fscores Per-position average forward strand motif hits
rscores Per-position average reverse strand motif hits

Examples

# Load sequences
seqfile = system.file("extdata", "seq.fasta", package = "motifcounter")
seqs = Biostrings::readDNAStringSet(seqfile)
seqs = seqs[1:10]

# Load background
bg = readBackground(seqs, 1)

# Load motif
motiffile = system.file("extdata", "x31.tab", package = "motifcounter")
motif = t(as.matrix(read.table(motiffile)))

# Compute the motif hit profile
motifHitProfile(seqs, motif, bg)

motifHits  Motif hit observations

Description

This function determines per-position motif hits in a given DNA sequence.

Usage

motifHits(seq, pfm, bg, threshold = NULL)
motifValid

Arguments

- **seq**: A DNASTring object
- **pfm**: An R matrix that represents a position frequency matrix
- **bg**: A Background object
- **threshold**: Score threshold for calling motif matches. If NULL, the threshold will determined from alpha.

Value

List containing

- **fhits**: Per-position motif hits on the forward strand
- **rhits**: Per-position motif hits on the reverse strand

Examples

```r
# Load sequences
seqfile = system.file("extdata", "seq.fasta", package = "motifcounter")
seq = Biostrings::readDNASTringSet(seqfile)

# Load background
bg = readBackground(seq, 1)

# Load motif
motiffile = system.file("extdata", "x31.tab", package = "motifcounter")
motif = t(as.matrix(read.table(motiffile)))

# Determine the motif hits
motifHits(seq[[1]], motif, bg)
```

motifValid

*Check validity of PFM*

Description

This function checks if the PFM is valid. The function throws an error if the R matrix does not represent a PFM.

Usage

```r
motifValid(pfm)
```

Arguments

- **pfm**: An R matrix that represents a position frequency matrix
normalizeMotif

Value

None

Examples

# Load motif
motiffile = system.file("extdata", "x1.tab", package = "motifcounter")
motif = t(as.matrix(read.table(motiffile)))

# Check validity
motifcounter:::motifValid(motif)

normalizeMotif

Normalizes a PFM

Description

This function normalizes a PFM and optionally adds pseudo-evidence to each entry of the matrix.

Usage

normalizeMotif(pfm, pseudo = 0.01)

Arguments

pfm

An R matrix that represents a position frequency matrix

pseudo

Small numeric pseudo-value that is added to each entry in the PFM in order to ensure strictly positive entries. Default: pseudo = 0.01

Value

A normalized PFM

Examples

# Load motif
motiffile = system.file("extdata", "x1.tab", package = "motifcounter")
motif = t(as.matrix(read.table(motiffile)))

# Normalize motif
new_motif = normalizeMotif(motif)
numMotifHits

Number of motif hits in a set of DNA sequences

Description

This function counts the number of motif hits that are found in a given set of DNA sequences.

Usage

numMotifHits(seqs, pfm, bg, singlestranded = FALSE)

Arguments

seqs A DNAStringSet or DNAString object
pfm An R matrix that represents a position frequency matrix
bg A Background object
singlestranded Boolean that indicates whether a single strand or both strands shall be scanned for motif hits. Default: singlestranded = FALSE.

Details

Optionally, it can be used to count motif hits on one or both strands, respectively.

Value

A list containing

nseq Number of individual sequences
lseq Vector of individual sequence lengths
numofhits Vector of the number of hits in each individual sequence

Examples

# Load sequences
seqfile = system.file("extdata", "seq.fasta", package = "motifcounter")
seqs = Biostrings::readDNAStringSet(seqfile)

# Load background
bg = readBackground(seqs, 1)

# Load motif
motiffile = system.file("extdata", "x31.tab", package = "motifcounter")
motif = t(as.matrix(read.table(motiffile)))

# Count motif hits both strands
noc = motifcounter:::numMotifHits(seqs, motif, bg)
noc$numofhits
# Count motif hits on a single strand
noc = motifcounter::numMotifHits(seqs, motif, bg, singlestranded = TRUE)
noc$numofhits

---

**Overlap-class**

**Overlap class definition**

**Description**

Objects of this class serve as a container that holds parameters for the overlapping hit probabilities.

