Package 'rphast'

February 11, 2018

Maintainer Ritika Ramani <rramani@cshl.edu>

License BSD_3_clause + file LICENSE

Title Interface to 'PHAST' Software for Comparative Genomics

Author Melissa Hubisz, Katherine Pollard, and Adam Siepel

Description Provides an R interface to the 'PHAST'(<http://compgen.cshl.edu/phast/>) software (Phylogenetic Analysis with Space/Time Models). It can be used for many types of analysis in comparative and evolutionary genomics, such as estimating models of evolution from sequence data, scoring alignments for conservation or acceleration, and predicting elements based on conservation or custom phylogenetic hidden Markov models. It can also perform many basic operations on multiple sequence alignments and phylogenetic trees.

Version 1.6.9

URL http://compgen.cshl.edu/rphast

Date 2018-01-30

Imports methods

Depends stats

Suggests ape, seqLogo

Collate 'bgc.R' 'checkArgs.R' 'feat.R' 'hmm.R' 'listOfLists.R' 'msa.R' 'optim.R' 'phastCons.R' 'phyloFit.R' 'phyloP.R' 'plot.R' 'rphast.R' 'treeModel.R' 'trees.R' 'zzz.R'

RoxygenNote 5.0.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2018-02-11 15:39:12 UTC

R topics documented:

`rphast-package` *Interface to 'PHAST' Software for Comparative Genomics*

Description

Provides an R interface to the 'PHAST'(<http://compgen.cshl.edu/phast/>) software (Phylogenetic Analysis with Space/Time Models). It can be used for many types of analysis in comparative and evolutionary genomics, such as estimating models of evolution from sequence data, scoring alignments for conservation or acceleration, and predicting elements based on conservation or custom phylogenetic hidden Markov models. It can also perform many basic operations on multiple sequence alignments and phylogenetic trees.

Details

Copyright:	All code is Copyright (c) 2002-2018 University of California, Cornell University, Cold Spring Harbor Laboratory.
Package:	rphast
License:	BSD_3_clause + file LICENSE
Version:	1.6.9
URL:	http://compgen.cshl.edu/rphast
Date:	2018-01-30
Imports:	methods
Depends:	stats
Suggests:	ape, seqLogo
Collate:	'bgc.R' 'checkArgs.R' 'feat.R' 'hmm.R' 'listOfLists.R' 'msa.R' 'optim.R' 'phastCons.R' 'phyloFit.R' 'phyloP.R' 'plot.R' 'rphast.R' 'treeModel.R' 'trees.R'

	'zzz.R'
RoxygenNote:	5.0.1
Built:	R 3.1.2; x86_64-unknown-linux-gnu; 2018-01-30 15:53:11 UTC; unix

Index:

density.feat	Features kernel density
depth.tree	Get the distance from a node to the root of a tree
dim.feat	Feature dimensions
dim.msa	Returns the dimensions of an msa object as (# of species, # of columns)
enrichment.feat	Enrichment of features with respect to annotation types
expected.subs.msa	Obtain expected number of substitutions on each branch and site
extract.feature.msa	Extract features from an MSA object
feat	Features Objects
fix.semicolon.tree	Add a semi-colon to end of tree string
fix.start.stop.feat	Fix start and stop signals
flatten.feat	Combine adjacent features with the same "feature" field
freq3x4.msa	Get codon frequencies based on 3x4 model
from.pointer.feat	Convert a features object from C memory (external pointer) to R memory
from.pointer.msa	MSA From Pointer
gc.content.msa	Get the fraction of G's and C's in an alignment
get.rate.matrix.params.tm	Get the parameters describing a rate matrix
get4d.msa	Extract fourfold degenerate sites from an MSA object
guess.format.msa	MSA Guess Format
hist.feat	plot histogram of feature lengths
hmm	Create an rphast HMM object
informative.regions.msa	Get informative regions of an alignment
inverse.feat	Get inverse features
is.format.msa	Check an MSA Format String
is.msa	Check an MSA object
is.ordered.msa	MSA is Ordered?
is.subst.mod.tm	Check Substitution Model Strings
is.tm	Tree Models
is.track	Is this a track?
label.branches	Label tree branches
label.subtree	Label subtree
leafnames.tree	Get the names of a tree's leaf nodes
likelihood.msa	MSA Likelihood
mod.backgd.tm	Adjust tree model background frequencies while maintaining reversibility
msa	MSA Objects
name.ancestors	Name Ancestral Nodes
names.msa	MSA Sequence Names
ncol.feat	Number of Columns in Features
ncol.msa	MSA Sequence Length.

ninf.msa	The number of informative columns in an alignment
nothanks.rphast	Stop rphast registration reminders
nrow.feat	Number of Features
nrow.msa	MSA Number of Sequences
nstate.hmm	HMM number of states
numleaf.tree	Number of leaves in a Tree
numnodes.tree	Number of Nodes in a Tree
offset.msa	MSA Index Offset
optim.rphast	Optimize using phast's optimization code
overlap.feat	Feature overlap
pairwise.diff.msa	Get pairwise differences per site between sequences
phastBias	phastBias
phastCons	Produce conservation scores and identify conserved elements, given a multiple alignment and a phylo-HMM.
phyloFit	Fit a Phylogenetic model to an alignment...
phyloP	phyloP (basewise or by feature)
phyloP.prior	phyloP prior
phyloP.sph	phyloP SPH
plot.feat	Features plot
plot.gene	Gene plot
plot.lsmodel.tm	Make a bubble plot of a lineage-specific transition matrix of a tree model.
plot.msa	Plot an alignment
plot.rate.matrix	Make a bubble plot of a transition matrix
plot.tm	Make a bubble plot of the transition matrix for a tree model.
plot.track	Make browser-like plot in rphast
postprob.msa	Obtain posterior probilities of every state at every node
print.feat	Printing a features Object
print.msa	Printing MSA objects
print.phastBiasResult	Pretty-print the phastBias result list without spilling giant matrices onto the screen
print.tm	Printing Tree Models
prune.tree	Prune a Tree
range.feat	Features range
range.track	Get the coordinate range of a list of RPHAST results
rbind.feat	concatenate feature objects
read.feat	Read a Feature File (GFF, BED, or GenePred)
read.hmm	Read an HMM object from a file
read.msa	Reading an MSA Object
read.newick.tree	Read a Newick Tree from a File
read.tm	Read a Tree Model
read.wig	Read a wig file

reflect.phylo.hmm	Reflect a phylo-hmm across a strand
register.rphast	Register RPHAST
rename.tree	Tree Node Renaming
rescale.tree	Scale a Tree or Subtree
reverse.complement.msa	Reverse complement a multiple sequence alignment
sample.msa	Sample columns from an MSA
score.hmm	Score an alignment using a general phylo-HMM
set.rate.matrix.tm	Set the rate matrix of a tree model using model-specific parameters.
setup.branch.site.tm	Set up a tree model for branch site selection analysis
simulate.msa	Simulate a MSA given a tree model and HMM.
smooth.wig	Smooth a wig plot in rphast
sort.feat	Sort a GFF
split.by.feature.msa	Split an MSA by feature
split.feat	Split features by length
state.freq.msa	Get the observed frequencies of states in an alignment
strip.gaps.msa	MSA Strip Gaps
sub.msa	MSA Subset
subst.mods	List PHAST Substitution Models
subtree	Subtree
summary.feat	Features Summary
summary.msa	MSA Summary
summary.tm	Tree Model Summary
summary.tree	Get a summary of a Newick-formatted tree, edge lengths, node names, etc
tagval	Extract value from tag-value formatted attributes
tagval.feat	Extract value from tag-value formatted attribute in features object
tm	Tree Models
total.expected.subs.msa	Obtain expected number of substitutions of each type on each branch
translate.msa	Get amino acid sequences from an alignment
unapply.bgc.sel	Unapply bgc+selection parameters from a matrix
unique.feat	Remove overlapping genes
write.feat	Writing a features Object
write.hmm	Write an HMM object to a file
write.msa	Writing MSA Objects to Files
write.tm	Wrting Tree Models
write.wig	Writing a wig file
write.wig.feat	Write a features object in fixedStep wig format

Author(s)

Melissa Hubisz, Katherine Pollard, and Adam Siepel

Maintainer: Ritika Ramani <rramani@cshl.edu>

`add.introns.feat` *Add introns to features*

Description

Add introns to features

Usage

```
add.introns.feat(x)
```

Arguments

`x`	An object of type `feat`. CDS regions must be present with type "CDS", and the transcript_id must be indicated in the attribute field.

Value

An object of type `feat`, with all the entries of the original object, but also with intron annotations.

Note

If x is stored as a pointer to an object stored in C, introns will be added to x.

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
exampleArchive <- system.file("extdata", "examples.zip", package="rphast")
featFile <- "gencode.ENr334.gp"
unzip(exampleArchive, featFile)
f <- read.feat(featFile)
table(f$feature)
coverage.feat(f[f$feature=="CDS",])
coverage.feat(f[f$feature=="exon",])
f <- add.introns.feat(f)
table(f$feature)
coverage.feat(f[f$feature=="intron",])
unlink(featFile)
```

add.ls.mod *Add a lineage-specific model*

Description

Lineage-specific models allow a different substitution model to be defined on a specified set of branches. An entirely different substitution model can be used, as long as it is of the same order and has the same number of states as the model used in the rest of the tree. Or, if the same substitution model is used, certain parameters can be optimized separately from the main model, whereas others are shared with the main model.

Usage

```
add.ls.mod(x, branch = NULL, label = NULL, category = 0,
  subst.mod = NULL, separate.params = NULL, const.params = NULL,
  backgd = NULL, selection = NULL, bgc = NULL)
```

Arguments

x	An object of type tm
branch	If the lineage-specific model applies to a single branch, it can be named here using the name of the node which descendes from the branch. See name.ancestors for naming internal nodes.
label	(Alternative to branch). The label which identifies the branch(es) which this lineage-specific model should apply to. Labels are denoted in a tree with a pound sign and label following the node. See label.branches and label.subtree to add a label to a tree.
category	An integer indicating which category/categories to apply the lineage-specific model. This only works if x$nratecats > 1. A value of 0 or NULL implies all categories. Otherwise this can be an integer (or vector of integers) from 1..x$nratecats.
subst.mod	A character string indicating the substitution model to be used for the lineage-specific model. If NULL, use the same model as the rest of the tree. See subst.mods for a list of possible substituion models.
separate.params	(Only applies if subst.mod is the same as main model) A vector of character strings indicating which parameters to estimate separately. Possible values are "kappa", "sel", "bgc", "gap_param", "backgd", and "ratematrix". If backgd, selection, or bgc are provided as arguments, they are automatically considered separate parameters and do not need to be explicitly listed here. "ratematrix" implies all parameters describing the substitution model (but does not include backgd, sel, or bgc). Boundaries can be optionally appended to parameter names with brackets, ie, "kappa[1,10]" will set boundaries for kappa between 1 and 10 (see "Parameter boundaries" section of phyloFit). If subst.mod is different from the main model, then no parameters are shared with main model. However the equilibrium frequencies can be shared by setting backgd to NULL.

const.params	A character vector indicating which parameters to hold constant at their initial values, rather than being optimized upon a call to phyloFit. Possible values are the same as for separate.params, although no boundaries can be given here.
backgd	The initial equilibrium frequencies to use for this model. If NULL, use the same as in the main model.
selection	The selection parameter (from the sel+bgc model), relative to selection in the main model.
bgc	The bgc parameter (from the sel+bgc model).

Details

A lineage-specific model is stored as a list with the following elements: defn, rate.matrix, and optional elements backgd, selection, bgc.

defn is a character string which defines the model in a way that phast can parse; it is a colon-delimited string with 2 or 3 elements. The first element indicates which branches the model applies to, the second indicates which substitution model to use or which parameters to optimize if the same substitution model is used (and also may impose boundaries on these parameters). The optional third element is a list of parameters which will not be optimized by phyloFit.

backgd is the initial set of equilibrium frequencies for this model; if not present, then the equilibrium frequencies will be shared with the main model.

selection and bgc are optional parameters for the model with biased gene conversion and selection. If they are not provided this model is not used. Note that selection is defined relative to selection in the main model, if x$selection is not NULL (so the total selection in the lineage-specific model is the sum of the selection value in the main and lineage-specific model.

A tree model can have multiple lineage-specific models; if a later model applies to the same branch as an earlier model, then the later one overrides it.

All lineage-specific models are stored in the ls.model element of the tm object.

Value

An object of type tm, identical to the input model but with a new lineage-specific model added on. This lineage-specific model is not validated by this function.

Author(s)

Melissa J. Hubisz

---

add.signals.feat *Add start/stop codon, 3'/5' splice signals to features*

Description

Add start/stop codon, 3'/5' splice signals to features

Usage

```
add.signals.feat(x)
```

Arguments

x	An object of type feat. CDS regions must be present with type "CDS", and the transcript_id must be indicated in the attribute field.

Value

An object of type feat, with all the entries of the original object, but also with stop codons, start, codons, 3' splice, and 5' splice sites annotated.

Note

- If x is stored as a pointer to an object stored in C, signals will be added to x.
- Does not correctly handle case of splice site in middle of start or stop codon.

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
exampleArchive <- system.file("extdata", "examples.zip", package="rphast")
featFile <- "gencode.ENr334.gp"
unzip(exampleArchive, featFile)
f <- read.feat(featFile)
table(f$feature)
coverage.feat(f[f$feature=="CDS",])
coverage.feat(f[f$feature=="exon",])
f <- add.signals.feat(f)
table(f$feature)
coverage.feat(f[f$feature=="3'splice",])
coverage.feat(f[f$feature=="5'splice",])
coverage.feat(f[f$feature=="start_codon",])
coverage.feat(f[f$feature=="stop_codon",])
unlink(featFile)
```

---

add.UTRs.feat *Add UTRs to features*

Description

Add UTRs to features

Usage

```
add.UTRs.feat(x)
```

Arguments

x	An object of type `feat`. CDS regions must be present with type "CDS", and the transcript_id must be indicated in the attribute field.

Value

An object of type `feat`, with all the entries of the original object, but also with UTR annotations.

Note

If x is stored as a pointer to an object stored in C, then UTRs will be added to x.

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
exampleArchive <- system.file("extdata", "examples.zip", package="rphast")
featFile <- "gencode.ENr334.gp"
unzip(exampleArchive, featFile)
f <- read.feat(featFile)
table(f$feature)
coverage.feat(f[f$feature=="CDS",])
coverage.feat(f[f$feature=="exon",])
f <- add.UTRs.feat(f)
table(f$feature)
coverage.feat(f[f$feature=="3'UTR",])
coverage.feat(f[f$feature=="5'UTR",])
unlink(featFile)
```

---

alphabet.msa *MSA Alphabet*

Description

Returns the alphabet used by an MSA object.

Usage

```
alphabet.msa(x)
```

Arguments

x	an MSA object

Value

the valid non-missing-data characters for an MSA object.

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
m <- msa(seqs=c("a--acgtaa", "NN-nnnTAA", "AGGAGGTAG"),
         names=c("human", "mouse", "rat"))
alphabet.msa(m)
```

---

apply.bgc.sel *Apply bgc+selection parameters to a matrix*

Description

Apply bgc+selection parameters to a matrix

Usage

```
apply.bgc.sel(m, bgc = 0, sel = 0, alphabet = "ACGT")
```

Arguments

m	A transition matrix
bgc	The bgc (biased gene conversion) parameter, population-scaled.
sel	The selection parameter (population-scaled)
alphabet	The alphabet used for nucleotide states

Value

A matrix with bgc+sel applied. This matrix may no longer be time reversible.

