cosmo

November 11, 2009

R topics documented:

`addCon` *Add constraints to an already existing constraint set*

Description

Given a constraint set, the addCon command allows to add either interval-specific or global constraints to the constraintSet object. The possible interval-specific constraints are: boundCon (for information content bounding constraints), posFreqCon (for position frequency constraints), and shapeCon (for shape constraints). The possible global constraints are: subMotifCon (when part of the motif is known), palCon (for palindromic constraints) and shapeDiffCon (for information content differences between interval extremities). Each constraint can be built with its associated make-command: makeBoundCon, makePosFreqCon, makeShapeCon, makeSubMotifCon, makePalCon and makeShapeDiffCon.

Usage

```
addCon(conSet, constraint, int=1)
```

Arguments

`conSet`	an object of class `"constraint set"`
`constraint`	list of constraints constructed with one of the 6 make constraint commands. The length of the list should be the same as that of the 'int' argument. Instead of a list, this may also be a single constraint.
`int`	vector of the intervals to which the constraints given in the constraint argument should be applied. There should be a one-to-one correspondance between the elements of the list and the interval argument. For global constraints, the value of the interval is of no importance but should be present and numeric.

Value

The output will be an object of class `"constraintSet"` resulting of the addition of the constraints to the original constraint set passed as first argument.

Author(s)

Fabian Gallusser, ⟨fgallusser@berkeley.edu⟩

See Also

`boundCon`, `posFreqCon`, `shapeCon`, `subMotifCon`, `shapeDiffCon`, `palCon` `makeBoundCon`, `makePosFreqCon`, `makeShapeCon`, `makeSubMotifCon`, `makeShapeDiffCon`, `makePalCon`

Examples

```
set=makeConSet(4,c("B","P","V","B"),c(4,50,NA,4))
con1=makeBoundCon(1,2)
con2=makePalCon(1,4,0.5)
con3=makeSubMotifCon("TATA",0.6)
con4=makePosFreqCon("2","A",0.5)
con5=makePosFreqCon("All","G",0.4)
conSet=addCon(set,list(con1,con2,con3,con4,con5),c(2,NA,NA,1,3))
```

```
#Because of the palindromic constraint on intervals 1 and 4, the fourth
#interval will inherit the nucleotide position frequency constraint
#assigned to the first interval.
```

`align-class` *Class "align"*

Description

An object of class `"align"` summarizes the motif occurrences that were predicted by `cosmo`. For each predicted motif occurrence, it contains the sequences on which the site was found, the position on that sequence, the orientation of the motif (1 if found on the forward strand, -1 otherwise), the site itself, as well as the posterior probability of a motif occurrence at that site.

Objects from the Class

Objects can be created by calls of the form `new("align", ...)`.

Slots

`seq`: Object of class `"numeric"` The sequences on which the sites were predicted.

`pos`: Object of class `"numeric"` The starting postions of the discovered sites.

`orient`: Object of class `"numeric"` The orientation in which each motif was discovered: 1 for the forward strand orientation, and -1 for the reverse complement orientation

`motif`: Object of class `"character"` The site that was predicted to be a motif occurrence.

`prob`: Object of class `"numeric"` The posterior probability of a motif occurrence at this position

`eval`: Object of class `"numeric"` The E-value of the multiple alignment containing the predicted motif occurrences

Methods

summary `signature(object = "align",...)` Prints the discovered motifs along with the sequences they were discovered on, the starting positions, the strand, and the posterior probability of a motif occurrence at this site. Also prints the E-value of the discovered alignment.

print `signature(x = "align",...)` Prints the discovered motifs along with the sequences they were discovered on, the starting positions, the strand, and the posterior probability of a motif occurrence at this site.

show `signature(object = "align")` Prints the discovered motifs along with the sequences they were discovered on, the starting positions, the strand, and the posterior probability of a motif occurrence at this site.

Author(s)

Oliver Bembom, ⟨bembom@berkeley.edu⟩

`bfile2tmat` *Converting a MEME-style background file to a transition matrix*

Description

bfile2tmat converts a MEME-style background file for specifiying the background Markov model into a transition matrix.

Usage

```
bfile2tmat(file)
```

Arguments

`file`	The MEME-style background file.

Value

The estimated transition matrix for the background Markov model. This is a list of matrices, with the first matrix given the transition probabilities for the 0th order Markov model, the second matrix giving the transition probabilities for a 1st order Markov model, and so on.

Author(s)

Oliver Bembom, ⟨bembom@berkeley.edu⟩

See Also

`cosmo`, `bgModel`

Examples

```
## path to example MEME-style background file
bfile <- system.file("Exfiles","bfile",package="cosmo")

tmat <- bfile2tmat(bfile)
```

`bgModel` *Estimating the background Markov model*

Description

bgModel() obtains an estimate of the Markov model used by cosmo() for modeling the distribution of nucleotides that are not part of the motif. bgModel() can select the order of this model data-adaptively by likelihood-based cross-validation (a k-th order Markov model allows the probability of encountering the four different nucleotides in a given position to depend on the k previous nucleotides).

Usage

```
bgModel(seqs, order=NULL, fold = 5, maxOrder = 6)
```

Arguments

`seqs`	This argument specifies the sequences that are to be used to estimate the background Markov model. If seqs == "browse", a browser appears that allows the user to select a file that contains the sequences in FASTA format. If seqs is another character string, it is assumed to give the path to a FASTA file containing the sequences of interest. Lastly, seqs may be a list with each element representing a sequence in the form of a single string such as "ACGTAGCTAG" ("seq" entry) and a description ("desc" entry).
`order`	`numerical` The order of the Markov background model. If this argument is NULL, the order is selected data-adaptively by likelihood-based cross-validation. Otherwise, a Markov model for the specified order is estimated.
`fold`	`numerical` cross-validation fold for selecting order of background Markov model
`maxOrder`	`numerical` Maximum order to consider for Markov background model.