**Details**

An Overlap object is constructed via the `probOverlapHit`

**Slots**

- `alpha` Scalar numeric significance level to call motif matches
- `beta` Numeric vector of principal overlapping hit probabilities on the same strand.
- `beta3p` Numeric vector of principal overlapping hit probabilities with 3’-overlap.
- `beta5p` Numeric vector of principal overlapping hit probabilities with 5’-overlap.
- `gamma` Numeric vector of marginal overlapping hit probabilities.
- `singlestranded` logical flag to indicate whether one or both strands are scanned for motif matches.

---

**probOverlapHit**

**Overlapping motif hit probabilities**

**Description**

This function computes a set of self-overlapping probabilites for a motif and background model.

**Usage**

`probOverlapHit(pfm, bg, singlestranded = FALSE)`

**Arguments**

- `pfm` An R matrix that represents a position frequency matrix
- `bg` A Background object
- `singlestranded` Boolean that indicates whether a single strand or both strands shall be scanned for motif hits. Default: `singlestranded = FALSE`. 
Details

The ‘gamma’s are determined based on two-dimensional score distributions (similar as described in Pape et al. 2008), however, they are computed based on an order-d background model. On the other hand, the ‘beta’s represent overlapping hit probabilities that were corrected for intermediate hits.

Value

An Overlap object

Examples

```r
# Load sequences
seqfile = system.file("extdata", "seq.fasta", package = "motifcounter")
seqs = Biostrings::readDNAStringSet(seqfile)

# Load background
bg = readBackground(seqs, 1)

# Load motif
motiffile = system.file("extdata", "x31.tab", package = "motifcounter")
motif = t(as.matrix(read.table(motiffile)))

# Compute overlapping hit probabilities for scanning both DNA strands
op = motifcounter:::probOverlapHit(motif, bg, singlestranded = FALSE)

# Compute overlapping hit probabilities for scanning a single DNA strand
op = motifcounter:::probOverlapHit(motif, bg, singlestranded = TRUE)
```

readBackground

Estimates a background model from a set of DNA sequences

Description

Given a set of DNA sequences and an order, this function estimates an order-d Markov model which is used to characterize random DNA sequences.

Usage

```r
readBackground(seqs, order = 1)
```

Arguments

- **seqs**: A DNAStringSet object
- **order**: Order of the Markov models that shall be used as the background model. Default: order = 1.
revcompMotif

### Value

A Background object

### Examples

```r
# Load sequences
file = system.file("extdata", "seq.fasta", package = "motifcounter")
seqs = Biostrings::readDNAStringSet(file)

# Estimate an order-1 Markov model
bg = readBackground(seqs, 1)
```

---

**revcompMotif**  
*Reverse complements a PFM*

### Description

This function computes the reverse complement of a given PFM.

### Usage

```r
revcompMotif(pfm)
```

### Arguments

- `pfm`  
  An R matrix that represents a position frequency matrix

### Value

Reverse complemented PFM

### Examples

```r
# Load motif
motiffile = system.file("extdata", "x1.tab", package = "motifcounter")
motif = t(as.matrix(read.table(motiffile)))

# Reverse complement motif
revcompMotif = motifcounter:::revcompMotif(motif)
```
Description

This function computes the score distribution for the given PFM and background. The Score distribution is computed based on an efficient dynamic programming algorithm.

Usage

scoreDist(pfm, bg)

Arguments

pfm  An R matrix that represents a position frequency matrix
bg   A Background object

Value

List that contains

scores  Vector of scores
dist    Score distribution

Examples

# Load sequences
seqfile = system.file("extdata", "seq.fasta", package = "motifcounter")
seqs = Biostrings::readDNAStringSet(seqfile)

# Load background
bg = readBackground(seqs, 1)

# Load motif
motiffile = system.file("extdata", "x31.tab", package = "motifcounter")
motif = t(as.matrix(read.table(motiffile)))

# Compute the score distribution
dp = scoreDist(motif, bg)
**scoreDistBf**

---

**Description**

This function computes the score distribution for a given PFM and a background model.

**Usage**

```r
scoreDistBf(pfm, bg)
```

**Arguments**

- `pfm` An R matrix that represents a position frequency matrix
- `bg` A Background object

**Details**

The result of this function is identical to `scoreDist`, however, the method employs a less efficient algorithm that enumerates all DNA sequences of the length of the motif. This function is only used for debugging and testing purposes and might require substantial computational resources for long motifs.