Author(s)

Melissa J. Hubisz and Adam Siepel

`as.data.frame.feat` *Features to Data Frame*

Description

Convert a features object to a data frame

Usage

```
## S3 method for class 'feat'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

`x`	an object of type `feat`
`row.names`	optional names for each feature
`optional`	logical, if `TRUE`, setting row names and converting column names (to syntactic names: see `make.names` is optional.
`...`	additional arguments to be passed to other methods

Value

a data frame containing features data

Author(s)

Melissa J. Hubisz and Adam Siepel

See Also

`feat` for a description of features data frames, and `as.pointer.feat` for conversion in the other direction.

Examples

```
seq <- rep("hg18.chr6", 10)
src <- rep("fake_example", 10)
feature <- rep("CDS", 10)
start <- seq(1, 100, by=10)
end <- seq(10, 100, by=10)
f1 <- feat(seq, src, feature, start, end, pointer.only=TRUE)
summary(f1)
f2 <- as.data.frame(f1)
summary(f2)
dim(f2)
```

as.list.tm *Tree Model to List*

Description

Coerce a tree model into a list

Usage

```
## S3 method for class 'tm'
as.list(x, ...)
```

Arguments

x	an object of class tm
...	arguments to be passed to/from other functions

Author(s)

Melissa J. Hubisz and Adam Siepel

See Also

tm

Examples

```
tm <- tm(tree="((human:0.01, chimp:0.01):0.03, mouse:0.3)",
         subst.mod="JC69")
is.list(tm)
is.list(as.list(tm))
```

as.pointer.feat *Features To Pointer*

Description

Take a set of features stored in R and return one stored by reference

Usage

```
as.pointer.feat(x)
```

Arguments

x	an object of type feat stored as a data frame in R

Value

an object of type `feat` stored by reference as a pointer to an object created in C.

Author(s)

Melissa J. Hubisz

See Also

`feat` for more details on features storage options.

Examples

```
seq <- rep("hg18.chr6", 10)
src <- rep("fake_example", 10)
feature <- rep("CDS", 10)
start <- seq(1, 100, by=10)
end <- seq(10, 100, by=10)
f1 <- feat(seq, src, feature, start, end)
f2 <- as.pointer.feat(f1)
f1
f2
```

---

as.pointer.msa *MSA To Pointer*

Description

Take an MSA stored in R and return one stored by reference

Usage

```
as.pointer.msa(src)
```

Arguments

`src`	an MSA object stored by value in R

Value

an MSA object stored by reference as a pointer to an object created in C.

Author(s)

Melissa J. Hubisz and Adam Siepel

See Also

`msa` for details on MSA storage options.

Examples

```
m <- msa(seqs=c("A--ACGTAT", "AG-AGGTAA", "AGGAGGTAG"),
         names=c("human", "mouse", "rat"))
m
m <- as.pointer.msa(m)
m
```

`as.track.feat` *Create a features track*

Description

Create a features track

Usage

```
as.track.feat(x, name, short.label = NULL, col = "black", is.gene = FALSE,
  arrow.density = 10)
```

Arguments

`x`	An object of type `feat`
`name`	The name of the track (a character string)
`short.label`	An optional character string to be displayed in left hand margin of track
`col`	The color to use plotting this track (can be a single color or a color for each element)
`is.gene`	A logical value; if `TRUE`, extract and plot gene information from features. The features which will be plotted are the ones with types "CDS", "exon", or "intron". All others will be ignored.
`arrow.density`	(Only used if `is.gene==TRUE`. The number of lines per inch used to denote strand in gene plots.

Value

An object of type `track` which can be plotted with plot.track function

Author(s)

Melissa J. Hubisz

Examples

```
exampleArchive <- system.file("extdata", "examples.zip", package="rphast")
featFile <- "sol1.gp"
unzip(exampleArchive, featFile)
f <- read.feat(featFile)
featTrack <- as.track.feat(f, "basic feature track")
f <- add.introns.feat(f)
geneTrack <- as.track.feat(f, "gene track", is.gene=TRUE)
plot.track(list(featTrack, geneTrack))
plot.track(list(featTrack, geneTrack, geneTrack, geneTrack, geneTrack),
           xlim=c(14800, 16000))
unlink(featFile)
```

---

as.track.msa *Create an alignment track*

Description

Create an alignment track

Usage

```
as.track.msa(x, name, refseq = names.msa(x)[1], short.label = NULL,
  pretty = TRUE, nuc.text = NULL, nuc.text.pos = "bottom",
  nuc.text.col = "black")
```

Arguments

x	An object of type msa
name	The name of the track (a character string)
refseq	A character string identifying the sequence whose coordinate range to use in the plot. A value of NULL implies the frame of reference of the entire alignment.
short.label	An optional character string to be displayed in the left-hand margin of the track
pretty	If TRUE, display bases in the non-reference species which are identical to the reference species as a dot.
nuc.text	If not NULL, can be a vector of character strings. Each character string should be the same length as the MSA with respect to refseq.
nuc.text.pos	If nuc.text is not NULL, can be either "top" or "bottom" to indicate where to place nuc.text relative to the alignment. Will be recycled to the length of nuc.text.
nuc.text.col	If nuc.text is not NULL, color to be used for printing nuc.text. Will be recycled to the length of nuc.text.

Value

An object of type track which can be plotted with the plot.track function

Note

alignment plots will only be displayed if the plot is zoomed in enough to show the alignment data.

Author(s)

Melissa J. Hubisz

See Also

plot.track

---

as.track.wig *Create a wig track*

Description

Create a wig track

Usage

```
as.track.wig(wig = NULL, name, coord = NULL, score = NULL,
  short.label = NULL, col = "black", ylim = NULL, smooth = FALSE,
  numpoints = 250, horiz.line = NULL, horiz.lty = 2,
  horiz.col = "black")
```

Arguments

wig	A "wig" object (Must have elements wig$coord and wig$score which should both be numeric vectors). coord/score may be passed directly instead.
name	The name of the track (a character string)
coord	(Alternative to wig) A numeric vector of coordinates (to be used for x-axis)
score	(Alternative to wig) A numeric vector of scores (y-axis coords), should be same length as coord.
short.label	An optional character string to be displayed in left hand margin of track
col	The color to be used to plot this track.
ylim	The limits to be used on the y-axis. If NULL use entire range of score.
smooth	A logical value indicating whether to perform smoothing when plotting this track
numpoints	(Only used if smooth==TRUE). An integer value indicating how many points to display in the smoothed wig.
horiz.line	If non-NULL, draw horizontal lines on the display at the given y coordinates
horiz.lty	If horiz.line is defined, use this line type.
horiz.col	If horiz.line is defined, use this color

Value

An object of type `track` which can be plotted with the plot.track function

Author(s)

Melissa J. Hubisz

`base.freq.msa` *Get the frequencies of characters in an alignment*

Description

Get the frequencies of characters in an alignment

Usage

```
base.freq.msa(x, seq = NULL, ignore.missing = TRUE, ignore.gaps = TRUE)
```

Arguments

`x`	An object of type `msa`
`seq`	A vector of character strings identifying the sequence(s) to get base frequencies for. If `NULL`, use all sequences.
`ignore.missing`	If TRUE, ignore missing data characters ("N" and "?"). Must be TRUE if seq is stored as a pointer.
`ignore.gaps`	If TRUE, ignore gaps. Must be TRUE if seq is stored as a pointer.

Value

A data frame with one row for each unique state (usually "A", "C", "G", "T", and possibly "N", "?", "-", counts for each state, and overall frequency of each state.

Author(s)

Melissa J. Hubisz

See Also

`statfreq.msa`, which gets observed frequencies of states in an alignment with respect to a substitution model, and works for pointers.

bgc.informative *Return features indicating regions informative for bgc*

Description

Return features indicating regions informative for bgc

Usage

```
bgc.informative(align, foreground, tree, not.informative = FALSE)
```

Arguments

align	An MSA object representing a multiple alignment
foreground	A character string giving the name of a branch (or a label given to several branches) indicating which branch should be in the foreground. The foreground branch is where GC-biased gene conversion is applied, and, if using a coding model, is where a test of positive selection can be performed.
tree	The phylogenetic tree to be used. Can be a newick string describing a tree, or an object of type tm.
not.informative	If TRUE, return the regions that are not informative for bgc.

Value

An object of type feat indicating which regions are informative for bgc on the named foreground branch. If not.informative==TRUE, it will instead return the inverse of this, indicating which regions are not informative for bgc. The coordinates of the features object are in the frame of the reference species of the alignment.

Author(s)

Melissa J. Hubisz

bgc.nucleotide.tests *Do maximum likelihood analysis for gBGC and selection using nucleotide model*

Description

Do maximum likelihood analysis for gBGC and selection using nucleotide model

Usage

```
bgc.nucleotide.tests(align, neutralMod, branch, sel.limits = c(-200, 200),
  bgc.limits = c(0, 200))
```

Arguments

align	A nucleotide alignment of type msa
neutralMod	A model of neutral evolution of type tm. Should be a nucleotide (4x4) model.
branch	A character string giving the name of a branch from neutralMod$tree where lineage-specific selection/gBGC
sel.limits	Numeric vector of length 2 giving lower and upper limits for selection parameter.
bgc.limits	Numeric vector of length 2 giving lower and upper limits for gBGC parameter B

Value

A data.frame with four rows. Each row represents one of the models "null", "sel", "bgc", and "sel+bgc". All models have a global selection coefficient; the sel and sel+bgc models have a lineage-specific selection coefficient as well, and the bgc and sel+bgc models have a lineage-specific gBGC parameter. The likelihoods and parameter estimates for each model are returned in the data frame.

Author(s)

Melissa J. Hubisz

---

bgc.sel.factor *BGC+selection factor*

Description

BGC+selection factor

Usage

```
bgc.sel.factor(x)
```

Arguments

x	The cumulative effect of bgc and selection. If bgc and sel are population-scaled parameters describing biased gene conversion and selection, then x should be sel+bgc for strong mutations, sel-bgc for weak mutations, a and sel for neutral mutations.

Value

x/(1-e^(-x)), the factor to scale the rate matrix entry by.

Author(s)

Melissa J. Hubisz

branchlength.tree *Get the total length of the edges of a tree*

Description

Get the total length of the edges of a tree

Usage

```
branchlength.tree(tree)
```

Arguments

tree	A vector of character strings, each containing a newick tree

Value

A numeric vector containing the total branchlength of each tree

Author(s)

Melissa J. Hubisz and Adam Siepel

classify.muts.bgc *Count the number of mutations of each gBGC type on each branch*

Description

Count the number of mutations of each gBGC type on each branch

Usage

```
classify.muts.bgc(align, mod, branch = NULL)
```

Arguments

align	An alignment of type msa
mod	An evolutionary model of type tm
branch	A character vector giving the name(s) of the branch or branches we are interested in.

Value

A data frame with a row for each branch, giving the number of expected weak (A or T) to strong (C or G) mutations, strong to weak, weak to weak, and strong to strong. I

Author(s)

Melissa J. Hubisz

`codon.clean.msa` *Clean an alignment for codon analysis*

Description

Clean an alignment for codon analysis

Usage

```
codon.clean.msa(x, refseq = NULL, strand = "+")
```

Arguments

`x`	An object of type `msa`
`refseq`	The name of the reference sequence to be used. If given, strip all columns which contain gaps in refseq. Once this is done, alignment should be in frame. If `refseq==NULL` then alignment should be in frame as it is sent in (no gaps are stripped).
`strand`	Either "+" or "-". If "-", reverse complement the alignment.

Value

An object of type `msa`. It will be the same as the original msa, with the following modifications:

- If refseq is not NULL, columns with gaps in refseq will be stripped.
- If strand is "-", the new msa will be the reverse complement of the original.
- After the gap stripping and reverse complementing steps, each sequence is searched for stop codons. If encountered, the stop codon and the rest of the sequence to follow is converted to missing data. The resulting msa has a length equal to the longest remaining sequence (end columns with all missing data are removed).

Note

If the input msa (x) is stored as a pointer, its value will be changed to the return value of the function.

Author(s)

Melissa J. Hubisz

`col.expected.subs.msa` *Obtain expected number of substitutions on each branch for each site pattern and each substitution type*

Description

Obtain expected number of substitutions on each branch for each site pattern and each substitution type

Usage

```
col.expected.subs.msa(x, tm)
```

Arguments

`x`	An object of type `msa`
`tm`	An object of type `tm`

Value

An array giving the expected number of substitutions on each branch, for each distinct alignment column, for each type of substitution.

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
exampleArchive <- system.file("extdata", "examples.zip", package="rphast")
unzip(exampleArchive, "ENr334-100k.maf")
m <- read.msa("ENr334-100k.maf")
mod <- phyloFit(m, tree="((hg18,(mm9,rn4)),canFam2)")
x <- col.expected.subs.msa(sub.msa(m, start.col=41447839, end.col=41448033, refseq="hg18"), mod)
dim(x)
dimnames(x)
x["mm9-rn4","CCTT",,]
unlink("ENr334-100k.maf")
```

`complement` *complement*

Description

DNA complement

Usage

```
complement(x)
```

Arguments

x	A character vector with DNA sequences to be complemented

Value

The complement of the given sequence(s). Characters other than A,C,G,T,a,c,g,t are unchanged.

Author(s)

Melissa J. Hubisz

`composition.feat` *Composition of features with respect to annotations*

Description

Composition of features with respect to annotations

Usage

```
composition.feat(x, annotations)
```

Arguments

x	An object of type `feat`.
annotations	An object of type `feat` containing some annotations.

Value

A data frame with two columns and a row for each type of element in the annotations. The second column gives the fraction of x which fall in the corresponding annotation type. Given non-overlapping annotations which cover the entire range of interest, the second column should sum to 1 (otherwise not).

Note

If x or annotations are passed to this function as pointers to objects stored in C, they will be sorted after the function call.

Author(s)

Melissa J. Hubisz

`concat.msa` *Concatenate msa objects*

Description

If the MSAs do not contain the same set of sequences, the sequences will be added to each MSA and filled with missing data. The order of sequences is taken from the first MSA, and sequences are added to this as necessary.

Usage

```
concat.msa(msas, ordered = FALSE, pointer.only = FALSE)
```

Arguments

`msas`	A list of MSA objects to concatenate together.
`ordered`	If FALSE, disregard the order of columns in the combined MSA.
`pointer.only`	(Advanced use only, for very large MSA objects) If TRUE, return object will be a pointer to an object stored in C.

Value

An object of type MSA

Note

None of the msas passed to this function will be altered, even if they are stored as pointers to objects in C.

Author(s)

Melissa J. Hubisz and Adam Siepel

`convert.coords.feat` *Convert coordinates from one frame of reference to another*

Description

Converts coordinates of features in a GFF according to a multiple alignment. Will map from the coordinate system of any sequence to any other sequence; can also map to or from the coordinate system of the entire alignment.