Value

A list with the folowing elements:

`transMat`	The estimated transition matrix for the background Markov model. This is a list of matrices, with the first matrix given the transition probabilities for the 0th order Markov model, the second matrix giving the transition probabilities for a 1st order Markov model, and so on.
`order`	The selected order of the background Markov model.
`klDivs`	The Kullback-Leibler divergences for the different candidate orders for the background Markov model. Likelihood-based cross-validation selects the order with the minimum Kullback-Leibler divergence.

Author(s)

Oliver Bembom, ⟨bembom@berkeley.edu⟩

See Also

`cosmo`

Examples

```
## path to example sequence file in FASTA format
seqFile <- system.file("Exfiles","seq.fasta",package="cosmo")

## estimate transition matrix for order 2
tmat1 <- bgModel(seqFile, order=2)

## select order data-adaptively
tmat2 <- bgModel(seqFile)
```

`boundCon-class` *Class "boundCon" ~~~*

Description

This class defines a constraint concerning the lower and uper bounds of the information content over an interval. This is a useful substitute to the shapeCon when accurate knowledge of the evolution over the interval is unknown.

Objects from the Class

Objects can be created by calls of the form `new('boundCon',`
`lower = ...., # Object of class numeric`
`upper = ...., # Object of class numeric`
`)` or by: `makeBoundCon(`
`lower = ...., # Object of class numeric`
`upper = ...., # Object of class numeric`
`)`

Slots

`lower`: Object of class `"numeric"` lower bound

`upper`: Object of class `"numeric"` upper bound

Methods

No methods defined with class "boundCon" in the signature.

Note

~~further notes~~

Author(s)

Fabian Gallusser, ⟨fgallusser@berkeley.edu⟩

See Also

`shapeCon`, `subMotifCon`, `posFreqCon`, `palCon`, `shapeDiffCon` `makeConSet`

Examples

```
makeBoundCon(1,2)
```

`constraintSet-class`

Class "constraintSet" ~~~

Description

An object of class `"constraintSet"` regroups all the constraints one wishes to impose when performing detection. The object consits essentially of constraints on the motif broken down into seperate intervals, objects of class `"intInfo"`

Objects from the Class

Objects can be created by calls of the form `new('constraintSet',`

```
description = ...., # Object of class \code{"character"}
numIntervals = ...., # Object of class \code{"numeric"}
intervals = ...., # Object of class \code{"list"}
shapeDiffCon = ...., # Object of class \code{"data.frame"}
subMotifCon = ...., # Object of class \code{"data.frame"}
palCon = ...., # Object of class \code{"data.frame"}
objectCall = ...., # Object of class \code{"call"}
)
```

Slots

`description`: Object of class `"character"` a general description of the constraint set

`numIntervals`: Object of class `"numeric"` number of intervals the motif was split into

`intervals`: Object of class `"list"` a list of objects of class `"intInfo"` each describing the constraints imposed on the individual intervals

`shapeDiffCon`: Object of class `"data.frame"` a data frame summarizing the global constraints of the shape constraints. The data frame has four columns: the first two columns are the intervals and the extremity subject to the constraint, the extremity being supplied as 'a' for beginning, and 'b' for end. The third and fourth column are respectively the lower and upper bounds on the difference between the interval extremities considered. For example, if c("1b","2a",-0.2,0.5) were a row in the data frame, it would indicate that the difference in the information content between the end of the first interval and the beginning of the second should be between -0.2 and 0.5.

`subMotifCon`: Object of class `"data.frame"` a data frame specifying the submotif global constraints. The two columns respectiuvely indicate the submotif of interest and it's error tolerance. The error is computed as the difference in terms of frequency between the submotif and the position weight matrix.

`palCon`: Object of class `"data.frame"` a data frame summarizing the palindromic constraints imposed on the motif: the first two columns indicate the two palindromic intervals and the third is the error tolerated between the nucleotide frequencies in both intervals.

`objectCall`: Object of class `"call"` stores how the object was created

Methods

plot `signature(x = "constraintSet", varLen=4, propLen=4, plot.IC=TRUE, plot.nucFreq=TRUE)`: Plots the `"constraintSet"` object. varLen and propLen are integers indicating how wide to make the variable and proportion intervals (1 unit is one base pair). plot.IC and plot.nucFreq are logicals respectively indicating whether the IC content and the nucleotide frequencies are to be drawn on the plot.

print `signature(x = "constraintSet",...)` Prints the constraint set in the standard cosmo format.

show `signature(object = "constraintSet")` Prints the constraint set in the standard cosmo format.

Author(s)

Fabian Gallusser, ⟨fgallusser@berkeley.edu⟩

See Also

`writeConFile`, `constraintSet-class`, `intInfo-class`

`cosmo-class` *Class "cosmo"*

Description

An object of class `"cosmo"` is automatically created after calling the 'cosmo' function and summarizes the results.

Objects from the Class

Objects can be created by calls of the form `new("cosmo", ...)`.

Slots

`seqs`: A list of the input sequences.

`pwm`: An object of class `pwm` representing the estimated position weight matrix.

`back`: A data frame summarizing the candidate Markov models for the background distribution.

`tmat`: The estimated transition matrix/matrices used to model the distribution of background nucleotides.

`cand`: A data frame summarizing the candidate models considered.

`cons`: The constraint set applied in the final model.

`sel`: The selected model.

`motifs`: An object of class `"align"` representing the predicted motif occurrences.

`probs`: A list giving the posterior probabilities of motif occurrences along each sequence.

`objectCall`: The call which produced this object.