**Value**

List containing

- `scores` Vector of scores
- `dist` Score distribution

**See Also**

- `scoreDist`

**Examples**

```r
# Load sequences
seqfile = system.file("extdata", "seq.fasta", package = "motifcounter")
seqs = Biostrings::readDNAStringSet(seqfile)

# Load background
bg = readBackground(seqs, 1)

# Load motif
motiffile = system.file("extdata", "x31.tab", package = "motifcounter")
motif = t(as.matrix(read.table(motiffile)))

# Compute the score distribution
dp = motifcounter:::scoreDistBf(motif, bg)
```
**scoreDistEmpirical**  

*Empirical score distribution*

**Description**

This function estimates the empirical score distribution on a set of randomly generated DNA sequences based on the background model. This function is only used for benchmarking analysis.

**Usage**

```r
scoreDistEmpirical(pfm, bg, seqlen, nsim)
```

**Arguments**

- `pfm`: An R matrix that represents a position frequency matrix.
- `bg`: A Background object.
- `seqlen`: Integer-valued vector that defines the lengths of the individual sequences. For a given DNAStringSet, this information can be retrieved using `numMotifHits`.
- `nsim`: Integer number of random samples.

**Value**

List containing

- `scores`: Vector of scores
- `dist`: Score distribution

**See Also**

`scoreDist`

**Examples**

```r
# Load sequences
seqfile = system.file("extdata", "seq.fasta", package = "motifcounter")
seqs = Biostrings::readDNAStringSet(seqfile)

# Load background
bg = readBackground(seqs, 1)

# Load motif
motiffile = system.file("extdata", "x31.tab", package = "motifcounter")
motif = t(as.matrix(read.table(motiffile)))

# Compute the empirical score distribution in sequences of length 1kb using 1000 samples
motifcounter:::scoreDistEmpirical(motif, bg, seqlen = 1000, nsim = 1000)
```
Description

This function computes the empirical score distribution for a given set of DNA sequences.

Usage

scoreHistogram(seqs, pfm, bg)

Arguments

seqs A DNAStringSet or DNAString object

pfm An R matrix that represents a position frequency matrix

bg A Background object

Details

It can be used to compare the empirical score distribution against the theoretical one (see scoreDist).

Value

List containing

scores Vector of scores

dist Score distribution

See Also

scoreDist

Examples

# Load sequences
seqfile = system.file("extdata", "seq.fasta", package = "motifcounter")
seqs = Biostrings::readDNAStringSet(seqfile)

# Load background
bg = readBackground(seqs, 1)

# Load motif
motiffile = system.file("extdata", "x31.tab", package = "motifcounter")
motif = t(as.matrix(read.table(motiffile)))

# Compute the empirical score histogram
scoreHistogram(seqs, motif, bg)
scoreHistogramSingleSeq

Score histogram on a single sequence

Description

This function computes the empirical score distribution by normalizing the observed score histogram for a given sequence.

Usage

scoreHistogramSingleSeq(seq, pfm, bg)

Arguments

- **seq**: A DNAString object
- **pfm**: An R matrix that represents a position frequency matrix
- **bg**: A Background object

Value

List containing

- **scores**: Vector of scores
- **dist**: Score distribution

Examples

```r
# Load sequences
seqfile = system.file("extdata", "seq.fasta", package = "motifcounter")
seqs = Biostrings::readDNAStringSet(seqfile)

# Load background
bg = readBackground(seqs, 1)

# Load motif
motiffile = system.file("extdata", "x31.tab", package = "motifcounter")
motif = t(as.matrix(read.table(motiffile)))

# Compute the per-position and per-strand scores
motifcounter:::scoreHistogramSingleSeq(seqs[[1]], motif, bg)
```
scoreProfile

Score profile across multiple sequences

Description
This function computes the per-position and per-strand average score profiles across a set of DNA sequences. It can be used to reveal positional constraints of TFBSs.