Usage

```
convert.coords.feat(x, align, from = -1, to = 1)
```

Arguments

`x`	A features object; if the from parameter is -1, the first column should indicate which frame of reference is used for that row (ie, the species name). For coordinates in the frame of reference of the entire alignment, set the first column to "MSA".
`align`	An msa object containing the alignment
`from`	A single character string or integer, used to indicate the current frame of reference of the features. If the rows of the features are not all in the same frame of reference, set this to -1 and indicate the frame of reference in the 1st column of the features. Otherwise, it can be specified here as a single integer (from 0 to nrow.msa(align)), with 0 indicating the frame of reference of the entire alignment, and 1 indicating the 1st species, 2 the second, etc. Or it can be a single character string giving the name of the species, with "MSA" indicating the entire alignment.
`to`	A single character string or integer, used to indicate the frame of reference to convert to. This is specified in the same way as the "from" argument, above, except that -1 is not an option.

Value

A features object with elements in the frame of refernece indicated by the "to" argument.

Note

Ignores any offset in MSA. All coordinates should start with the first position in the alignment as 1.

If the endpoints of an element have gaps in the "to" species, the elements will be truncated

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
require("rphast")
align <- msa(seqs=c("A-GTAT", "-GGTAA", "AG--AG"),
             names=c("human", "mouse", "rat"))
feats <- feat(seqname=c("MSA", "human", "human", "mouse", "mouse", "rat"),
              start=c(1, 2, 3, 1, 3, 3),
              end=c(6, 4, 4, 4, 5, 3))
convert.coords.feat(feats, align)  # convert everything to human coords
convert.coords.feat(feats, align, to="MSA")  # convert to alignment coords

# here, there is no position 6 in human alignment so feature is removed
convert.coords.feat(feat(seqname="human", start=6, end=6),
                    align, to="MSA")

# here, feature goes beyond end of MSA so it is truncated:
convert.coords.feat(feat(seqname="rat", start=2, end=100),
                    align, to="MSA")


# note that if the "to" species has gaps at the endpoints, they will
# be truncated:
align <- msa(seqs=c("A-GT-T", "ACGTGT"), names=c("human", "mouse"))
convert.coords.feat(feat(seqname="mouse", start=2, end=5),
                    align, to="human")
```

`coord.range.msa` *Obtain the range of coordinates in a MSA objects*

Description

Obtain the range of coordinates in a MSA objects

Usage

```
coord.range.msa(x, refseq = names.msa(x)[1])
```

Arguments

`x`	An object of type `msa`
`refseq`	A character string identifying the reference sequence (or NULL to use frame of reference of entire alignment)

Value

A numeric vector of length 2 giving the smallest and highest coordinate in the alignment. If refseq is the first sequence in alignment, offset.msa(x) is added to the range, otherwise it is ignored.

Author(s)

Melissa J. Hubisz and Adam Siepel

copy.feat *Features copy*

Description

Creates a copy of a features object

Usage

```
copy.feat(x)
```

Arguments

x	an object of type feat

Details

If x is stored in R (as it is by default), then this is no different than x2 <- x. But if it is stored as a pointer to a structure in C, then this is the only way to make an explicity copy of the features.

Value

a features object which can be modified independently from the original object

Author(s)

Melissa J. Hubisz and Adam Siepel

copy.msa *MSA copy*

Description

Creates a copy of an MSA sequence

Usage

```
copy.msa(x)
```

Arguments

x	an MSA object

Details

If m is stored in R (as it is by default), then m2 <- copy.msa(m1) is no different than m2 <- m1. But if it is stored as a pointer to a C structure, this is the only way to make an explicit copy of the MSA object.

Value

an MSA object which can be modified independently from the original object

Author(s)

Melissa J. Hubisz and Adam Siepel

---

coverage.feat *Features coverage*

Description

Features coverage

Usage

```
coverage.feat(..., or = FALSE, not = FALSE, get.feats = FALSE,
  pointer.only = FALSE)
```

Arguments

...	objects of type feat
or	if TRUE, get the coverage of union of feat arguments. or is FALSE by default, which takes the intersection.
not	If not NULL, a vector of logicals the same length as the number of features provided (or will be recycled to this length). For each value which is TRUE, then any base *not* included in this feature will be counted. (The negation is done before any other operation). If NULL, do not negate any features. There must be at least one feature which is not negated (so that boundaries can be established).
get.feats	if TRUE, return an object of type feat representing the intersection (or union of or==TRUE) of the features
pointer.only	(Only used if get.feats==TRUE). If TRUE, the features object returned will be stored as a pointer to an object in C.

Value

The number of bases covered by the feat arguments, or the combined feat object if get.feats==TRUE.

Note

Any features object passed into this function which is stored as a pointer to an object stored in C may be reordered (sorted) by this function.

Author(s)

Melissa J. Hubisz

Examples

```
feat1 <- feat(seqname=c(rep("chr1", 3), rep("chr2", 2)),
              start=c(1, 5, 100, 10, 20),
              end=c(7, 10, 105, 15, 30))
feat2 <- feat(seqname=c("chr1","chr2"),
              start=c(1,1), end=c(5,10))
coverage.feat(feat1, feat2, or=FALSE)
coverage.feat(feat1, feat2, or=TRUE)
coverage.feat(feat1, feat2, get.feats=TRUE, or=TRUE)
coverage.feat(feat1, feat2, or=TRUE)
rm(feat1, feat2)
```

---

density.feat *Features kernel density*

Description

Features kernel density

Usage

```
## S3 method for class 'feat'
density(x, type = "length", ...)
```

Arguments

x	An object of type feat
type	a character string, denoting the value to compute the density for. Currently the only valid types are "length" and "score"
...	additional arguments to be passed to density

Value

A kernel density object as defined by density

Author(s)

Melissa J. Hubisz

depth.tree *Get the distance from a node to the root of a tree*

Description

Get the distance from a node to the root of a tree

Usage

```
depth.tree(tree, node)
```

Arguments

tree	A vector of character strings, each containing a newick tree
node	A vector of character strings, giving the node name to use for each tree. Will be recycled to the length of the first argument.

Value

A numeric vector containing the distance from each given node to the root of the corresponding tree.

Author(s)

Melissa J. Hubisz and Adam Siepel

dim.feat *Feature dimensions*

Description

Get the dimensions of a features object

Usage

```
## S3 method for class 'feat'
dim(x)
```

Arguments

x	an object of type feat

Value

An integer vector of length two containing the number of rows and number of columns in the features object.

Author(s)

Melissa J. Hubisz

Examples

```
seq <- rep("hg18.chr6", 10)
src <- rep("fake_example", 10)
feature <- rep("CDS", 10)
start <- seq(1, 100, by=10)
end <- seq(10, 100, by=10)
f1 <- feat(seq, src, feature, start, end)
dim(f1)
dim.feat(f1)
f2 <- as.pointer.feat(f1)
dim(f2)
dim.feat(f2)
```

---

dim.msa *Returns the dimensions of an msa object as (# of species, # of columns)*

Description

Returns the dimensions of an msa object as (# of species, # of columns)

Usage

```
## S3 method for class 'msa'
dim(x)
```

Arguments

x	An object of type msa

Value

An integer vector of length two giving number of species and number of columns in the alignment

Author(s)

Melissa J. Hubisz and Adam Siepel

enrichment.feat *Enrichment of features with respect to annotation types*

Description

Enrichment of features with respect to annotation types

Usage

```
enrichment.feat(x, annotations, region.bounds)
```

Arguments

x	An object of type feat
annotations	An object of type feat containing some annotations.
region.bounds	An object of type feat representing the boundary coordinates of the regions of interest (such as chromosome boundaries). All elements from the first two arguments should fall entirely within region.bounds.

Value

A data frame with two columns and a row for each type of element in the annotations. The second column gives the fold-enrichment of x across the corresponding annotation type, which is equal to the fraction of x which fall within the annotation type, divided by the fraction of the entire region covered by the annotation type.

Note

If any of the arguments to this function are passed as pointers to objects stored in C, they will be sorted after this function call.

Author(s)

Melissa J. Hubisz

expected.subs.msa *Obtain expected number of substitutions on each branch and site*

Description

Obtain expected number of substitutions on each branch and site

Usage

```
expected.subs.msa(x, tm)
```

Arguments

x	An object of type msa
tm	An object of type tm

Value

An array giving the expected number of substitutions on each branch at each unique site pattern, summed across all types of substitutions.

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
exampleArchive <- system.file("extdata", "examples.zip", package="rphast")
unzip(exampleArchive, "ENr334-100k.maf")
m <- read.msa("ENr334-100k.maf")
mod <- phyloFit(m, tree="((hg18,(mm9,rn4)),canFam2)")
x <- expected.subs.msa(sub.msa(m, start.col=41447839, end.col=41448033, refseq="hg18"), mod)
dim(x)
dimnames(x)
x[,"CCCC"]
x["mm9-rn4",]
unlink("ENr334-100k.maf")
```

---

extract.feature.msa *Extract features from an MSA object*

Description

Returns the subset of the MSA which appears in the features object.

Usage

```
extract.feature.msa(x, features, do4d = FALSE, pointer.only = FALSE)
```

Arguments

x	An object of type MSA
features	An object of type features denoting the regions of the alignment to extract.
do4d	If TRUE, then some elements of features must have type "CDS", and only fourfold-degenerate sites will be extracted.
pointer.only	If TRUE, return only a pointer to an object stored in C (useful for large alignments; advanced use only)

Value

An msa object containing only the regions of x appearing in the features object.

Note

If x was loaded with `pointer.only==TRUE`, then x will be modified to the return value of the function. Use `extract.feature.msa(copy.msa(x), features,...)` if you don't want this behavior!

Author(s)

Melissa J. Hubisz and Adam Siepel

See Also

`sub.msa`, `[.msa`

---

feat *Features Objects*

Description

Create a new features object

Usage

```
feat(seqname = "default", src = ".", feature = ".", start, end,
  score = NULL, strand = NULL, frame = NULL, attribute = NULL,
  pointer.only = FALSE)
```

Arguments

`seqname`	a character vector containing the name of the sequence. If the features correspond to regions of an alignment, then seqname should be the name of the sequence in the alignment that is used as the frame of reference in the features. To use the entire alignment as a frame of reference, set seqname to "MSA".
`src`	The source of the feature
`feature`	The feature type name
`start`	The start of the feature. Sequence numbering begins at 1.
`end`	The end of the feature. This is the last coordinate included in the feature.
`score`	The feature score, or NA if there is no score.
`strand`	A character string which is either "+", "-", or "." (if strand is not available or relevant).
`frame`	A 0, 1, or 2, which specifies whether the feature is in frame.
`attribute`	A feature attribute (character string).
`pointer.only`	Whether to store object as a pointer to an object in C, rather than as a data.frame in R.

Details

See http://www.sanger.ac.uk/resources/software/gff/spec.html for more detailed description of each parameter.

All arguments which are provided should be vectors of equal length.

If pointer.only==FALSE, the new object is a data frame, with columns mirroring the GFF Specification Otherwise, it is a list containing a single element, which is a pointer to an object stored in C.

Value

If pointer.only==FALSE, returns a data.frame whose format mirrors the GFF specification. Otherwise, returns a list with a single object, which is a external pointer to a C structure representing a features object.

Author(s)

Melissa J. Hubisz and Adam Siepel

See Also

read.feat

msa for more details on the pointer.only option.

Examples

```
seq <- rep("hg18.chr6", 10)
src <- rep("fake_example", 10)
feature <- rep("CDS", 10)
start <- seq(1, 100, by=10)
end <- seq(10, 100, by=10)
f <- feat(seq, src, feature, start, end)
dim(f)
dim.feat(f)
f <- feat(seq, src, feature, start, end, pointer.only=TRUE)
dim.feat(f)
```

---

fix.semicolon.tree *Add a semi-colon to end of tree string*

Description

Check if tree string ends in semi-colon and if not add one. This is mostly done for compatibility with ape, which requires them.

Usage

```
fix.semicolon.tree(x)
```

Arguments

x	A character string or vector of character strings each representing a tree in Newick format.

Value

The same value, but with a semi-colon added to the end of any strings which did not already end in semi-colons.

Author(s)

Melissa J. Hubisz

Examples

```
str <- c("213", "345")
fix.semicolon.tree(str)
str <- c("213;", "345;")
fix.semicolon.tree(str)
str <- c("213", "345;")
fix.semicolon.tree(str)
```

---

fix.start.stop.feat *Fix start and stop signals*

Description

Fix start and stop signals

Usage

```
fix.start.stop.feat(x)
```

Arguments

x	An object of type `feat`. CDS regions must be present with type "CDS", and the transcript_id must be indicated in the attribute field. Start and stop codons should have feature type "start_codon" and "stop_codon" (as produced by addSignals.feat).

Value

An object of type `feat`, in which CDS regions are ensured to include start codons and exclude stop codons, as required by the GTF2 standard.

Note

- If x is stored as a pointer to an object stored in C, signals will be added to x.
- Assumes at most one start_codon and at most one stop_codon per transcript.