Methods

- **summary** `signature(object = "cosmo",...)` Summarizes the cosmo object.
- **plot** `signature(x = "cosmo",type="PWM", ...)` If `type == "PWM"`, plots the sequence logo of the discovered motif. If `type == "prob"`, plots the posterior probabilities of motif occurrences along all input sequences.
- **print** `signature(x = "cosmo",...)` Prints the discovered motifs along with the sequences they were discovered on, the starting positions, the strand, and the posterior probability of a motif occurrence at this site. Also prints the estimates position weight matrix.
- **show** `signature(object = "cosmo")` Prints the discovered motifs along with the sequences they were discovered on, the starting positions, the strand, and the posterior probability of a motif occurrence at this site.

Author(s)

Oliver Bembom, ⟨bembom@berkeley.edu⟩

`cosmo` *Constrained motif detection main function*

Description

cosmo searches a set of unaligned DNA sequences for a shared motif that may, for example, represent a common transcription factor binding site. The algorithm is similar to MEME, but also allows the user to specify a set of constraints that the position weight matrix of the unknown motif must satisfy. Such constraints may include bounds on the information content across certain regions of the unknown motif, for example, and can often be formulated on the basis of prior knowledge about the structure of the transcription factor in question.

Usage

```
cosmo(seqs="browse",constraints="None", minW=6, maxW=15,
  models = "ZOOPS", revComp = TRUE, minSites = NULL, maxSites = NULL,
  starts = 5, approx = "over", cutFac = 5, wCrit = "bic",
  wFold = 5, wTrunc = 100, modCrit = "lik", modFold = 5, modTrunc = 100,
  conCrit = "likCV", conFold = 5, conTrunc = 90, intCrit = "lik",
  intFold = 5, intTrunc = 100, maxIntensity = FALSE, lstarts = FALSE,
  backSeqs = NULL, backFold = 5, bfile = NULL, transMat = NULL,
  order = NULL, maxOrder=6, silent = FALSE)
```

Arguments

`seqs`	This argument specifies the sequences to be analyzed. If seqs == "browse", a browser appears that allows the user to select a file that contains the sequences in FASTA format. If seqs is another character string, it is assumed to give the path to a FASTA file containing the sequences of interest. Lastly, seqs may be a list with each element representing a sequence in the form of a single string such as "ACGTAGCTAG" ("seq" entry) and a description ("desc" entry).

`constraints`	These are the constraints that are to be imposed on the unknown motif. If constraints == "None", cosmo() will be run without constraints. If constraints == "GUI" and the `cosmoGUI` package has been installed, a GUI will pop up that allows the user to interactively create a set of constraints, either from scratch or on the basis of several templates of interest. If constraints is another character string, it is assumed to give the path to a file that contains the constraint definitions in the standard text format (see http://cosmoweb.berkeley.edu/constraints.html). Lastly, constraints may be an object of class constraintSet or a list of such objects that defines the constraints of interest.
`minW`	`numeric` indicating the minimum motif width to consider
`maxW`	`numeric` indicating the maximum motif width to consider
`models`	`character` a vector containing the different models to be considered for the distribution of motif occurrences ("OOPS", "ZOOPS", and "TCM"). The One-Occurrence-Per-Sequence (OOPS) model assumes that each sequence contains exactly one occurrence of the motif. The Zero-or-One-Occurrences-Per-Sequence model allows zero or one occurrences of the motif on a given sequence. The Two-Compoment-Mixture (TCM) model allows an arbitrary number of motif occurrences on a given seqence.
`revComp`	`logical` indicating whether motifs are allowed to occur in the reverse complement orientation.
`minSites`	`numerical` The minimum number of motif occurrences in the input sequences (default: 2)
`maxSites`	`numerical` The maximum number of motif occurrences in the input sequences (default: MIN(5*number of sequences, 50))
`starts`	`numerical` number of starting values to use for each optimization
`approx`	approximation for TCM likelihood; one of "over", "cut", "exact"
`cutFac`	`numerical` if TCM model is approximated by over or cut models, subsequences are of length cutFac * motif width
`wCrit`	Criterion for choosing the motif width. This can be either "lik" for the likelihood, "aic" for Akaike's Information Criterion, "bic" for the Bayesian Information Criterion, "eval" for the E-value of the alignment of the predicted motif sites, or "likCV" for likelihood-based cross-validation.
`wFold`	`numerical` cross-validation fold for selecting motif width
`wTrunc`	`numerical` truncate loss-function for selecting motif width to this percentile (1-100)
`modCrit`	Criterion for choosing the model type. This can be either "lik" for the likelihood, "aic" for Akaike's Information Criterion, "bic" for the Bayesian Information Criterion, "eval" for the E-value of the alignment of the predicted motif sites, or "likCV" for likelihood-based cross-validation.
`modFold`	`numerical` cross-validation fold for selecting the model type
`modTrunc`	`numerical` truncate loss-function for selecting model type to this percentile (1-100)
`conCrit`	Criterion for choosing the constraint set. This can be either "lik" for the likelihood, "eval" for the E-value of the alignment of the predicted motif sites, "likCV" for likelihood-based cross-validation, or "pwmCV" for cross-validation based on the Euclidean norm between two position weight matrices.
`conFold`	`numerical` cross-validation fold for selecting the constraint set (likelihood cross-validation only).