Usage
scoreProfile(seqs, pfm, bg)

Arguments
seqs A DNAStringSet or DNAString object
pfm An R matrix that represents a position frequency matrix
bg A Background object

Value
List containing

fscores Vector of per-position average forward strand scores
rscores Vector of per-position average reverse strand scores

Examples

# Load sequences
seqfile = system.file("extdata", "seq.fasta", package = "motifcounter")
seqs = Biostrings::readDNAStringSet(seqfile)

# Load background
bg = readBackground(seqs, 1)

# Load motif
motiffile = system.file("extdata", "x31.tab", package = "motifcounter")
motif = t(as.matrix(read.table(motiffile)))

# Compute the score profile
scoreProfile(seqs, motif, bg)
Description

This function computes the per-position and per-strand score in a given DNA sequence.

Usage

scoreSequence(seq, pfm, bg)

Arguments

seq A DNAString object
pfm An R matrix that represents a position frequency matrix
bg A Background object

Value

List containing

fscores Vector of scores on the forward strand
rscores Vector of scores on the reverse strand

Examples

# Load sequences
seqfile = system.file("extdata", "seq.fasta", package = "motifcounter")
seqs = Biostrings::readDNAStringSet(seqfile)

# Load background
bg = readBackground(seqs, 1)

# Load motif
motiffile = system.file("extdata", "x31.tab", package = "motifcounter")
motif = t(as.matrix(read.table(motiffile)))

# Compute the per-position and per-strand scores
scoreSequence(seqs[[1]], motif, bg)
Description

This function computes the per-position score in a given DNA strand.

Usage

scoreStrand(seq, pfm, bg)

Arguments

seq A DNAString object
pfm An R matrix that represents a position frequency matrix
bg A Background object

Details

The function returns the per-position scores for the given strand. If the sequence is too short, it contains an empty vector.

Value

scores Vector of scores on the given strand

Examples

# Load sequences
seqfile = system.file("extdata", "seq.fasta", package = "motifcounter")
seqs = Biostrings::readDNAStringSet(seqfile)

# Load background
bg = readBackground(seqs, 1)

# Load motif
motiffile = system.file("extdata", "x31.tab", package = "motifcounter")
motif = t(as.matrix(read.table(motiffile)))

# Compute the per-position and per-strand scores
motifcounter:::scoreStrand(seqs[[1]], motif, bg)
Description

This function computes the score threshold for a desired false positive probability ‘alpha’.

Usage

scoreThreshold(pfm, bg)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pfm</td>
<td>An R matrix that represents a position frequency matrix</td>
</tr>
<tr>
<td>bg</td>
<td>A Background object</td>
</tr>
</tbody>
</table>

Details

Note that the returned alpha usually differs slightly from the one that is prescribed using `motifcounterOptions`, because of the discrete nature of the sequences.

Value

List containing

- **threshold**  Score threshold
- **alpha**     False positive probability

Examples

```r
# Load sequences
seqfile = system.file("extdata", "seq.fasta", package = "motifcounter")
seqs = Biostrings::readDNAStringSet(seqfile)

# Load background
bg = readBackground(seqs, 1)

# Load motif
motiffile = system.file("extdata", "x31.tab", package = "motifcounter")
motif = t(as.matrix(read.table(motiffile)))

# Compute the score threshold
motifcounter:::scoreThreshold(motif, bg)
```
**sigLevel**

*Retrieve the false positive probability*

**Description**

This function returns the current false positive level for calling motif hits in random sequences.

**Usage**

```
sigLevel()
```

**Details**

The returned value is usually slightly smaller than the prescribed ‘alpha’ in ‘motifcounterOptions’, because of the discrete nature of sequences.

**Value**

False positive probability

**Examples**

```
motifcounter:::sigLevel()
```

---

**simulateClumpSizeDist**

*Empirical clump size distribution*

**Description**

This function repeatedly simulates random DNA sequences according to the background model and subsequently counts the number of k-clump occurrences, where denotes the clump size. This function is only used for benchmarking analysis.

**Usage**

```
simulateClumpSizeDist(pfm, bg, seqlen, nsim = 10, singlestranded = FALSE)
```

**Arguments**

- `pfm`: An R matrix that represents a position frequency matrix
- `bg`: A Background object
- `seqlen`: Integer-valued vector that defines the lengths of the individual sequences. For a given DNAStringSet, this information can be retrieved using `numMotifHits`.
- `nsim`: Integer number of random samples.
- `singlestranded`: Boolean that indicates whether a single strand or both strands shall be scanned for motif hits. Default: `singlestranded = FALSE`. 
simulateNumHitsDist