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
exampleArchive <- system.file("extdata", "examples.zip", package="rphast")
featFile <- "gencode.ENr334.gp"
unzip(exampleArchive, featFile)
f <- read.feat(featFile)
f <- add.signals.feat(f)
# let's just look at one gene
geneNames <- tagval.feat(f, "transcript_id")
f <- f[geneNames==geneNames[1],]

# This features file already is correct, so let's mess it up to see
# how fix.start.stop can fix it:

#modify first CDS to not include start
startCodon <- f[f$feature=="start_codon",]
firstCds <- which(f$feature=="CDS" & f$start==startCodon$start)
f[firstCds,]$start <- startCodon$end+1
#modify last CDS to include stop
stopCodon <- f[f$feature=="stop_codon",]
lastCds <- which(f$feature=="CDS" & f$end+1==stopCodon$start)
f[lastCds,]$end <- stopCodon$end
# now call fix.start.stop to fix
f.fixed <- fix.start.stop.feat(f)

# first CDS has been fixed to include start codon
f[firstCds,]
f.fixed[firstCds,]
# last CDS has been fixed to not include stop codon
f[lastCds,]
f.fixed[lastCds,]

unlink(featFile)
```

---

flatten.feat *Combine adjacent features with the same "feature" field*

Description

Combine adjacent features with the same "feature" field

Usage

```
flatten.feat(x, weightedAverageScore = FALSE, minScore = FALSE)
```

Arguments

x	An object of type feat
weightedAverageScore	If TRUE, scores of merged features are given by the average of those features, weighted by their original lengths. Otherwise, scores of merged features are simply summed.
minScore	If TRUE, scores of merged features are given by the minimum score of the combined features.

Value

A features object in which adjacent features are combined into one longer feature.

Note

If x is stored as a pointer to a C structure, then x will be modified to the return value.

Author(s)

Melissa J. Hubisz and Adam Siepel

---

freq3x4.msa *Get codon frequencies based on 3x4 model*

Description

Get codon frequencies based on 3x4 model

Usage

```
freq3x4.msa(x)
```

Arguments

x	An object of type msa. It is assumed to represent in-frame codons. Length should be a multiple of 3.

Value

A vector of length 64 corresponding to the 64 codon frequencies. The frequencies corresponding to stop codons should be 0.

Author(s)

Melissa J. Hubisz

from.pointer.feat *Convert a features object from C memory (external pointer) to R memory*

Description

Convert a features object from C memory (external pointer) to R memory

Usage

```
from.pointer.feat(x)
```

Arguments

x	A features object stored as a pointer to C memory

Value

A features object (inheriting from the data.frame class) stored in R memory

Author(s)

Melissa J. Hubisz

from.pointer.msa *MSA From Pointer*

Description

Take an MSA stored by reference and return one stored in R

Usage

```
from.pointer.msa(src)
```

Arguments

src	an MSA object stored by reference

Value

an MSA object stored in R. If src is already stored in R, returns a copy of the object.

Author(s)

Melissa J. Hubisz and Adam Siepel

See Also

msa for details on MSA storage options.

Examples

```
m <- msa(seqs=c("A--ACGTAT", "AG-AGGTAA", "AGGAGGTAG"),
         names=c("human", "mouse", "rat"), pointer.only=TRUE)
m
m <- from.pointer.msa(m)
m
```

---

gc.content.msa *Get the fraction of G's and C's in an alignment*

Description

Get the fraction of G's and C's in an alignment

Usage

```
gc.content.msa(x, seq = NULL, ignore.missing = TRUE, ignore.gaps = TRUE)
```

Arguments

x	An object of type msa
seq	A vector of character strings identifying the sequence(s) to use in the base frequency tabulation. If NULL, use all sequences.
ignore.missing	If FALSE, count missing data in the denominator.
ignore.gaps	If TRUE, count gaps in the denominator.

Value

The fraction of bases which are C's and G's

Author(s)

Melissa J. Hubisz

get.rate.matrix.params.tm *Get the parameters describing a rate matrix*

Description

Get the parameters describing a rate matrix

Usage

```
get.rate.matrix.params.tm(x)
```

Arguments

x	An object of type tm.

Value

A numeric vector of parameters which can be used to describe the transition matrix under the substitution model indicated in x. May be NULL for certain models which have no parameters (JC69, F81). The meaning of the parameters is described in set.rate.matrix.tm.

Note

The params returned may not describe the rate matrix passed in, if the rate matrix does not follow the model indicated in x.

Author(s)

Melissa J. Hubisz and Adam Siepel

get4d.msa *Extract fourfold degenerate sites from an MSA object*

Description

Extract fourfold degenerate sites from an MSA object

Usage

```
get4d.msa(x, features)
```

Arguments

x	An object of type msa
features	an object of type feat. Should have defined coding regions with feature type "CDS"

Value

An unordered msa object containing only the sites which are fourfold degenerate.

Note

If x is stored as a pointer, it will be reduced to four-fold degenerate sites, so the original alignment will be lost. Use get4d.msa(copy.msa(x), features) to avoid this behavior. The return value will always be stored in R regardless of how the original alignment was stored.

For very large MSA objects it is more efficient to use the do.4d option in the read.msa function instead.

Author(s)

- Melissa J. Hubisz and Adam Siepel

Examples

```
exampleArchive <- system.file("extdata", "examples.zip", package="rphast")
files <- c("ENr334-100k.maf", "ENr334-100k.fa", "gencode.ENr334-100k.gff")
unzip(exampleArchive, files)
f <- read.feat("gencode.ENr334-100k.gff")
f$seqname <- "hg18.chr6"
m1 <- read.msa("ENr334-100k.maf", features=f, do.4d=TRUE)
m2 <- read.msa("ENr334-100k.maf")
m3 <- get4d.msa(m2, features=f)
m4 <- get4d.msa(read.msa("ENr334-100k.maf"), features=f)
m5 <- get4d.msa(read.msa("ENr334-100k.fa", offset=41405894), features=f)
unlink(files)
```

---

guess.format.msa *MSA Guess Format*

Description

Guess the format of an MSA file by looking at the file contents.

Usage

```
guess.format.msa(filename, method = "content")
```

Arguments

filename	A vector of file names
method	Either "content" or "extension". "content" implies to open the file and guess the format based on content; "extension" simply guesses based on the extension on the file name (it does not open the file). This argument will be recycled to the length of filename.

Value

A character vector giving the format of each file (one of "MAF", "FASTA", "LAV", "SS", "PHYLIP", "MPM", or "UNKNOWN").

Author(s)

Melissa J. Hubisz

See Also

is.format.msa

Examples

```
exampleArchive <- system.file("extdata", "examples.zip", package="rphast")
files <- c("ENr334-100k.maf", "ENr334-100k.fa", "ENr334-100k.ss", "sol1.maf", "sol1.gp")
unzip(exampleArchive, files=files)
guess.format.msa(files)
# the last file is not an alignment, which is why it returns UNKNOWN
unlink(files)
```

---

hist.feat *plot histogram of feature lengths*

Description

plot histogram of feature lengths

Usage

```
## S3 method for class 'feat'
hist(x, type = "length", ...)
```

Arguments

x	an object of type feat
type	a character string, denoting the value to make the histogram with. Currently the only valid types are "length" or "score"
...	additional arguments to be passed to hist

Author(s)

Melissa J. Hubisz

hmm *Create an rphast HMM object*

Description

Create a new HMM object

Usage

```
hmm(trans.mat, eq.freq = NULL, begin.freq = NULL, end.freq = NULL)
```

Arguments

trans.mat	A square matrix object of dimension n x n where n is the number of states, and element [i,j] is the rate of transition from state i to state j
eq.freq	A vector of length n giving the equilibrium frequencies of each state. If NULL, calculate equilibrium frequencies that will make a reversible markov chain.
begin.freq	A vector of length n giving the initial state frequencies. If NULL, use equilibrium frequencies.
end.freq	A vector of length n giving the final state frequencies. If NULL, do not condition on end frequencies.

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
h <- hmm(matrix(1, nrow=4, ncol=4))
h
```

informative.regions.msa *Get informative regions of an alignment*

Description

Get informative regions of an alignment

Usage

```
informative.regions.msa(x, min.numspec, spec = NULL,
  refseq = names.msa(x)[1], gaps.inf = FALSE)
```

Arguments

x	An object of type msa.
min.numspec	The minimum number of species with non-missing data required for an alignment column to be considered informative.
spec	A character vector of species names, or an integer vector of species indices. Only data in the named species count towards deciding if a site is informative. The default value of NULL implies use all species in the alignment.
refseq	Defines the frame of reference for the return value. Should be a character vector with the name of one of the sequences in the alignment, or NULL to indicate use the frame of reference of the entire alignment.
gaps.inf	Logical value indicating whether a gap should be considered informative. The default value of FALSE indicates that gaps as well as missing data are not counted as informative.

Value

An object of type feat indicating the regions of the alignment which meet the informative criteria. Note that unless refseq==NULL, columns with gaps in the reference sequence will be ignored, and will fall in "informative" or "uninformative" features based on the informativeness of neighboring columns.

Note

- If the msa object has an idx.offset, it is assumed to be a coordinate offset for the first species in the alignment. So the idx.offset will be added to the coordinates in the returned features object only if refseq==names.msa(x)[1].
- This function will not alter the value of x even if it is stored as a pointer.

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
require("rphast")
m <- msa(seqs=c("A--ACGTAT-", "AG-AGGTAA-", "AGGAGGTA--"),
         names=c("human", "mouse", "rat"))
informative.regions.msa(m, 1, refseq=NULL)
informative.regions.msa(m, 3, refseq=NULL)
informative.regions.msa(m, 3, refseq="mouse", spec=c("mouse", "rat"))
```

inverse.feat *Get inverse features*

Description

Get inverse features

Usage

```
inverse.feat(x, region.bounds, pointer.only = FALSE)
```

Arguments

x	An object of type feat
region.bounds	An object of type feat which defines the boundaries of all relevant chromosomes in the first argument
pointer.only	If TRUE, return a pointer to a structure stored in C (advanced use only).

Value

An object of type feat which contains all regions in region.bounds that are not in the first argument

Note

If x is stored as a pointer to C memory, then its elements will be sorted by this function. region.bounds will not be changed.

Author(s)

Melissa J. Hubisz

is.format.msa *Check an MSA Format String*

Description

Returns TRUE if the argument is a valid string describing a multiple sequence alignment (MSA) format.

Usage

```
is.format.msa(format)
```

Arguments

format	a character vector of strings to test

Details

Valid formats include "FASTA", "PHYLIP", "SS" (Sufficient statistics format used by PHAST), "MPM" (format used by MultiPipMaker), "LAV" (used by blastz), or "MAF" (Multiple Alignment Format used by MULTIZ and TBA.

Value

a logical vector indicating whether each element of the input parameter is a valid format string.

Author(s)

Melissa J. Hubisz

Examples

```
is.format.msa(c("MAF", "SS", "PHYLIP", "MPM", "LAV", "FASTA",
                "BAD_FORMAT_STRING"))
```

---

is.msa *Check an MSA object*

Description

Check an MSA object

Usage

```
is.msa(msa)
```

Arguments

msa	An object to tests

Value

A logical indicating whether object is of type msa

Author(s)

Melissa J. Hubisz

is.ordered.msa *MSA is Ordered?*

Description

Determines if an MSA object represents an ordered alignment.

Usage

```
## S3 method for class 'msa'
is.ordered(x)
```

Arguments

x	an MSA object

Value

a boolean indicating whether the columns are in order

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
m <- msa(seqs=c("A--ACGTAT", "AG-AGGTAA", "AGGAGGTAG"),
         names=c("human", "mouse", "rat"))
is.ordered.msa(m)
m <- msa(seqs=c("A--ACGTAT", "AG-AGGTAA", "AGGAGGTAG"),
         names=c("human", "mouse", "rat"), is.ordered=FALSE)
is.ordered.msa(m)
```

is.subst.mod.tm *Check Substitution Model Strings*

Description

Check if a string represents a phast substitution model

Usage

```
is.subst.mod.tm(mod)
```

Arguments

mod	A vector of character strings representing substitution model names to be tested

Value

A vector of logical values indicating whether each string represents a defined substitution model

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
is.subst.mod.tm(c("JC69", "K80", "F81", "HKY85", "HKY85+Gap",
                  "REV", "SSREV", "REV+GC", "UNREST", "R2", "U2", "R2S",
                  "U2S", "R3", "R3S", "U3", "U3S", "GC", "HB",
                  "bad.model"))
```

`is.tm` *Tree Models*

Description

Check whether an object is of type tm (tree model)

Usage

```
is.tm(x)
```

Arguments

`x`	an object to be tested

Value

TRUE if an object has class "tm", FALSE otherwise

Author(s)

Melissa J. Hubisz

is.track *Is this a track?*

Description

Is this a track?

Usage

```
is.track(x, ...)
```

Arguments

x	An object to test
...	ignored

Value

A logical indicating whether x is an object of type track

Author(s)

Melissa J. Hubisz

label.branches *Label tree branches*

Description

Apply a label to some branches

Usage

```
label.branches(tree, branches, label)
```

Arguments

tree	A vector of character strings, each containing a newick tree
branches	A vector of character strings, indicating the branches which should get the label. The branch is named by the node which descends from it. If multiple trees and branches are given, all named branches will be labelled in every tree.
label	A single character string giving the label to apply to the named branches.

Value

A vector of character strings containing the modified trees; the branches are labelled by appending a pound sign and the label to the node name in the tree string.

Author(s)

Melissa J. Hubisz

Examples

```
trees <- c("((hg18:1.0, panTro2:2.0)hg18-panTro2:3.0, mm9:4.0);",
           "((hg18:0.142679,(mm9:0.083220,rn4:0.090564)mm9-rn4:
             0.269385)hg18-rn4:0.020666,canFam2:0.193569);")
label.branches(trees, c("hg18", "mm9"), "humanAndMouse")
```

---

label.subtree *Label subtree*

Description

Apply a label to a subtree

Usage

```
label.subtree(tree, node, label, include.leading = FALSE)
```

Arguments

tree	A vector of character strings, each containing a newick tree
node	A character string, giving the node at the head of the subtree.
label	A single character string giving the label to apply to the branches in the subtree.
include.leading	A logical value; if TRUE, include the branch leading to the subtree in the labelled group; otherwise include only descendants of the named node.

Value

A vector of character strings containing the modified trees; the branches are labelled by appending a pound sign and the label to the node name in the tree string.

Author(s)

Melissa J. Hubisz

Examples

```
trees <- c("((hg18:1.0, panTro2:2.0)hg18-panTro2:3.0, mm9:4.0);",
           "(((hg18:0.01, panTro2:0.01)hg18-panTro2:0.07,
              (mm9:0.083220,rn4:0.090564)mm9-rn4:
             0.269385)hg18-rn4:0.020666,canFam2:0.193569);")
label.subtree(trees, "hg18-panTro2", "human-chimp", include.leading=FALSE)
label.subtree(trees, "hg18-panTro2", "human-chimp", include.leading=TRUE)
```

---

leafnames.tree *Get the names of a tree's leaf nodes*

Description

Get the names of a tree's leaf nodes

Usage

```
leafnames.tree(object, ...)
```

Arguments

object	A character string containing a newick tree
...	Not currently used

Value

A character vector containing the names of the leaf nodes

Author(s)

Melissa J. Hubisz and Adam Siepel

---

likelihood.msa *MSA Likelihood*

Description

Likelihood of an alignment given a tree model

Usage

```
likelihood.msa(x, tm, features = NULL, by.column = FALSE)
```

Arguments

x	An object of class msa representing the multiple alignment
tm	An object of class tm representing the tree and model of substitution
features	A features object. If non-null, compute likelihoods for each feature rather than the whole alignment.
by.column	Logical value. If TRUE, return the log likelihood for each alignment column rather than total log likelihood. Ignored if features is not NULL.

Value

Either the log likelihood of the entire alignment (if by.column==FALSE && is.null(features), or a numeric vector giving the log likelihood of each feature (if !is.null(features)), or a numeric vector giving the log likelihood of each column (if by.column==TRUE).

Author(s)

Melissa J. Hubisz and Adam Siepel

See Also

phyloFit, tm

Examples

```
files <- c("rev.mod", "ENr334-100k.maf", "ENr334-100k.fa", "small.gff")
exampleArchive <- system.file("extdata", "examples.zip", package="rphast")
unzip(exampleArchive, files)
msa <- read.msa("ENr334-100k.fa")
mod <- read.tm("rev.mod")
likelihood.msa(msa, mod)
like1 <- likelihood.msa(msa, mod, by.column=TRUE)
length(like1)==ncol.msa(msa)
sum(like1)
msa <- read.msa("ENr334-100k.maf")
likelihood.msa(msa, mod)
like2 <- likelihood.msa(msa, mod, by.column=TRUE)
sum(like2)
mod$subst.mod <- "JC69"
likelihood.msa(msa, mod)
#'
# can also get likelihood by feature
features <- read.feat("small.gff")
features$seqname <- names(msa)[1]
likelihood.msa(msa, mod, features=features)
unlink(files)
```

mod.backgd.tm *Adjust tree model background frequencies while maintaining reversibility*

Description

Adjust tree model background frequencies while maintaining reversibility

Usage

```
mod.backgd.tm(tm, new.backgd = NULL, gc = NULL)
```

Arguments

tm	An object of type tm
new.backgd	A numeric vector of length 4 giving the background frequencies of A,C,G,T
gc	(Alternative to new.backgd) A numeric value giving the GC content, which is used to calculate new background frequencies. Assumes freq(C)==freq(G)==gc/2, and freq(A)==freq(T)==(1-gc)/2

Note

Currently only works with models of order 0, without lineage- specific models, and which use the default alphabet "ACGT".

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
exampleArchive <- system.file("extdata", "examples.zip", package="rphast")
filename <- "rev.mod"
unzip(exampleArchive, filename)
tm <- read.tm(filename)

# change background frequencies to new value, adjusting rate matrix
mod.backgd.tm(tm, c(0.25, 0.25, 0.25, 0.25))

# change background frequencies so that GC content is 0.6
mod.backgd.tm(tm, gc=0.6)

unlink(filename)
```

`msa` *MSA Objects*

Description

Creates a new MSA object given sequences.