`conTrunc`	`numerical`truncate loss-function for selecting constraint set to this percentile (1-100)
`intCrit`	Criterion for estimating the intensity parameter in the ZOOPS or TCM model. This can be either "lik" for the likelihood, "aic" for Akaike's Information Criterion, "bic" for the Bayesian Information Criterion, or "eval" for the E-value of the alignment of the predicted motif sites.
`intFold`	`numerical` cross-validation fold for selecting the intensity parameter
`intTrunc`	`numerical` truncate loss-function for selecting intensity parameter to this percentile (1-100)
`maxIntensity`	`logical` maximize likelihood function with respect to intensity parameter (in ZOOPS or TCM model) instead of using profiling approach?
`lstarts`	`logical` should likelihood-based starting values be used rather than E-value-based starting values?
`backSeqs`	This argument specifies the sequences that are to be used to estimate the background Markov model. If backseqs == NULL, the background model is estimated from the sequences supplied in the seqs argument. If backSeqs == "browse", a browser appears that allows the user to select a file that contains the sequences in FASTA format. If backSeqs is another character string, it is assumed to give the path to a FASTA file containing the sequences of interest. Lastly, backSeqs may be a list with each element representing a sequence in the form of a single string such as "ACGTAGCTAG" ("seq" entry) and a description ("desc" entry).
`backFold`	`numerical` cross-validation fold for selecting order of background Markov model.
`bfile`	`character` The name of a MEME-style background file for specifying the background Markov model. Such a file lists the frequencies of all tuples of all possible tuples of length up to order + 1. See the help file on the function `bfile2tmat()` for an example.
`transMat`	The transition matrix to use for the background Markov model. This is a list of matrices, with the first matrix given the transition probabilities for the 0th order Markov model, the second matrix giving the transition probabilities for a 1st order Markov model, and so on. The entry in cell(i,j) of a k-th order transition matrix gives the probability of observing the nucleotide in column j given that the previous k nucleotides are equal to those in row i. Type 'data(transMats)' to look at an example. The function `bgModel` can be used to obtain a transition matrix from a set of sequences that can be used for this argument. The function `bfile2tmat` may be used to obtain a transition matrix from a MEME-style background file.
`order`	`numerical` order of Markov background model
`maxOrder`	`numerical` maximum order to consider for Markov background model
`silent`	`logical` suppress output?

Value

An object of class `cosmo`, returning all the results of the motif detection analysis.

Author(s)

Oliver Bembom, ⟨bembom@berkeley.edu⟩, Fabian Gallusser, ⟨fgallusser@berkeley.edu⟩

References

Oliver Bembom, Sunduz Keles, and Mark J. van der Laan, "Supervised Detection of Conserved Motifs in DNA Sequences with cosmo" (2007). Statistical Applications in Genetics and Molecular Biology: Vol. 6 : Iss. 1, Article 8. http://www.bepress.com/sagmb/vol6/iss1/art8

See Also

`bgModel`, `bfile2tmat`

Examples

```
## initialize constraint set
## consisting of three intervals
## 1st and 3rd intervals are 3bp long
## middle interval is variable lenght
conSet <- makeConSet(numInt=3, type=c("B","V","B"),length=c(3,NA,3))

## construct two bound constraints
boundCon1 <- makeBoundCon(lower=1.0, upper=2.0)
boundCon2 <- makeBoundCon(lower=0.0, upper=1.0)

## construct palindromic constraint
## require intervals 1 and 3 to be palindromes
## to within 0.05 tolerance
palCon1 <- makePalCon(int1=1, int2=3, errBnd=0.05)

## add constraints to initial constraint set
constraint <- list(boundCon1, boundCon2, palCon1)
int <- list(1, 2, NA)
conSet <- addCon(conSet=conSet, constraint=constraint, int=int)

## path to example sequence file in FASTA format
seqFile <- system.file("Exfiles","seq.fasta",package="cosmo")

## search for motifs of width 8
## assume zero or one occurrences of motif per sequence (ZOOPS)
res <- cosmo(seqs=seqFile, constraints=conSet, minW=8, maxW=8, models="ZOOPS")
plot(res)
```

`intInfo-class` *Class "intInfo" ~~~*

Description

An object of class `"intInfo"` contains all the constraints one wishes to impose on an interval of the motif. Combined `"intInfo"` objects will constitue an object of class `"constraintSet"`

Objects from the Class

Objects can be created by calls of the form `new('intInfo',`
`constraintID = ...., # Object of class \code{"numeric"}`
`intervalID = ...., # Object of class \code{"numeric"}`
`type = ...., # Object of class \code{"character"}`

```
length = ...., # Object of class \code{"numeric"}
prop = ...., # Object of class \code{"numeric"}
boundedCon = ...., # Object of class \code{"data.frame"}
posFreqConCon = ...., # Object of class \code{"data.frame"}
shapeCon = ...., # Object of class \code{"data.frame"}
objectCall = ...., # Object of class \code{"call"}
)
```

Slots

`constraintID`: Object of class `"numeric"`, indicating which constraint set the interval is part of

`intervalID`: Object of class `"numeric"`, indicating the order of the interval in the constraint set

`type`: Object of class `"character"`, the interval type: ’B’ for base pairs, ’P’ for proportion, or ’V’ for variable

`length`: Object of class `"numeric"`, the length of the interval: if the interval is of type ’B’, the length is the number of base pairs, if the interval is of type ’P’ or ’V’, this slot is NA

`prop`: Object of class `"numeric"`, the ratio of the interval lengtb to the total motif width. This slot is only defined for intervals of type ’P’, in which case the value is between 0 and 100%. For intervals of type ’B’ or ’V’, this slot is NA

`boundedCon`: Object of class `"data.frame"` a data frame summarizing the information content bounding constraints on the interval: the first column indicates the position of the interval among the intervals constituing the constraint set, the second and third column are respetively the lower and upper bound of the information content on the interval.

`posFreqCon`: Object of class `"data.frame"` a data frame summarizing the position nucleotide frequency constraints imposed on the interval: the first column indicates the interval to which the constraint is applied, the second column the position concerned (’All’ cab be specified if the constraint applies to the entire interval). It is to be noted that a specific position mat only be specified if the interval is of type ’B’. The third column is the nucleotide concerned, either A, C, G, T, AT, or GC are accepted at this time. Finally the fourth column indicates the lower bound for the given nucleotide at the given position.