Value
A List that contains

dist Empirical distribution of the clump sizes

See Also
compoundPoissonDist, combinatorialDist

Examples

# Load sequences
seqfile = system.file("extdata", "seq.fasta", package = "motifcounter")
seqs = Biostrings::readDNAStringSet(seqfile)

# Load background
bg = readBackground(seqs, 1)

# Load motif
motiffile = system.file("extdata", "x31.tab", package = "motifcounter")
motif = t(as.matrix(read.table(motiffile)))

# Study the clump size frequencies in one sequence of length 1 Mb
seqlen = 1000000

# scan both strands
simc = motifcounter:::simulateClumpSizeDist(motif, bg, seqlen)

# scan a single strand
simc = motifcounter:::simulateClumpSizeDist(motif, bg, seqlen, singlestranded = TRUE)

simulateNumHitsDist Empirical number of motif hits distribution

Description
This function repeatedly simulates random DNA sequences according to the background model and subsequently counts how many motif hits occur in them. Thus, this function gives rise to the empirical distribution of the number of motif hits. This function is only used for benchmarking analysis.

Usage

simulateNumHitsDist(pfm, bg, seqlen, nsim, singlestranded = FALSE)
simulateNumHitsDist

**Arguments**

- **pfm**
  An R matrix that represents a position frequency matrix

- **bg**
  A Background object

- **seqlen**
  Integer-valued vector that defines the lengths of the individual sequences. For a given DNAStringSet, this information can be retrieved using `numMotifHits`.

- **nsim**
  Integer number of random samples.

- **singlestranded**
  Boolean that indicates whether a single strand or both strands shall be scanned for motif hits. Default: `singlestranded = FALSE`.

**Value**

A List that contains

- **dist**
  Empirical distribution of the number of motif hits

**See Also**

`compoundPoissonDist`, `combinatorialDist`

**Examples**

```r
# Load sequences
seqfile = system.file("extdata", "seq.fasta", package = "motifcounter")
seqs = Biostrings::readDNAStringSet(seqfile)

# Load background
bg = readBackground(seqs, 1)

# Load motif
motiffile = system.file("extdata", "x31.tab", package = "motifcounter")
motif = t(as.matrix(read.table(motiffile)))

# Study the counts in one sequence of length 150 bp
seqlen = rep(150, 1)

# Compute empirical distribution of the number of motif hits
# by scanning both strands using 100 samples
simc = motifcounter:::simulateNumHitsDist(motif, bg, seqlen, nsim = 100, singlestranded = FALSE)

# Compute empirical distribution of the number of motif hits
# by scanning a single strand using 100 samples
simc = motifcounter:::simulateNumHitsDist(motif, bg, seqlen, nsim = 100, singlestranded = TRUE)
```
Index

* MotifEnrichment
  motifcounter-package, 3

* PFM,
  motifcounter-package, 3
  .Background (Background-class), 5
  .Overlap (Overlap-class), 28

Background-class, 5

clumpSizeDist, 5
combinatorialDist, 7, 9, 23, 41, 42
compoundPoissonDist, 7, 8, 10, 20, 23, 41, 42
computeClumpStartProb, 9

generateDNAString, 11
generateDNAStringSet, 11, 12, 12
getAlpha, 13
getBeta, 13
getBeta3p, 14
getBeta5p, 14
getCounts, 15
gamma, 15
getOrder, 16
getSinglestranded, 16
getStation, 17
getTrans, 17

hitStrand, 18

lenSequences, 19

markovModel, 19
motifAndBackgroundValid, 20
motifcounter (motifcounter-package), 3
motifcounter-package, 3
motifcounterOptions, 3, 21, 39
motifEnrichment, 22
motifHitProfile, 23
motifHits, 24
motifValid, 25

normalizeMotif, 26
numMotifHits, 7–10, 12, 20, 27, 33, 40, 42
Overlap-class, 28
probOverlapHit, 6–10, 20, 28, 28
readBackground, 5, 29
revcompMotif, 30
scoreDist, 31, 32–34
scoreDistBf, 32
scoreDistEmpirical, 33
scoreHistogram, 34
scoreHistogramSingleSeq, 35
scoreProfile, 36
scoreSequence, 37
scoreStrand, 38
scoreThreshold, 39
sigLevel, 40
simulateClumpSizeDist, 40
simulateNumHitsDist, 41