Usage

```
msa(seqs, names = NULL, alphabet = "ACGT", is.ordered = TRUE,
  offset = NULL, pointer.only = FALSE)
```

Arguments

`seqs`	a character vector containing sequences, one per sample
`names`	a character vector identifying the sample name for each sequence. If `NULL`, use "seq1", "seq2", ...
`alphabet`	a character string containing valid non-missing character states
`is.ordered`	a logical indicating whether the alignment columns are stored in order. If NULL, assume columns are ordered.
`offset`	an integer giving the offset of coordinates for the reference sequence from the beginning of the chromosome. The reference sequence is assumed to be the first sequence. Not used if is.ordered==FALSE.
`pointer.only`	a boolean indicating whether returned alignment object should be stored by reference (see Details)

Details

Make a new multiple sequence alignment (MSA) object given a vector of character strings. They can be optionally annotated with sample names.

Each character string in seqs must be the same length, and number of elements in names (if provided) must match the number of elements in seqs.

Alphabet generally does not have to be specified if working with DNA alignments.

About storing objects as pointers: If `pointer.only==FALSE`, the MSA object will be stored in R and can be viewed and modified by base R code as well as RPHAST functions. Setting `pointer.only=TRUE` will cause the object to be stored by reference, as an external pointer to an object created by C code. This may be necessary to improve performance, but the object can then only be viewed/manipulated via RPHAST functions. Furthermore, if an object is stored as a pointer, then its value is liable to be changed when passed as an argument to a function. All RPHAST functions which change the value of an external pointer make a note of this in the help pages for that function. For example, extract.feature.msa will alter an alignment if it is passed in as an external pointer (the argument will be changed into the return value). If this is undesireable, the copy.msa function can be used: extract.feature.msa(copy.msa(align)) will preserve the original alignment. Simple copying, ie, `align2->align1` of objects stored as pointer will not behave like normal R objects: both objects will point to the same C structure, and both will be changed if either one is altered. Instead `align2 <- copy.msa(align1)` should be used.

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
# Here is an MSA object stored in the default mode
m1 <- msa(seqs=c("ACGTAT", "AGGTAA", "AGGTAG"),
          names=c("human", "mouse", "rat"))
m2 <- m1
# NOTE seqs would not be directly accessible if stored by reference
m2$seqs[3] <- "AAAAAA"
print(m1)
print(m1, print.seq=TRUE)
print(m2, print.seq=TRUE)
```

---

name.ancestors *Name Ancestral Nodes*

Description

Name ancestors of a tree

Usage

```
name.ancestors(tree)
```

Arguments

tree	A vector of character strings, each containing a newick tree

Value

A vector of character strings containing newick trees with all ancestors named.

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
trees <- c("((hg18, panTro2), mm9);",
           "((hg18:0.142679,(mm9:0.083220,rn4:0.090564):0.269385):
                0.020666,canFam2:0.193569);")
name.ancestors(trees)
```

names.msa *MSA Sequence Names*

Description

Returns the sequence names for an MSA object.

Usage

```
## S3 method for class 'msa'
names(x)
```

Arguments

x	an MSA object

Value

a character vector giving the names of the sequences, or NULL if they are not defined

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
m <- msa(seqs=c("ACGTAT", "AGGTAA", "AGGTAG"),
         names=c("human", "mouse", "rat"))
names.msa(m)
m <- msa(seqs=c("ACGTAT", "AGGTAA", "AGGTAG"))
names.msa(m)
```

ncol.feat *Number of Columns in Features*

Description

Get the number of columns in a features object

Usage

```
## S3 method for class 'feat'
ncol(x)
```

Arguments

x	An object of type feat

Value

An integer containing the number of columns in the features object

Note

If the features object is stored as a pointer in C, the number of columns is always 9.

Author(s)

Melissa J. Hubisz

Examples

```
seq <- rep("hg18.chr6", 10)
src <- rep("fake_example", 10)
feature <- rep("CDS", 10)
start <- seq(1, 100, by=10)
end <- seq(10, 100, by=10)
f <- feat(seq, src, feature, start, end)
ncol.feat(f)
ncol.feat(as.pointer.feat(f))
```

---

ncol.msa *MSA Sequence Length.*

Description

Returns the length of sequence in an MSA alignment.

Usage

```
## S3 method for class 'msa'
ncol(x, refseq = NULL)
```

Arguments

x	an MSA object
refseq	character vector giving name(s) of sequence whose length to return. The default NULL implies the frame of reference of the entire alignment.

Value

an integer vector containing the length of the named sequences. If refseq is NULL, returns the number of columns in the alignment.

Author(s)

Melissa J. Hubisz and Adam Siepel

See Also

msa

Examples

```
m <- msa(seqs=c("A--ACGTAT", "AG-AGGTAA", "AGGAGGTAG"),
         names=c("human", "mouse", "rat"))
ncol.msa(m)
ncol.msa(m, names.msa(m))
```

---

ninf.msa *The number of informative columns in an alignment*

Description

The number of informative columns in an alignment

Usage

```
ninf.msa(x)
```

Arguments

x	An object of type msa

Value

The number of "informative" columns in the msa. An informative column has at least two non-missing and non-gap characters.

Author(s)

Melissa J. Hubisz and Adam Siepel

See Also

pairwise.diff.msa To get differences per base between pairs of sequences

nothanks.rphast *Stop rphast registration reminders*

Description

Once this is called rphast will no longer produce startup messages prodding you to register.

Usage

```
nothanks.rphast()
```

Note

This creates an empty file called "notRegistered" in `Sys.getenv("rphastRegDir")`. The rphastRegDir is usually in %appdata%\rphast for Windows systems and ~/.rphast for other systems.

Author(s)

Melissa J. Hubisz

nrow.feat *Number of Features*

Description

Get the number of rows in a features object

Usage

```
## S3 method for class 'feat'
nrow(x)
```

Arguments

`x`	An object of type `feat`

Value

An integer containing the number of rows in each features object

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
seq <- rep("hg18.chr6", 10)
src <- rep("fake_example", 10)
feature <- rep("CDS", 10)
start <- seq(1, 100, by=10)
end <- seq(10, 100, by=10)
f <- feat(seq, src, feature, start, end)
nrow.feat(f)
```

`nrow.msa` *MSA Number of Sequences*

Description

Returns the number of sequence in an MSA alignment.

Usage

```
## S3 method for class 'msa'
nrow(x)
```

Arguments

`x`	an MSA object

Value

an integer containing the number of sequences in an alignment.

Author(s)

Melissa J. Hubisz and Adam Siepel

See Also

`msa`

Examples

```
m <- msa(seqs=c("A--ACGTAT", "AG-AGGTAA", "AGGAGGTAG"),
         names=c("human", "mouse", "rat"))
nrow.msa(m)
```

nstate.hmm *HMM number of states*

Description

HMM number of states

Usage

```
nstate.hmm(hmm)
```

Arguments

hmm	An object of type hmm

Value

The number of states in the hidden Markov Model

Author(s)

Melissa J. Hubisz

numleaf.tree *Number of leaves in a Tree*

Description

Get the number of leaves in a tree

Usage

```
numleaf.tree(tree)
```

Arguments

tree	A vector of character strings, each containing a newick tree

Value

A numeric vector containing the number of leaves (species) in each tree

Author(s)

Melissa J. Hubisz and Adam Siepel

numnodes.tree *Number of Nodes in a Tree*

Description

Get the number of nodes in a tree

Usage

```
numnodes.tree(tree)
```

Arguments

tree	A vector of character strings, each containing a newick tree

Value

A numeric vector containing the number of nodes in each tree

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
numnodes.tree(c("((hg18:0.142679,(mm9:0.083220,rn4:0.090564):0.269385):
                0.020666,canFam2:0.193569);",
                "(human, (mouse, rat));"))
```

offset.msa *MSA Index Offset*

Description

Returns the offset of the first position in an alignment from some reference sequence.

Usage

```
offset.msa(x)
```

Arguments

x	an MSA object

Value

The difference between the first position in an alignment from the beginning of a chromosome.

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
m <- msa(seqs=c("A--ACGTAT", "AG-AGGTAA", "AGGAGGTAG"),
         names=c("human", "mouse", "rat"))
offset.msa(m)
m <- msa(seqs=c("A--ACGTAT", "AG-AGGTAA", "AGGAGGTAG"),
         names=c("human", "mouse", "rat"), offset=500000)
offset.msa(m)
```

---

optim.rphast *Optimize using phast's optimization code*

Description

Optimize an R function using phast's numerical optimization procedure

Usage

```
optim.rphast(func, params, lower = NULL, upper = NULL, precision = "HIGH",
  logfile = NULL, ...)
```

Arguments

func	A function to be maximized. The first argument of the function should be a numeric vector of the parameters to be optimized
params	A vector of initial values to send as the first argument of func.
lower	A vector of the same length as the vector of parameters to be optimized, giving the lower bounds for each parameter. If NULL, set the lower bounds to -Inf for all parameters.
upper	A vector of the same length as the vector of parameters to be optimized, giving the upper bounds for each parameter. If NULL, set the upper bounds to Inf for all parameters.
precision	The "precision" to use for the optimization, which affects convergence criteria. Choices are "LOW", "MED", "HIGH", or "VERY_HIGH".
logfile	If non-NULL, give the name of a file to write an optimization log to
...	Additional arguments to be passed to func at each function call. These arguments will not be optimized.

Details

This function works very much like the optim function in the stats package. In many phast applications, however, I have noticed that this function converges just as well while taking many fewer function evaluations. It uses the same optimization routine as phyloFit. In general it is most efficient to use phyloFit, because some efficiency is lost in passing objects back and forth from R to C (as is necessary when using C code to optimize an R function, whereas phyloFit uses C code to optimize a C function).

Value

A list with three elements: value: the optimized value of the function, par: a vector giving the parameters at the optimized value, and neval: the number of function evaluations used in the optimization.

Author(s)

Melissa J. Hubisz and Adam Siepel

`overlap.feat` *Feature overlap*

Description

Creates a features object containing all the features from one set which overlap features from another.

Usage

```
overlap.feat(x, filter, numbase = 1, min.percent = NULL,
  overlapping = TRUE, get.fragments = FALSE, pointer.only = FALSE)
```

Arguments

`x`	An object of type `feat` containing features to select
`filter`	An object of type `feat` which determines which elements of x to select
`numbase`	The number of bases of overlap between x and filter required to choose a record. Use NULL to ignore (but then min.percent must be defined)
`min.percent`	The minimum percent that a record must overlap with the combined records in filter in order to be chosen
`overlapping`	If `FALSE`, choose records with less than numbase overlapping bases, and less than min.percent fraction overlap if min.percent is not `NULL`
`get.fragments`	If `FALSE`, entire records are selected from x based on whether they meet selection criteria. If `TRUE`, return only the fragments of x that overlap with filter. In this case, the same fragments may be output multiple times, if they are selected by multiple entries in filter. numbase and min.percent apply in either case. When

this option is used, the return value is a list with two gffs. The first (named frags) contains the overlapping fragments, and the second (filter.frags) contain the fragments from filter which selected the overlapping fragments.

`pointer.only` If `TRUE`, the return object will only be a pointer to an object stored in C (useful for very large features; advanced use only).

Value

an object of type `feat` containing the selected entries from x (unless get.fragments==TRUE, then it returns a list with two `feat` objects; see get.fragments).

Note

If either x or filter are feature objects stored as a pointer to C memory, then this function may reorder the elements in these objects, but leave them otherwise unchanged.

Author(s)

Melissa J. Hubisz

Examples

```
feat1 <- feat(seqname=c(rep("chr1", 3), rep("chr2", 2)),
              start=c(1, 5, 100, 10, 20),
              end=c(7, 10, 105, 15, 30))
feat2 <- feat(seqname=c("chr1","chr2"),
              start=c(1,1),
              end=c(5,10))

overlap.feat(feat1, feat1)
overlap.feat(feat1, feat2, min.percent=0.25)
overlap.feat(feat1, feat2, min.percent=0.25, overlapping=FALSE)
overlap.feat(feat1, feat2, get.fragments=TRUE)
overlap.feat(feat1, feat2, get.fragments=TRUE)
rm(feat1, feat2)
```

---

pairwise.diff.msa *Get pairwise differences per site between sequences*

Description

Get pairwise differences per site between sequences

Usage

```
pairwise.diff.msa(x, seq1 = NULL, seq2 = NULL, ignore.missing = TRUE,
  ignore.gaps = TRUE)
```

Arguments

x	An object of type msa
seq1	A character vector or integer index indicating seq1 (see Value)
seq2	A character vector of integer index indicating seq2. Can only be provided if seq1 is provided.
ignore.missing	A logical value indicating whether to compare sites where either sequence has missing data.
ignore.gaps	A logical value indicating whether to compare sites where either sequence contains a gap.

Value

If seq1 and seq2 are provided, returns a numeric value giving the fraction of sites in the alignment where seq1 and seq2 differ (or zero if there are no sites to compare). If seq1 is provided and seq2 is NULL, returns a numeric vector giving this value for seq1 compared to every sequence (including itself; order of results is same as order of sequences in alignment). If both seq1 and seq2 are NULL, returns a matrix giving this value for every sequence compared with every other sequence.

Author(s)

Melissa J. Hubisz

See Also

ninf.msa To count the number of non-gap and non-missing character

---

phastBias *phastBias*

Description

PhastBias performs a phylo-HMM analysis which assesses the evidence for GC-biased gene conversion (gBGC) on a particular branch of the tree.