`shapeCon`: Object of class `"data.frame"` a data frame summarizing the shape constraint on the interval. The first column indicates the interval to which the constraint is applied, the second the shape of the variation either ’linear’, ’monotone decreasing’, or ’monotone increasing’. The third and fourth column (respectively fifth and sixth) record the lower and upper bounds of the information content at the beginning (respectively end) of the interval. Finally, the seventh column records the error tolerated.

`objectCall`: Object of class `"call"` stores how the object was created

Methods

plot `signature(x = "intInfo", varLen=4, propLen=4, plot.IC=TRUE, plot.nucFreq=TRU` plots the `"intInfo"` object. varLen and propLen are integers indicating how wide to make the variable and proportion intervals (1 unit is one base pair). plot.IC and plot.nucFreq are logicals respectively indicating whether the IC content and the nucleotide frequencies are to be drawn on the plot.

Author(s)

Fabian Gallusser, ⟨fgallusser@berkeley.edu⟩

See Also

`writeConFile`,

`license.cosmo` *Print cosmo license*

Description

This function prints the license under which cosmo is made available.

Usage

```
license.cosmo()
```

Value

Null.

Author(s)

Oliver Bembom, ⟨bembom@berkeley.edu⟩

Examples

```
license.cosmo()
```

`makeBoundCon` *Constructing a bound constraint*

Description

This function constructs a `boundCon` object representing a constraint on the unknown position weight matrix that requires the information content to be bounded between a given lower bound and a given upper bound.

Usage

```
makeBoundCon(lower, upper)
```

Arguments

`lower`	`numeric` the lower bound on the information content
`upper`	`numeric` the upper bound on the information content

Value

An object of class `boundCon`.

Author(s)

Fabian Gallusser, ⟨fgallusser@berkeley.edu⟩

See Also

`boundCon`, `subMotifCon`, `posFreqCon`, `shapeCon`, `shapeDiffCon`, `palCon`

Examples

```
bc <- makeBoundCon(1.0, 2.0)
```

`makeConSet` *Constructing a constraint set.*

Description

The first step to building a constraint set is to define the number and type of intervals defining the set. This is done using the makeConSet function. Constraints are then built using the makeBoundCon (for information content bounding constraints), makeIntFreqCon (for interval nucleotide frequency constraints), makePosFreqCon (for position frequency constraints) and finally makeShapeCon (for shape constraints). The constraints are then added to the sets created with makeConSet using the addCon command.

Usage

```
makeConSet(numInt,type,length,descrip="Constraint Set")
```

Arguments

`numInt`	integer relating to the number of intervals composing the set
`type`	character vector describing the types of each of the intervals composing the set: the length of this vector should be equal to the 'numInt' argument. Possible values are 'B' for basepairs, 'V' for variable, and 'P' for proportion
`length`	numeric vector of length 'numInt', with a numeric argument describing the lengths of the intervals: either the number of basepairs if type is 'B', proportion coefficent if type is 'P'. For type 'V', the value is of no importance.
`descrip`	a character string to describe the constraint set.

Details

These commands are an alternative to the GUI for constucting constraint sets and groups.

Value

Null. Depending on the arguments, 1, 2 or 3 text files are created, by default in the working directory.

Author(s)

Fabian Gallusser, ⟨fgallusser@berkeley.edu⟩

See Also

`boundCon`, `subMotifCon`, `posFreqCon`, `shapeCon`, `shapeDiffCon`, `palCon`

Examples

```
makeConSet(4,c("B","P","V","B"),c(4,50,NA,3))
```

`makePalCon` *Constructing a palindromic constraint*

Description

This function constructs a `palCon` object representing a constraint on the unknown position weight matrix that requires the two intervals to be palindromes of each other.

Usage

```
makePalCon(int1, int2, errBnd)
```

Arguments

`int1`	`numeric` the first interval
`int2`	`numeric` the second interval
`errBnd`	`numeric` bound on how much corresponding nucleotide frequency are allowed to deviate from each other

Value

An object of class `palCon`.

Author(s)

Fabian Gallusser, ⟨fgallusser@berkeley.edu⟩

See Also

`boundCon`, `subMotifCon`, `palCon`, `posFreqCon`, `shapeDiffCon`, `palCon`

Examples

```
pfc <- makePalCon(1, 3, 0.05)
```

`makePosFreqCon` *Constructing a nucleotide frequency constraint*

Description

This function constructs a `posFreqCon` object representing a constraint on the unknown position weight matrix that requires the frequency of a given nucleotide to be above a given lower bound, either at a single position in a given interval, or on average across the entire interval

Usage

```
makePosFreqCon(pos, nuc, lower)
```

Arguments

`pos`	This is either the string `avg`, indicating that the average nucleotide frequency across the interval is bounded from below, or a position in the interval, indicating the nucleotide frequency at that position in the interval is bounded from below.
`nuc`	`character` This is one of `A`, `C`, `G`, `T`, `GC`, `AT`.
`lower`	`numeric` the lower bound on the nucleotide frequency of interest

Value

An object of class `posFreqCon`.

Author(s)

Fabian Gallusser, ⟨fgallusser@berkeley.edu⟩

See Also

`boundCon`, `shapeCon`, `subMotifCon`, `posFreqCon`, `shapeDiffCon`, `palCon`

Examples

```
pfc <- makePosFreqCon("avg","GC", 0.75)
```

`makeShapeCon` *Constructing a shape constraint*

Description

This function constructs a `shapeCon` object representing a constraint on the unknown position weight matrix that requires the information content to follow a particular functional form.

Usage

```
makeShapeCon(shape, sLower, sUpper, eLower, eUpper, error)
```

Arguments

`shape`	`character` the functional form of the information content. This must be either `Linear`, `MonotoneIncreasing`, or `MonotoneDecreasing`.
`sLower`	`numeric` lower bound on the information content at the start of the interval
`sUpper`	`numeric` upper bound on the information content at the start of the interval
`eLower`	`numeric` lower bound on the information content at the end of the interval
`eUpper`	`numeric` upper bound on the information content at the end of the interval
`error`	`numeric` tolerance for how much the actual information content profile is allowed to deviate from the prescribed form

Value

An object of class `shapeCon`.

Author(s)

Fabian Gallusser, ⟨fgallusser@berkeley.edu⟩

See Also

`boundCon`, `subMotifCon`, `posFreqCon`, `shapeCon`, `shapeDiffCon`, `palCon`

Examples

```
sc <- makeShapeCon("Linear",1.0, 2.0, 1.5, 2.0, 0.05)
```

`makeShapeDiffCon` *Constructing a shape parameter difference constraint*

Description

This function constructs a `shapeDiffCon` object representing a constraint on the unknown position weight matrix that requires the the difference between the information content at the edge of one interval and the information content at the edge of another interval to be bounded betwene given bounds. This constraint may only be applied to intervals that already have a shape constraint.

Usage

```
makeShapeDiffCon(int1, int2, lower, upper)
```

Arguments

`int1`	`character` the location of the first information content of interest. This is specified as the number of the interval followed by the letter `a` or `b` depending on whether the left or right edge of the interval is desired.
`int2`	`character` the location of the second information content of interest. This is specified as the number of the interval followed by the letter `a` or `b` depending on whether the left or right edge of the interval is desired.
`lower`	`numeric` the lower bound on the difference in information contents.
`upper`	`numeric` the upper bound on the difference in information contents.

Value

An object of class `shapeDiffCon`.

Author(s)

Fabian Gallusser, ⟨fgallusser@berkeley.edu⟩

See Also

`boundCon`, `shapeCon`, `subMotifCon`, `posFreqCon`, `shapeDiffCon`, `palCon`

Examples

```
## continuous information content across interval 1
sdc1 <- makeShapeDiffCon("1a", "1b", 0.0, 0.0)

## continuous information content at junction
## between intervals 1 and 2
sdc2 <- makeShapeDiffCon("1b", "2a", 0.0, 0.0)

## decreasing information content across interval 1
sdc3 <- makeShapeDiffCon("1a", "1b", 0.0, 2.0)
```

`makeSubMotifCon` *Constructing a submotif constraint*

Description

This function constructs a `subMotifCon` object representing a constraint on the unknown position weight matrix that requires the motif to contain a given submotif

Usage

```
makeSubMotifCon(submotif, minfreq)
```

Arguments

`submotif`	`character` the submotif
`minfreq`	`numeric` roughly, the minimum frequency with which nucleotides in the submotif must occur in the motif

Value

An object of class `subMotifCon`.

Author(s)

Fabian Gallusser, ⟨fgallusser@berkeley.edu⟩

See Also

`boundCon`, `shapeCon`, `subMotifCon`, `posFreqCon`, `shapeDiffCon`, `palCon`

Examples

```
pfc <- makeSubMotifCon("CGC", 0.8)
```

`motifPWM` *Example position weight matrix*

Description

Example position weight matrix of a motif of width 8 with consensus sequence CGCGCGCG.

Usage

```
data(motifPWM)
```

Format

A 4 by 8 matrix with PWM entries.

Examples

```
data(motifPWM)
```

`palCon-class` *Class "palCon" ~~~*

Description

This class defines a global constraint when palindromic patterns are known to be found in the motif.

Objects from the Class

Objects can be created by calls of the form `new('palCon',`

```
int1 = ...., # Object of class numeric
int2 = ...., # Object of class numeric
errBnd = ...., # Object of class numeric
```

`)` or by: `makePalCon(`

```
int1 = ...., # Object of class numeric
int2 = ...., # Object of class numeric
errBnd = ...., # Object of class numeric
)
```

Slots

`int1`: Object of class `"numeric"` first palindromic interval

`int2`: Object of class `"numeric"` second palindromic interval

`errBnd`: Object of class `"numeric"` error tolerated in mismatches

Methods

No methods defined with class "palCon" in the signature.

Note

~~further notes~~

Author(s)

Fabian Gallusser, ⟨fgallusser@berkeley.edu⟩

See Also

`shapeCon`, `subMotifCon`, `posFreqCon`, `boundCon`, `shapeDiffCon` `makeConSet`

Examples

```
set=makeConSet(3,c("B","V","B"),c(4,NA,4))
palcon=makePalCon(1,3,0.5)
set=addCon(set,list(palcon),NA)
```

`posFreqCon-class` *Class "posFreqCon" ~~~*

Description

This class defines a constraint concerning the lower bound for the proportion of a nucleotide at a particular position of the interval

Objects from the Class

Objects can be created by calls of the form `new('posFreqCon',`

```
pos = ...., # Object of class character
nuc = ...., # Object of class character
lower = ...., # Object of class numeric
```

`)` or by: `makePosFreqCon(`

```
pos = ...., # Object of class character
nuc = ...., # Object of class character
lower = ...., # Object of class numeric
)
```

Slots

`pos`: Object of class `"numeric"` position in the interval of the nucleotide affected by the constraint

`nuc`: Object of class `"character"` nucleotide to which the constraint is applied

`lower`: Object of class `"numeric"` lower bound for the nucleotide's frequency

Methods

No methods defined with class "posFreqCon" in the signature.

Note

~~further notes~~

Author(s)

Fabian Gallusser, ⟨fgallusser@berkeley.edu⟩

See Also

`boundCon`, `shapeCon`, `subMotifCon`, `shapeDiffCon`, `palCon` `makeConSet`

Examples

```
set=makeConSet(2,c("B","V"),c(5,NA))
posCon1=makePosFreqCon("2","A",0.5)
posCon2=makePosFreqCon("All", "G", 0.3)
conSet=addCon(set,list(posCon1,posCon2),c(1,2))
# Because the first nucleotide position frequency constraint applies to
#a specific position, it can only be applied to the interval of type "B".
```

`postProbs-class` *Class "postProbs"*

Description

An object of class `"postProbs"` collects information about the posterior probability of motif occurrences in each of the eligible positions of the input sequences. Most notably these probabilities can be plotted using the plot function.