Usage

```
phastBias(align, mod, foreground = NULL, do.bgc = TRUE, bgc = 3,
  estimate.bgc = FALSE, bgc.expected.length = 1000,
  estimate.bgc.expected.length = FALSE, bgc.target.coverage = 0.01,
  estimate.bgc.target.coverage = TRUE, sel = -2.01483,
  cons.expected.length = 45, cons.target.coverage = 0.3,
  estimate.scale = FALSE, post.probs = TRUE)
```

Arguments

`align`	An msa object representing an alignment
`mod`	An object of type `tm` representing the neutral nucleotide substitution model.
`foreground`	A character string giving the name of a branch (or a label given to several branches) indicating which branch should be in the foreground. The foreground branch is where gBGC is tested.
`do.bgc`	If `FALSE`, do not model GC-biased gene conversion
`bgc`	Initial value for gBGC parameter B
`estimate.bgc`	If `FALSE`, do not optimize the gBGC parameter, just hold it at its initial value.
`bgc.expected.length`	Initial value for expected length of gBGC tract lengths.
`estimate.bgc.expected.length`	If `FALSE`, do not optimize the transition rate out of gBGC states (which determines the distribution of gBGC tract lengths)
`bgc.target.coverage`	Initial value for prior expected target coverage of gBGC tracts (as a fraction between 0 and 1).
`estimate.bgc.target.coverage`	If `FALSE`, constrain the rates into and out of gBGC state so that bgc.target.coverage does not change.
`sel`	Set the scaling factor for the conserved state. This is a population genetic parameter which translates to a scaling factor of sel/(1-exp(-sel)). The default value of s=-2.01483 translates to a scaling factor of 0.31 in the background branches.
`cons.expected.length`	Set the expected length of conserved elements.
`cons.target.coverage`	Set the target coverage for conserved elements.
`estimate.scale`	If `TRUE`, estimate a scaling factor for the branch lengths in all states.
`post.probs`	If `TRUE`, return value will include a data frame containing posterior probabilities for every position in the alignment and every state. Set to `FALSE` to suppress.

Details

PhastBias utilizes a HMM with the following states: neutral, conserved, neutral with gBGC, and conserved with gBGC. The scaling factor between conserved/neutral, the strength of gBGC, and the transition rates between states can be configured. It produces posterior probabilities for each state for every column of the alignment, or a set of gBGC "tracts" giving the regions where gBGC is predicted (by thresholding the posterior probability at 0.5).

Value

A list containing parameter estimates, a features object predicting which part of the alignments have gBGC probability > 0.5, and a data frame with posterior probabilities at all positions (if post.probs==TRUE)

Author(s)

Melissa J. Hubisz

`phastCons` *Produce conservation scores and identify conserved elements, given a multiple alignment and a phylo-HMM.*

Description

A phylo-HMM consisting of two states is assumed: a "conserved" state and a "non-conserved" state. If two phylogenetic models are given, the first is the conserved state, and the second is the non-conserved state. If only one model is given, then this is used as the non-conserved state, and the conserved state is obtained by multiplying the branch lengths by the parameter rho.

Usage

```
phastCons(msa, mod, rho = 0.3, target.coverage = 0.05,
  expected.length = 10, transitions = NULL, estimate.rho = FALSE,
  estimate.expected.length = FALSE, estimate.transitions = FALSE,
  estimate.trees = FALSE, viterbi = TRUE, gc = NULL, nrates = NULL,
  ref.idx = 1, quiet = FALSE)
```

Arguments

`msa`	An object of type `msa` representing the multiple alignment to be scored.
`mod`	Either a single object of type `tm`, or a list containing two `tm` objects. If two objects are given, they represent the conserved and non-conserved phylogenetic models. If one is given, then this represents the non-conserved model, and the conserved model is obtained by scaling the branches by a parameter rho.
`rho`	Set the scale (overall evolutionary rate) of the model for the conserved state to be <rho> times that of the model for the non-conserved state (0 < rho < 1). If used with estimate.trees or estimate.rho, the specified value will be used for initialization only, and rho will be estimated. This argument is ignored if mod contains two tree model objects.
`target.coverage`	A single numeric value, representing the fraction of sites in conserved elements. This argument sets a prior expectation rather than a posterior and assumes stationarity of the state-transition process. Adding this constraint causes the ratio of between-state transitions to be fixed at (1-gamma)/gamma (where gamma is the target.coverage value).
`expected.length`	A single numeric value, representing the parameter omega, which describes the expected length of conserved elements. This is an alternative to the transitions argument. If provided with target.coverage, than transition rates are fully determined, otherwise the target-coverage parameter will be estimated by maximum likelihood.

`transitions` (Alternative to target.coverage and expected.length; ignored if either of these are specified). A numeric vector of length one or two, representing the transition probabilities for the two-state HMM. The first value represents mu, the transition rate from the conserved to the non-conserved state, and the second value is nu, the rate from non-conserved to conserved. If only one value is provided then mu=nu. The rate of self-transitions are then 1-mu and 1-nu, and the expected lengths of conserved and non-conserved elements are 1/mu and 1/nu, respectively. If estimate.transition is `TRUE`, the provided values will be used for initialization.

`estimate.rho` A logical value. If `TRUE`, Estimate the parameter rho (as described above), using maximum likelihood. Estimated value is reported in return list. This use is discouraged (see note below).

`estimate.expected.length`
A logical value. If `TRUE`, estimate the expected length of conserved elements by maximum likelihood, and use the target.coverage parameter for initialization. Setting this parameter to `TRUE` is discouraged (see note below).

`estimate.transitions`
A logical value. If `TRUE`, estimate the transition rates between conserved and non-conserved states by maximum likelihood. The parameter transitions is then used for initialization. This argument is ignored if estimate.expected.length==TRUE. Setting this argument to `TRUE` is discouraged (see note below).

`estimate.trees` A logical value. If `TRUE`, estimate free parameters of tree models for conserved and non-conserved state. Setting this argument to `TRUE` is discouraged (see note below).

`viterbi` A logical value. If `TRUE`, produce discrete elements using the Viterbi algorithm.

`gc` A single numeric value given the fraction of bases that are G or C, for optional use with estimate.trees or estimate.rho. This overrides the default behavior of estimating the base composition empirically from the data.

`nrates` An integer vector of length one or two, for optional use with estimate.trees and a discrete-gamma model. Assume the specified number of rate categories, rather than the number given in the input tree model(s). If two values are given they apply to the conserved and nonconserved models, respectively.

`ref.idx` An integer value. Use the coordinate frame of the given sequence. Default is 1, indicating the first sequence in the alignment. A value of 0 indicates the coordinate frame of the entire alignment.

`quiet` If `TRUE`, suppress printing of progress information.

Value

A list containing parameter estimates. The list may have any of the following elements, depending on the arguments:

`transition.rates`
A numeric vector of length two giving the rates from the conserved to the non-conserved state, and from the non-conserved to the conserved state.

`rho` The relative evolutionary rate of the conserved state compared to the non-conserved state.

tree.models	Tree model objects describing the evolutionary process in the conserved and non-conserved states.
most.conserved	An object of type feat which describes conserved elements detected by the Viterbi algorithm.
post.prob.wig	A data frame giving a coordinate and score for individual bases in the alignment
likelihood	The likelihood of the data under the estimated model.

Note

Estimating transition rates between states by maximum likelihood, or the parameters for the phylogenetic models, does not perform very well and is discouraged. See CITE PHASTCONS PAPER for more details.

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
exampleArchive <- system.file("extdata", "examples.zip", package="rphast")
files <- c("ENr334-100k.fa", "rev.mod")
unzip(exampleArchive, files)
mod <- read.tm("rev.mod")
msa <- read.msa("ENr334-100k.fa")
rv <- phastCons(msa, mod)
names(rv)
rv2 <- phastCons(msa, mod, estimate.trees=TRUE)
names(rv2)
rv2$tree.models
unlink(files)
```

---

phyloFit *Fit a Phylogenetic model to an alignment...*

Description

Fit a Phylogenetic model to an alignment

Usage

```
phyloFit(msa, tree = NULL, subst.mod = "REV", init.mod = NULL,
  no.opt = c("backgd"), init.backgd.from.data = ifelse(is.null(init.mod),
  TRUE, FALSE), features = NULL, scale.only = FALSE, scale.subtree = NULL,
  nrates = NULL, alpha = 1, rate.constants = NULL, selection = NULL,
  init.random = FALSE, init.parsimony = FALSE, clock = FALSE,
  EM = FALSE, max.EM.its = NULL, precision = "HIGH", ninf.sites = 50,
  quiet = FALSE, bound = NULL, log.file = FALSE)
```

Arguments

msa	An alignment object. May be altered if passed in as a pointer to C memory (see Note).
tree	A character string containing a Newick formatted tree defining the topology. Required if the number of species > 3, unless init.mod is specified. The topology must be rooted, although the root is ignored if the substitution model is reversible.
subst.mod	The substitution model to use. Some possible models include "REV", "JC69", "K80", "F81", "HKY85", "R2", "U2". Run `subst.mods()` for a full list; some models are experimental.
init.mod	An object of class `tm` used to initialize the model.
no.opt	A character vector indicating which parameters NOT to optimize (instead hold constant at their initial values). By default, the equilibrium frequencies (backgd) are not optimized. Other parameters that may be indicated here are "ratematrix" for the entire rate matrix, "kappa" for models with transition/transversion ratios, "branches" to hold all branch lengths constant, "ratevar" for rate variation parameters, "scale" for the tree scaling factor, and "scale_sub" for the subtree scaling factor. This argument does NOT apply to parameters of a lineage-specific model created with `add.ls.mod`, though such parameters can be held constant by using appropriate arguments when the model is created. See `add.ls.mod` for more details about lineage-specific models.
init.backgd.from.data	A logical value; can be `FALSE` only if init.mod is provided. If `TRUE`, use observed base frequencies in data to initialize equilibrium frequencies. Otherwise use the values from init.mod. By default uses init.mod values if provided.
features	An object of type `feat`. If given, a separate model will be estimated for each feature type.
scale.only	A logical value. If `TRUE`, estimate only the scale of the tree. Branches will be held at initial values. Useful in conjunction with init.mod.
scale.subtree	A character string giving the name of a node in a tree. This option implies scale.only=TRUE. If given, estimate separate scale factors for subtree beneath identified node and the rest of the tree. The branch leading to the subtree is included in the subtree.
nrates	An integer. The number of rate categories to use. Specifying a value greater than one causes the discrete gamma model for rate variation to be used, unless rate constants are specified. The default value `NULL` implies a single rate category.
alpha	A numeric value > 0, for use with "nrates". Initial value for alpha, the shape parameter of the gamma distribution.
rate.constants	A numeric vector. Implies `nrates = length(rate.constants)`. Also implies `EM=TRUE`. Uses a non-parametric mixture model for rates, instead of a gamma distribution. The weight associated with each rate will be estimated. alpha may still be used to initialize these weights.
selection	A numeric value. If provided, use selection in the model. The value given will be the initial value for selection. If `NULL`, selection will not be used unless init.mod

	is provided and indicates a model with selection. selection scales the rate matrix by s/(1-exp(-s)). Selection is applied after the rate matrix is scaled so that the expected number of substitutions per unit time is 1. When using codon models, selection only scales nonsynonymous substitutions.
`init.random`	A logical value. If `TRUE`, parameters will be initialized randomly.
`init.parsimony`	A logical value. If `TRUE`, branch lengths will be estimated based on parsimony counts for the alignments. Currently only works for models of order0.
`clock`	A logical value. If `TRUE`, assume a molecular clock in estimation.
`EM`	A logical value. If `TRUE`, the model is fit using EM rather than the default BFGS quasi-Newton algorithm. Not available for all models/options.
`max.EM.its`	An integer value; only applies if `EM==TRUE`. The maximum number of EM iterations to perform. The EM algorithm may quit earlier if other convergence criteria are met.
`precision`	A character vector, one of "HIGH", "MED", or "LOW", denoting the level of precision to use in estimating model parameters. Affects convergence criteria for iterative algorithms: higher precision means more iterations and longer execution time.
`ninf.sites`	An integer. Require at least this many "informative" sites in order to estimate a model. An informative site as an alignment column with at least two non-gap and non-missing-data characers.
`quiet`	A logical value. If `TRUE`, do not report progress to screen.
`bound`	Defines boundaries for parameters (see Details below).
`log.file`	If TRUE, write log of optimization to the screen. If a character string, write log of optimization to the named file. Otherwise write no optimization log.

Value

An object of class `tm` (tree model), or (if several models are computed, as is possible with the features or windows options), a list of objects of class `tm`.

Parameter boundaries

Boundaries can be set for some parameters using the bound argument. The bound argument should be a vector of character strings, each element defines the boundaries for a single parameter. The boundaries are best explained by example. A value of `c("scale[0,1]", "scale_sub[1,]", "kappa[,3]")` would imply to keep the scale between 0 and 1, the subtree scale between 1 and infinity, and kappa between 0 and 3. The blank entries in the subtree_scale upper bound and kappa's lower bound indicate not to set this boundary, in which case the normal default boundary will be used for that parameter. (Most parameters are defined between 0 and infinity). Most of the parameters listed in the description of no.opt can also have their boundaries set in this way.

Note

If msa or features object are passed in as pointers to C memory, they may be altered by this function! Use `copy.msa(msa)` or `copy.feat(features)` to avoid this behavior!

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
exampleArchive <- system.file("extdata", "examples.zip", package="rphast")
files <- c("ENr334-test.maf", "ENr334-100k.fa", "ENr334-test.gff", "rev.mod")
unzip(exampleArchive, files)
m <- read.msa("ENr334-test.maf")
mod <- phyloFit(m, tree="((hg18, (mm9, rn4)), canFam2)")
mod
phyloFit(m, init.mod=mod)
likelihood.msa(m, mod)
mod$likelihood
print(mod$likelihood, digits=10)
f <- read.feat("ENr334-test.gff")
mod <- phyloFit(m, tree="((hg18, (mm9, rn4)), canFam2)",
                features=f, quiet=TRUE)
names(mod)
mod$other
mod[["5'flank"]]
phyloFit(m, init.mod=mod$AR, nrates=3, alpha=4.0)
phyloFit(m, init.mod=mod$AR, rate.constants=c(10, 5, 1))
# background frequencies options

# this should use the background frequencies from the initial mod
phyloFit(m, init.mod=mod$AR, quiet=TRUE)$backgd
mod$AR$backgd

# this should use the background frequencies from the data
phyloFit(m, init.mod=mod$AR, init.backgd.from.data=TRUE, quiet=TRUE)$backgd
mod$AR$backgd

# this should optimize the background frequencies
phyloFit(m, init.mod=mod$AR, no.opt=NULL, quiet=TRUE)$backgd
mod$AR$backgd

unlink(files)
```

---

phyloP *phyloP (basewise or by feature)*

Description

Conservation/acceleration p-values on an alignment and evolutionary model. Produces scores for every column in an alignment, or for every element in a set of features.

Usage

```
phyloP(mod, msa, method = "LRT", mode = "CON", features = NULL,
  subtree = NULL, branches = NULL, ref.idx = 1, outfile = NULL,
  outfile.only = FALSE, outfile.format = "default")
```

Arguments

mod	An object of class tm representing the neutral model.
msa	The multiple alignment to be scored.
method	The scoring method. One of "SPH", "LRT", "SCORE", or "GERP".
mode	The type of p-value to compute. One of "CON", "ACC", "NNEUT", or "CONACC".
features	An object of type feat. If given, compute p-values for every feature.
subtree	A character string giving the name of a node in the tree. Partition the tree into the subtree beneath the node and the complementary supertree, and consider conservation or acceleration in the subtree given the supertree. The branch above the specified node is included with the subtree.
branches	A vector of character strings giving the names of branches to consider in the subtree. The remaining branches are considered part of the supertree, and the test considers conservation or acceleration in the subtree relative to the supertree. This option is currently only available for method="LRT" or "SCORE".
ref.idx	index of reference sequence in the alignment. If zero, use frame of reference of entire alignment. If ref.idx==-1 and features are provided, try to guess the frame of reference of each individual feature based on sequence name.
outfile	Character string. If given, write results to given file.
outfile.only	Logical. If TRUE, do not return any results to R (this may be useful if results are very large).
outfile.format	Character string describing format of file output. Possible formats depend on other options (see description below). Current options are "default", "gff", or "wig".