Objects from the Class

Objects can be created by calls of the form `new("postProbs", ...)`.

Slots

`seqMat`: Object of class `"matrix"` The sequences in numerical matrix format as output by cosmo.

`seqNames`: `"list"`A list of the sequence names.

`probs`: `"numeric"`A matrix of the posterior probabilities.

`revcomp`: `"logical"`A matrix indicating whether the motif is more likely to occure in the reverse complement orientation.

Methods

plot `signature(x = "postProbs")` Plots the posterior probabilities. If the motif is more likely to occur in the forward strand orientation, the bar extends upward from the horizontal, otherwise it extends downward.

Author(s)

Oliver Bembom, ⟨bembom@berkeley.edu⟩

`readConFile` *Reads in a constraint file.*

Description

This function reads in a contraints file into an object of class `constraintSet` or a list of such objects.

Usage

```
readConFile(conFile, description="Constraints")
```

Arguments

`conFile`	Path to the constraint file that is to be read in
`description`	Description of the constraint group that is to be created

Value

An object of class `constraintSet` or a list of such objects that contains the constraint definitions found in the constraint file.

Author(s)

Fabian Gallusser, ⟨fgallusser@berkeley.edu⟩

See Also

`constraintSet`, `writeConFile`

Examples

```
conFile <- system.file("Exfiles","conFile",package="cosmo")
cons <- readConFile(conFile)
```

`rseq` *Random generation of DNA sequence according to ZOOPS or TCM model*

Description

This function randomly generates a number of DNA sequences that contain a given motif accordint to the ZOOPS or TCM model. In the ZOOPS model, each sequence contains one or zero occurrences of the motif. In the TCM model, each sequence may contain an arbitrary number of motif occurrences.

Usage

```
rseq(numSeqs, seqLength, rate, pwm, transMats,
     model="ZOOPS", posOnly=FALSE)
```

Arguments

`numSeqs`	`numeric` The number of sequences to be generated
`seqLength`	`numeric` The length of each sequence. This may be either a single number, in which case that number is taken to be the common length of all sequence, or a vector of sequence lengths.
`rate`	`numeric` In the ZOOPS model, this is the proportion of sequences containg a motif occurrence. In the TCM model, this the rate parameter lambda with which motifs are inserted into the sequences.
`pwm`	`numeric` Position-weight matrix of the motif to be inserted.
`transMats`	The transition matrices to use for the background Markov model. This is a list of matrices, with the first matrix given the transition probabilities for the 0th order Markov model, the second matrix giving the transition probabilities for a 1st order Markov model, and so on.
`model`	Either "ZOOPS" or "TCM"
`posOnly`	`logical` If TRUE, motifs are inserted only in the forwards orientation. Otherwise, motifs are inserted in either of the two possible orientations with equal probabilities.

Value

`seqs`	A list with one element for each sequence in the file. The elements are in two parts, one the description and the second a character string of the biological sequence.
`motifs`	An "align" object summarizing the positions of the inserted motif occurrences.
`empPWM`	An object of class `pwm` representing the position weight matrix obtained by aligning the inserted motifs.

Author(s)

Oliver Bembom, ⟨bembom@berkeley.edu⟩

Examples

```
## generate 20 sequences according to ZOOPS model
## with an expected number of 10 sequences containing a
## motif

data(motifPWM)
data(transMats)
res <- rseq(20, 250, 0.5, motifPWM, transMats,"ZOOPS")
```

`shapeCon-class` *Class "shapeCon" ~~~*

Description

This class defines a constraint concerning the evolution of the information content over an interval

Objects from the Class

Objects can be created by calls of the form `new('shapeCon',`

```
shape = ...., # Object of class character
sLower = ...., # Object of class numeric
sUpper = ...., # Object of class numeric
eLower = ...., # Object of class numeric
eUpper = ...., # Object of class numeric
error = ...., # Object of class numeric
```

`)` or by: `makeShapeCon(`

```
shape = ...., # Object of class character
sLower = ...., # Object of class numeric
sUpper = ...., # Object of class numeric
eLower = ...., # Object of class numeric
eUpper = ...., # Object of class numeric
error = ...., # Object of class numeric
)
```

Slots

`shape`: Object of class `"character"` This slot describes how the information content varies across the interval: 'Linear', 'Monotone Increasing' and 'Monotone Decreasing' are currently implemented

`sLower`: Object of class `"numeric"` starting lower bound

`sUpper`: Object of class `"numeric"` starting upper bound

`eLower`: Object of class `"numeric"` ending lower bound

`eUpper`: Object of class `"numeric"` ending upper bound

`error`: Object of class `"numeric"` error tolerated

Methods

No methods defined with class "shapeCon" in the signature.

Note

~~further notes~~

Author(s)

Fabian Gallusser, ⟨fgallusser@berkeley.edu⟩

See Also

`boundCon`, `subMotifCon`, `posFreqCon`, `shapeDiffCon`, `palCon` `makeConSet`

Examples

```
makeShapeCon("Monotone Increasing",sLower=0,sUpper=1,eLower=1.5,eUpper=2,error=0.5)
```

`shapeDiffCon-class` *Class "shapeDiffCon" ~~~*

Description

This class defines a global constraint for the bounds of the difference of the information content at two interval extremities.