Details

outfile.format options:

If features is provided, then outfile.format can be either "default" or "gff". If it is "default", then the outfile will be a table in zero-based coordinates, which includes start and end coordinates, feature name, parameter estimates, and p-values. If outfile.format is "gff", then the output file will be a GFF file (in 1-based coordinates) with a score equal to the -log10 p-value for each element.

If features is not provided, then outfile.format can be either "default" or "wig". In either case the outfile will be in fixed step wig format (see http://genome.ucsc.edu/goldenPath/help/wiggle.html). If format is "default", then each row (corresponding to one alignment column) will contain several values, such as parameter estimates and p-values for that column. If outfile.format is "wig", then the output file will be in strict wig format, with a single value per line indicating the -log10 p-value.

Value

A data frame containing scores and parameter estimates for every feature (if features is given) or for every base (otherwise).

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
exampleArchive <- system.file("extdata", "examples.zip", package="rphast")
files <- c("ENr334-100k.fa", "gencode.ENr334-100k.gff", "rev.mod")
unzip(exampleArchive, files)
tm <- read.tm("rev.mod")
tm$tree <- name.ancestors(tm$tree)
msa <- read.msa("ENr334-100k.fa", offset=41405894)
phyloP(tm, msa, method="LRT", outfile="test.out", outfile.only=TRUE, outfile.format="wig")
t1 <- phyloP(tm, msa, method="LRT", outfile="test.out")
t2 <- phyloP(tm, msa, method="LRT", outfile="test.out", outfile.format="wig")
t1 <- phyloP(tm, msa, method="SPH")
f <- read.feat("gencode.ENr334-100k.gff")
t1 <- phyloP(tm, msa, method="LRT", outfile="test.out", features=f)
t2 <- phyloP(tm, msa, method="LRT", features=f,
             outfile="test.out", outfile.format="gff", outfile.only=TRUE)
unlink("test.out")
unlink(files)
```

---

phyloP.prior *phyloP prior*

Description

Prior distribution on number of substitutions

Usage

```
phyloP.prior(mod, nsites = 100, subtree = NULL, branches = NULL,
  outfile = NULL, outfile.only = FALSE, quantiles = FALSE,
  epsilon = 1e-10)
```

Arguments

mod	An object of class tm representing the neutral model.
nsites	The number of sites in the alignment
subtree	Character string specifying the name of a node in the tree. If given, partition the tree into the subtree beneath the node and the complementary supertree, and compute joint number of substitutions in the sub/supertree. The branch above the specified node is included in the subtree.

branches	A vector of character strings givingi the names of branches to consider in the subtree. The remaininig branches are in the supertree. Return joint distribution of number of substitutions in sub/supertree.
outfile	Character string. If given, write results to given file.
outfile.only	Logical. If TRUE, do not return any results to R (this may be useful if results are very large).
quantiles	Logical. If TRUE, report quantiles of distribution rather than whole distribution.
epsilon	Numeric value indicating the thhreshold used in truncating tails of distributions; tail probabilities less than this value are discarded. This only applies to the right tail.

Value

A data.frame. If quantiles=FALSE, the columns will be the number of substitutions and their probability under the null model. If quantiles=TRUE, there will be 101 rows with the 0, 0.05, ..., 1.0th quantile.

Examples

```
exampleArchive <- system.file("extdata", "examples.zip", package="rphast")
unzip(exampleArchive, "rev.mod")
tm <- read.tm("rev.mod")
t1 <- phyloP.prior(tm, nsites=10)
t2 <- phyloP.prior(tm, nsites=20)
t3 <- phyloP.prior(tm, nsites=20, quantiles=TRUE)
t4 <- phyloP.prior(tm, nsites=20, epsilon=1e-20)
plot(t1$nsub, t1$prior)
points(t2$nsub, t2$prior, col="red")
unlink("rev.mod")
```

---

phyloP.sph *phyloP SPH*

Description

phyloP in SPH mode

Usage

```
phyloP.sph(mod, msa = NULL, mode = "CON", features = NULL,
  basewise = FALSE, subtree = NULL, ref.idx = 1, outfile = NULL,
  outfile.only = FALSE, outfile.format = "default", prior.only = FALSE,
  nsites = NULL, post.only = FALSE, fit.model = FALSE,
  epsilon = ifelse(basewise, 1e-06, 1e-10), confidence.interval = NULL,
  quantiles = FALSE)
```

Arguments

mod	An object of class tm representing the neutral model.
msa	The multiple alignment to be scored.
mode	The type of p-value to compute. One of "CON", "ACC", "NNEUT", or "CONACC".
features	A features object of type feat. If given, compute p-values for each element.
basewise	Logical. If TRUE, compute scores for every base in reference sequence. Cannot be TRUE if features is provided.
subtree	A character string giving the name of a node in the tree. Partition the tree into the subtree beneath the node and the complementary supertree, and consider conservation/acceleration in the subtree given the supertree. The branch above the specified node is included with the subtree.
ref.idx	index of reference sequence in the alignment. If zero, use frame of reference of entire alignment. If -1 and features is used, try to guess the frame of reference for each feature based on sequence name.
outfile	Character string. If given, write results to given file.
outfile.only	Logical. If TRUE, do not return any results to R (this may be useful for saving memory).
outfile.format	Character string describing output format. Possible formats depend on other options (see description below).
prior.only	Logical. If TRUE, compute only prior distribution of number of substitutions over nsites sites. Alignment is ignored in this case.
nsites	Integer. Number of sites to consider if prior.only is TRUE.
post.only	Logical. If TRUE, compute the posterior distribution of the number of substitutions given the the neutral model and the alignment.
fit.model	Logical. If TRUE, re-scale the model (including a separate scale for the subtree, if applicable) before computing the posterior distribution. This makes p-values less conservative. Cannot currently be used with features.
epsilon	Numeric value indicating the thhreshold used in truncating tails of distributions; tail probabilities less than this value are discarded. This only applies to the right tail.
confidence.interval	Numeric value between 0 and 1. If given, allow for uncertainty in the estimate of the actual number of substitutions by using a central confidence interval about the mean of given size. To be conservative, the maximum of this interval is used when computing a p-value of conservation, and the minimum is used when computing a p-value of acceleration. The variance of the posterior is computed exactly, but the confidence interval is based on the assumption that the combined distribution will be approximately normal (true for large numbers of sites by the central limit theorem).
quantiles	Logical. If TRUE, report quantiles of distribution rather than whole distribution.

Value

Either a list, data frame, or matrix, depending on options.

Author(s)

Melissa J. Hubisz and Adam Siepel

`plot.feat` *Features plot*

Description

plot features

Usage

```
## S3 method for class 'feat'
plot(x, y = 0, height = 1, plottype = "r",
  arrow.density = 5, angle = 30, col = "black", fill.col = if (plottype
  == "r") col else NULL, lty = par("lty"), lwd = par("lwd"), add = FALSE,
  xlim = range.feat(x), ylim = c(y - height * 3/4, y + height * 3/4), ...)
```

Arguments

`x`	an object of type `feat`
`y`	the location of the plot on the y axis
`height`	the height of the boxes
`plottype`	either "r" for rectangles or "a" for arrows, "b" for arrows within rectangles, or "l" for line segments only.
`arrow.density`	If plottype=="a" or "b", then this gives the density of arrows in arrows per inch. Otherwise it gives the density of shading lines in the rectangles, and a value of `NULL` implies no shading lines.
`angle`	angle (in degrees) of the shading lines or arrows.
`col`	color to draw the boxes/lines/arrows with.
`fill.col`	Color to fill the rectangles with. If `NULL` then do not fill.
`lty`	line type for lines, arrows, borders, and shading
`lwd`	line width for lines, arrows, borders and shading
`add`	if `TRUE`, add to existing plot
`xlim`	A numerical vector of length 2 giving the range for the x-axis.
`ylim`	A numerical vector of length 2 giving the range for the y-axis.
`...`	graphical parameters to be passed to `plot`.

Author(s)

Melissa J. Hubisz

Examples

```
exampleArchive <- system.file("extdata", "examples.zip", package="rphast")
featFile <- "gencode.ENr334-100k.gff"
unzip(exampleArchive, featFile)
f <- read.feat(featFile)
# note that plot(f) does not work because features are stored as data.frames
plot.feat(f[f$feature=="CDS",])
unlink(featFile)
```

`plot.gene` *Gene plot*

Description

make gene plot

Usage

```
## S3 method for class 'gene'
plot(x, y = 0, height = 1, arrow.density = 5, angle = 30,
  col = "black", lty = par("lty"), lwd = par("lwd"), add = FALSE,
  xlim = range.feat(x), ylim = c(y - height * 3/4, y + height * 3/4), ...)
```

Arguments

`x`	An object of type `feat`
`y`	the location of the plot on the y axis
`height`	the height of the boxes
`arrow.density`	The density of the arrows in arrows per inch
`angle`	angle (in degrees) of the arrow heads
`col`	color to use for plotting
`lty`	line type for arrows, borders, and shading
`lwd`	line width for arrows, borders and shading
`add`	if `TRUE`, add to existing plot
`xlim`	A numerical vector of length 2 giving the range for the x-axis.
`ylim`	A numerical vector of length 2 giving the range for the y-axis.
`...`	graphical parameters to be passed to `plot`.

Author(s)

Melissa J. Hubisz

Examples

```
exampleArchive <- system.file("extdata", "examples.zip", package="rphast")
featFile <- "sol1.gp"
unzip(exampleArchive, featFile)
f <- read.feat(featFile)
plot.gene(f)
plot.gene(f, xlim=c(0, 10000))  #zoom in

unlink(featFile)
```

`plot.lsmodel.tm` *Make a bubble plot of a lineage-specific transition matrix of a tree model.*

Description

Make a bubble plot of a lineage-specific transition matrix of a tree model.

Usage

```
## S3 method for class 'lsmodel.tm'
plot(x, i = 1, show.eq.freq = TRUE, max.cex = 10,
  eq.freq.max.cex = 5, alphabet = NULL, col = NULL, eq.freq.col = NULL,
  filled = TRUE, add = FALSE, ...)
```

Arguments

`x`	An object of type `tm`.
`i`	An integer identifying which element of tm[["ls.model"]] to plot.
`show.eq.freq`	If `TRUE`, show bubbles representing equilibrium frequencies along the bottom of plot.
`max.cex`	A scaling factor which determines the size of the largest circle
`eq.freq.max.cex`	A scaling factor which determines the size of the largest circle in the equilibrium frequencies.
`alphabet`	A character vector representing the state names for each row/column of the matrix. Can either be a vector of size `nrow(m)` or a single character string with `nrow(m)` characters. Can also be `NULL` for no row/column labels.
`col`	If `NULL`, all circles will be drawn in black. Otherwise, col can be a matrix of the same dimension of `m`, each entry should indicate the color used for the corresponding cell in the transition matrix.
`eq.freq.col`	Should be vector of same length as eq.freq, though values will be recycled. Values in the vector indicate colors to draw the equilibrium frequency bubbles.
`filled`	If `TRUE`, plot filled circles.
`add`	If `TRUE`, add to the existing plot. Otherwise create a new plot.
`...`	Further arguments to be passed to `plot`.

Author(s)

Melissa J. Hubisz

Examples

```
exampleArchive <- system.file("extdata", "examples.zip", package="rphast")
filename <- "rev.mod"
unzip(exampleArchive, filename)
tm <- read.tm(filename)
tm <- add.ls.mod(tm, branch="mm9", subst.mod="HKY85")
plot.lsmodel.tm(tm, 1)
tm$ls.model$backgd <- c(0.9, 0.05, 0.03, 0.02)
plot.lsmodel.tm(tm, 1)
plot.rate.matrix(tm[["rate.matrix"]],
                 eq.freq=tm[["backgd"]],
                 filled=FALSE,
                 alphabet=tm[["alphabet"]])
unlink(filename)
```

---

plot.msa *Plot an alignment*

Description

Plot an alignment

Usage

```
## S3 method for class 'msa'
plot(x, refseq = names.msa(x)[1], xlim = NULL, ylim = c(0,
  1), add = FALSE, pretty = FALSE, min.char.size = 0.05,
  nuc.text = NULL, nuc.text.pos = "bottom", nuc.text.col = "black", ...)
```

Arguments

x	An object of type msa
refseq	A character string naming the reference sequence to use (NULL implies frame of reference of entire alignment).
xlim	(Only used when add==FALSE. A vector of length 2 giving the coordinate range to plot in terms of refseq coordinates. If NULL use entire range of alignment.
ylim	(Only used when add==TRUE. The limits to use on the y-axis.
add	If TRUE, add to the current plot
pretty	If TRUE, display bases as dots which are in 2nd or higher row and are identical to corresponding base in 1st row.
min.char.size	The smallest value (in inches) that a character can be. If characters need to be smaller than this, skip the plot.

nuc.text	If not NULL, can be a vector of character strings. Each character string should be the same length as the MSA with respect to refseq. Each string will be displayed in its own row along with the alignment.
nuc.text.pos	If nuc.text is not NULL, can be either "top" or "bottom" to indicate where to place nuc.text relative to the alignment. Will be recycled to the length of nuc.text.
nuc.text.col	If nuc.text is not NULL, color to be used for printing nuc.text. Will be recycled to the length of nuc.text.
...	Additional arguments to be passed to plot()

Author(s)

Melissa J. Hubisz

Examples

```
exampleArchive <- system.file("extdata", "examples.zip", package="rphast")
unzip(exampleArchive, "ENr334-100k.maf")
m <- read.msa("ENr334-100k.maf")
plot.msa(m)
plot.msa(m[, 1:2])
plot.msa(m[,1:20])
plot.msa(m[1:3,1:40])
plot.msa(m[,1:100])
plot.msa(m[,1:50], refseq=NULL)
plot.msa(m[,1:50], refseq=NULL, nuc.text=rep(paste(rep("ASDFG", 10), sep="", collapse=""), 2),
         nuc.text.col=c("black", "red"), nuc.text.pos=c("top"))
rm(m)
unlink("ENr334-100k.maf")
```

---

plot.rate.matrix *Make a bubble plot of a transition matrix*

Description

Make a bubble plot of a transition matrix

Usage