Objects from the Class

Objects can be created by calls of the form `new('shapeDiffCon',`

```
int1 = ...., # Object of class character
int2 = ...., # Object of class character
lower = ...., # Object of class numeric
upper = ...., # Object of class numeric
```

`)` or by: `makeShapeDiffCon(`

```
int1 = ...., # Object of class character
int2 = ...., # Object of class character
lower = ...., # Object of class numeric
upper = ...., # Object of class numeric
)
```

Slots

`int1`: Object of class `"character"` designing the first interval and extremity of the constraint: the first character is the interval number, the second is the extremity: 'a' for beginning and 'b' for end

`int2`: Object of class `"character"` designing the second interval and extremity of the constraint: the first character is the interval number, the second is the extremity: 'a' for beginning and 'b' for end

`lower`: Object of class `"numeric"` matrix containing the lower bounds of the linear constraints junctions

`upper`: Object of class `"numeric"` matrix containing the upper bounds of the linear constraints junctions

Methods

No methods defined with class "linCon" in the signature.

Note

The constraints is so that: lower <= Information content at int1 - Information content at int2 <= upper.

The 'makeShapeDiffCon' will build an object of class `"shapeDiffCon"` given the int1, int2, lower and upper values.

Author(s)

Fabian Gallusser, ⟨fgallusser@berkeley.edu⟩

See Also

`shapeCon`, `subMotifCon`, `posFreqCon`, `boundCon`, `palCon` `makeConSet`

Examples

```
makeShapeDiffCon(int1="1b", int2="2a", lower=0.2, upper=0.5)
```

`simScore` *Score motif detection simulation results*

Description

This function computes sensitivity and specifity for the results returned by cosmo.

Usage

```
  simScore(truth, cosmoOut, minOverlap=0.25)
```

Arguments

`truth`	`align` Alignment describing the true motif occurrences.
`cosmoOut`	`cosmo` The results returned by `cosmo()`.
`minOverlap`	`numeric` A predicted motif must overlap at least this proportion of a known motif to be considered a hit.

Value

`sens`	The proportion of true motif occurrences discovered (sensitivity).
`spec`	The proportion of true motif occurrences among the discovered sites (specificity).
`roc`	The area under the ROC curve.

Author(s)

Oliver Bembom, ⟨bembom@berkeley.edu⟩

See Also

`cosmo`

Examples

```
## generate 20 sequences according to OOPS model
## with an expected 50
## motif
data(motifPWM)
data(transMats)
res <- rseq(20, 100, 1.0, motifPWM, transMats,"ZOOPS")
truth <- res$motifs
seqs <- res$seqs
```

```
res <- cosmo(seqs, constraints="None", minW=8, maxW=8)
simScore(truth, res)
```

`subMotifCon-class` *Class "subMotifCon" ~~~*

Description

This class defines a global constraint when a part of the motif is partially known.

Objects from the Class

Objects can be created by calls of the form `new('subMotifCon',`
`submotif = ...., # Object of class \code{character}`
`minfreq = ...., # Object of class \code{numeric}`
`)` or by: `makeSubMotifCon(`
`submotif = ...., # Object of class \code{character}`
`minfreq = ...., # Object of class \code{numeric} indicates the lower bound for the nucleotide frequencies in order for the submotif to be considered as contained within the motif.The nucleotide frequencies are determined from the estinmated position weight matrix.`
`)`

Slots

`submotif`: Object of class `"matrix"`, it is the submotif which is thought to be contained within the motif

`minfreq`: Object of class `"numeric"` indicates the lower bound for the nucleotide frequencies in order for the submotif to be considered as contained within the motif. The nucleotide frequencies are determined from the estinmated position weight matrix.

Methods

No methods defined with class "shapeDiffCon" in the signature.

Author(s)

Fabian Gallusser, ⟨fgallusser@berkeley.edu⟩

See Also

`shapeCon`, `shapeDiffCon`, `posFreqCon`, `boundCon`, `palCon` `makeConSet`

Examples

```
submotifcon=makeSubMotifCon("TATAA", 0.5)
```

`transMats` *Example transition matrix in list format*

Description

An example transition matrix for a 2nd order transition matrix in list format, with each order having its own transition matrix.

Usage

```
data(transMats)
```

Format

The format is: List of 3 order2: num [1:16, 1:4] 0.284 0.306 0.218 0.398 0.356 ... order1: num [1:4, 1:4] 0.317 0.262 0.224 0.363 0.207 ... order0: num [1, 1:4] 0.298 0.201 0.213 0.287

Examples

```
data(transMats)
```

`writeConFile` *Converts an object of class 'constraintSet' into a text file*

Description

This function converts an object of class `constraintSet` or a list of such objects into a text file.

Usage

```
writeConFile(constraints, stringOutput=TRUE, outputFile="constraints.txt")
```

Arguments

`constraints`	object of class `constraintSet` or list of objects of class `constraintSet` which is to be converted in a text format
`stringOutput`	logical, whether the constraint should be written as a string. If FALSE, the constraint is written as a text file
`outputFile`	path and name of the constraint text file to be created

Details

Both the string and the text file can be used to submit constraints directly to the COSMO web application, which is an alternative to the GUI application. The text file has the advantage of being more easy to read and to load for future use.

Value

NULL A single text file is created, by default in the working directory.

Author(s)

Fabian Gallusser, ⟨fgallusser@berkeley.edu⟩

See Also

`constraintSet`, `readConFile`

Examples

```
set <- makeConSet(4,c("B","P","V","B"),c(4,50,NA,4))
con1 <- makeBoundCon(1,2)
con2 <- makePalCon(1,4,0.5)
con3 <- makeSubMotifCon("TATA",0.6)
con4 <- makePosFreqCon("2","A",0.5)
con5 <- makePosFreqCon("All","G",0.4)
conSet <- addCon(set,list(con1,con2,con3,con4,con5),c(2,NA,NA,1,3))
conString <- writeConFile(conSet)
```

Index