```
## S3 method for class 'rate.matrix'
plot(x, eq.freq = NULL, max.cex = 10,
  eq.freq.max.cex = 5, alphabet = NULL, col = NULL, eq.freq.col = NULL,
  filled = TRUE, add = FALSE, ...)
```

Arguments

x	A square matrix representing a continuous-time markov model; rows should sum to zero, with negative values cross the diagonal
eq.freq	A numeric vector giving the equilibrium frequencies of each state. If provided, the equilibrium frequencies will be plotted along the bottom.
max.cex	A scaling factor which determines the size of the largest circle
eq.freq.max.cex	A scaling factor which determines the size of the largest circle in the equilibrium frequencies.
alphabet	A character vector representing the state names for each row/column of the matrix. Can either be a vector of size nrow(m) or a single character string with nrow(m) characters. Can also be NULL for no row/column labels.
col	If NULL, all circles will be drawn in black. Otherwise, col can be a matrix of the same dimension of m, each entry should indicate the color used for the corresponding cell in the transition matrix.
eq.freq.col	(Only applicable when eq.freq provided). Should be vector of same length as eq.freq, though values will be recycled. Values in the vector indicate colors to draw the equilibrium frequency bubbles.
filled	If TRUE, plot filled circles.
add	If TRUE, add to the existing plot. Otherwise create a new plot.
...	Further arguments to be passed to plot.

Author(s)

Melissa J. Hubisz

---

plot.tm *Make a bubble plot of the transition matrix for a tree model.*

Description

Make a bubble plot of the transition matrix for a tree model.

Usage

```
## S3 method for class 'tm'
plot(x, show.eq.freq = TRUE, max.cex = 10,
  eq.freq.max.cex = 5, alphabet = NULL, col = NULL, eq.freq.col = NULL,
  filled = TRUE, add = FALSE, ...)
```

Arguments

x	An object of type tm.
show.eq.freq	If TRUE, show bubbles representing equilibrium frequencies along the bottom of plot.
max.cex	A scaling factor which determines the size of the largest circle
eq.freq.max.cex	A scaling factor which determines the size of the largest circle in the equilibrium frequencies.
alphabet	A character vector representing the state names for each row/column of the matrix. Can either be a vector of size nrow(m) or a single character string with nrow(m) characters. Can also be NULL for no row/column labels.
col	If NULL, all circles will be drawn in black. Otherwise, col can be a matrix of the same dimension of m, each entry should indicate the color used for the corresponding cell in the transition matrix.
eq.freq.col	Should be vector of same length as eq.freq, though values will be recycled. Values in the vector indicate colors to draw the equilibrium frequency bubbles.
filled	If TRUE, plot filled circles.
add	If TRUE, add to the existing plot. Otherwise create a new plot.
...	Further arguments to be passed to plot.

Author(s)

Melissa J. Hubisz

Examples

```
exampleArchive <- system.file("extdata", "examples.zip", package="rphast")
filename <- "rev.mod"
unzip(exampleArchive, filename)
tm <- read.tm(filename)
plot(tm)
plot(tm, show.eq.freq=FALSE)
plot(tm, max.cex=20, eq.freq.max.cex=1,
     col=matrix(1:16, nrow=4),
     eq.freq.col=c("red", "green"),
     filled=TRUE, add=TRUE)
plot.rate.matrix(tm[["rate.matrix"]],
                 eq.freq=tm[["backgd"]],
                 filled=FALSE)
plot.rate.matrix(tm[["rate.matrix"]],
                 eq.freq=tm[["backgd"]],
                 filled=TRUE, add=TRUE)
unlink(filename)
```

`plot.track` *Make browser-like plot in rphast*

Description

Make browser-like plot in rphast

Usage

```
## S3 method for class 'track'
plot(x, doLabels = TRUE, cex.axis = 1, cex.labels = 1,
  cex.shortLabels = 0.75, relWigSize = 5, relMsaSize = 5, xlim = NULL,
  xlab = "coord", ylab = "", blankSpace = 0.25, axisDigits = 3,
  labelSpace = min(length(x) * 0.05, 0.25), belowLabelSpace = 0.2,
  lmar = 4, ...)
```

Arguments

`x`	a list of tracks, created by the as.track.wig or as.track.feat
`doLabels`	Logical. Whether to plot the label above each plot. Will be recycled to the length of x. Does not affect printing of shortLabels.
`cex.axis`	The character expansion factor for axis annotations.
`cex.labels`	The character expansion factor for the labels
`cex.shortLabels`	The character expansion factor for the shortLabels
`relWigSize`	The relative size of wig plots compared to feature plots
`relMsaSize`	The relative size of msa plots compared to feature plots
`xlim`	The range of the x coordinate to be plotted. If `NULL` (the default), will use the entire range represented in the resultList.
`xlab`	The label for the x axis
`ylab`	The label for the y axis
`blankSpace`	The amount of vertical blank space between each plot. This should be a single numeric value between 0 and 1, representing the total fraction of the plot occupied by blank space.
`axisDigits`	The number of digits to use on the y-axis for wig plots.
`labelSpace`	The total fraction of vertical space given to plot labels.
`belowLabelSpace`	The amount of space between a label and the plot it corresponds to, in fractions of a character width.
`lmar`	The size of the left margin (in number of lines)
`...`	Other options to be passed to `plot`. See par.
`labels`	Labels to appear directly above each plot.

Author(s)

Melissa J. Hubisz

See Also

plotPhast, which may be easier to use but less flexible

---

postprob.msa *Obtain posterior probilities of every state at every node*

Description

Obtain posterior probilities of every state at every node

Usage

```
postprob.msa(x, tm, every.site = FALSE)
```

Arguments

x	An object of type msa
tm	An object of type tm
every.site	If TRUE, return probabilities for every site rather than every site pattern (this may be very redundant and large for a large alignment with few species).

Value

An array giving the posterior probabilities of all states for every unique site pattern, or for every site if every.site is TRUE

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
exampleArchive <- system.file("extdata", "examples.zip", package="rphast")
unzip(exampleArchive, "ENr334-100k.maf")
m <- read.msa("ENr334-100k.maf")
mod <- phyloFit(m, tree="((hg18,(mm9,rn4)),canFam2)")
x <- postprob.msa(sub.msa(m, start.col=41447839, end.col=41448033,
                          refseq="hg18"), mod)
dim(x)
dimnames(x)
x[,,"CCCC"]

# now get postprobs for every site
x <- postprob.msa(sub.msa(m, start.col=41447839, end.col=41448033,
```

```
                          refseq="hg18"), mod, every.site=TRUE)
unlink("ENr334-100k.maf")
```

`print.feat` *Printing a features Object*

Description

Prints a features object.

Usage

```
## S3 method for class 'feat'
print(x, ...)
```

Arguments

`x`	an object of type `feat`
`...`	further arguments to be passed to or from other methods

Author(s)

Melissa J. Hubisz and Adam Siepel

See Also

`write.feat`

`print.msa` *Printing MSA objects*

Description

Prints an MSA (multiple sequence alignment) object.

Usage

```
## S3 method for class 'msa'
print(x, ..., print.seq = ifelse(ncol.msa(x) * nrow.msa(x) <
  500, TRUE, FALSE), format = NULL, pretty.print = FALSE)
```

Arguments

x	an object of class msa
...	additional arguments sent to print
print.seq	whether to supress printing of the alignment
format	to print sequence in if printing alignment
pretty.print	whether to pretty.print pretty-print sequence if printing alignment

Details

Valid formats for printing are "FASTA", "PHYLIP", "MPM", and "SS". See is.format.msa for details on these formats. If format is specified, the alignment is printed regardless of print.seq.

Pretty-printing will cause all characters in a column which match the value in the first row to be printed as ".". It only works for FASTA, PHYLIP, or MPM formats.

If print.seq==TRUE, then the default printing format depends on whether the sequence is stored by value (the default storage mode), or by reference. If the MSA is stored by value, the default format is as a R character vector. Otherwise, the default format is FASTA.

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
# read in an MSA stored in R
m <- msa(seqs=c("ACGTAT", "AGGTAA", "AGGTAG"),
         names=c("human", "mouse", "rat"))
print(m)
print(m, format="FASTA")
print(m, format="PHYLIP", pretty.print=TRUE)
#'
# read in an MSA stored by reference in C
m <- msa(seqs=c("ACGTAT", "AGGTAA", "AGGTAG"),
         names=c("human", "mouse", "rat"),
         pointer.only=TRUE)
print(m)
```

---

print.phastBiasResult *Pretty-print the phastBias result list without spilling giant matrices onto the screen*

Description

Pretty-print the phastBias result list without spilling giant matrices onto the screen

Usage

```
## S3 method for class 'phastBiasResult'
print(x, ...)
```

Arguments

x	phastBias result object
...	not used

Author(s)

Melissa J. Hubisz

---

print.tm *Printing Tree Models*

Description

Print a tree model

Usage

```
## S3 method for class 'tm'
print(x, aslist = FALSE, ...)
```

Arguments

x	An object of class tm.
aslist	Logical. If TRUE, print the tree model as a list rather than in tree model format.
...	arguments to be passed to/from other functions

Author(s)

Melissa J. Hubisz and Adam Siepel

See Also

tm

Examples

```
exampleArchive <- system.file("extdata", "examples.zip", package="rphast")
filename <- "rev.mod"
unzip(exampleArchive, filename)
tm <- read.tm(filename)
tm
print(tm, aslist=TRUE)
unlink(filename)
```

prune.tree *Prune a Tree*

Description

Prune sequences from a file

Usage

```
prune.tree(tree, seqs, all.but = FALSE)
```

Arguments

tree	A vector of character strings, each containing a newick tree
seqs	The sequences to prune from the trees
all.but	A logical value. If false, prunes all the named sequences from the tree. If TRUE, prunes all sequences except the ones named.

Value

a vector of character strings representing the pruned trees.

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
trees <- c("((hg18, panTro2), mm9);",
           "((hg18:0.142679,(mm9:0.083220,rn4:0.090564):0.269385):
                0.020666,canFam2:0.193569);")
prune.tree(trees, c("panTro2", "mm9"), all.but=TRUE)
prune.tree(trees, "hg18", all.but=FALSE)
```

range.feat *Features range*

Description

Get the range of a features object

Usage

```
## S3 method for class 'feat'
range(..., na.rm = FALSE)
```

Arguments

...	Objects of type feat
na.rm	Whether to remove values of NA before calculating range.

Value

A vector of size 2 indicating minimum and maximum coord in the features object

Author(s)

Melissa J. Hubisz

---

range.track *Get the coordinate range of a list of RPHAST results*

Description

Get the coordinate range of a list of RPHAST results

Usage

```
## S3 method for class 'track'
range(..., na.rm = FALSE)
```

Arguments

...	a list of tracks
na.rm	logical, indicating if NA's should be omitted

Value

a numeric vector of length two giving the minimum and maximum coordinates in any wig or feature track in the list. MSA tracks are *only* used if there are no wig or feature tracks.

Author(s)

Melissa J. Hubisz

rbind.feat *concatenate feature objects*

Description

concatenate feature objects

Usage

```
## S3 method for class 'feat'
rbind(...)
```

Arguments

`...`	objects of type `feat` to be combined into a single object

Value

An object of type `feat` containing entries from all given features

Author(s)

Melissa J. Hubisz and Adam Siepel

read.feat *Read a Feature File (GFF, BED, or GenePred)*

Description

Read a features object from a file

Usage

```
read.feat(filename, pointer.only = FALSE)
```

Arguments

`filename`	the name of the file (can be GFF, BED, GenePred, or wig: rphast will auto-detect)
`pointer.only`	Whether to store object by reference instead of a data.frame

Details

The function will guess the format of the input file automatically.

Value

If pointer.only==FALSE, a data.frame with columns corresponding to the GFF specification. Otherwise, an object which is a pointer to an object stored in C.

Author(s)

Melissa J. Hubisz and Adam Siepel

See Also

feat for more description of features objects.

msa for more explanation of the pointer.only option.

http://www.sanger.ac.uk/resources/software/gff/spec.html for a detailed description of GFF file format. The columns in features objects mirror the GFF column definitions.

http://genome.ucsc.edu/FAQ/FAQformat for descriptions of BED and GenePred formats.

Examples

```
exampleArchive <- system.file("extdata", "examples.zip", package="rphast")
featFile <- "gencode.ENr334-100k.gff"
unzip(exampleArchive, featFile)
f <- read.feat(featFile)
dim(f)
f[1:10,]
unlink(featFile)
```

---

read.hmm *Read an HMM object from a file*

Description

This function uses phast's internal hmm format, which is quite simple. See write.hmm or file used in example below for exaples of hmm format.

Usage

```
read.hmm(filename)
```

Arguments

filename	The file to read

Value

An hmm object

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
exampleArchive <- system.file("extdata", "examples.zip", package="rphast")
file <- "coding.hmm"
unzip(exampleArchive, file)
# this is a 5-state hmm with states representing
# intergenic, intron, first, second, and third codon positions.
h <- read.hmm(file)
h
unlink(file)
```

---

read.msa *Reading an MSA Object*

Description

Reads an MSA from a file.

Usage

```
read.msa(filename, format = c(guess.format.msa(filename), "FASTA")[1],
  alphabet = NULL, features = NULL, do.4d = FALSE, ordered = (do.4d ==
  FALSE && is.null(features)), tuple.size = (if (do.4d) 3 else NULL),
  do.cats = NULL, refseq = NULL, offset = 0, seqnames = NULL,
  discard.seqnames = NULL, pointer.only = FALSE)
```

Arguments

filename	The name of the input file containing an alignment.
format	input file format: one of "FASTA", "MAF", "SS", "PHYLIP", "MPM", must be correctly specified.
alphabet	the alphabet of non-missing-data chraracters in the alignment. Determined automatically from the alignment if not given.
features	An object of type feat. If provided, the return value will only contain portions of the alignment which fall within a feature. The alignment will not be ordered. The loaded regions can be further constrained with the do.4d or do.cats options. Note that if this object is passed as a pointer to a structure stored in C, the values will be altered by this function!
do.4d	Logical. If TRUE, the return value will contain only the columns corresponding to four-fold degenerate sties. Requires features to be specified.
ordered	Logical. If FALSE, the MSA object may not retain the original column order.

`tuple.size` Integer. If given, and if pointer.only is `TRUE`, MSA will be stored in sufficient statistics format, where each tuple contains tuple.size consecutive columns of the alignment.

`do.cats` Character vector if features is provided; integer vector if cats.cylce is provided. If given, only the types of features named here will be represented in the (unordered) return alignment.

`refseq` Character string specifying a FASTA format file with a reference sequence. If given, the reference sequence will be "filled in" whereever missing from the alignment.

`offset` An integer giving offset of reference sequence from beginning of chromosome. Not used for MAF or SS format.

`seqnames` A character vector. If provided, discard any sequence in the msa that is not named here. This is only implemented efficiently for MAF input files, but in this case, the reference sequence must be named.

`discard.seqnames` A character vector. If provided, discard sequenced named here. This is only implemented efficiently for MAF input files, but in this case, the reference sequenced must NOT be discarded.

`pointer.only` If `TRUE`, MSA will be stored by reference as an external pointer to an object created by C code, rather than directly in R memory. This improves performance and may be necessary for large alignments, but reduces functionality. See `msa` for more details on MSA object storage options.

Value

an MSA object.

Note

If the input is in "MAF" format and features is specified, the resulting alignment will be stripped of gaps in the reference (1st) sequence.

Author(s)

Melissa J. Hubisz and Adam Siepel

See Also

`msa`, `read.feat`

Examples

```
exampleArchive <- system.file("extdata", "examples.zip", package="rphast")
files <- c("ENr334-100k.maf", "ENr334-100k.fa", "gencode.ENr334-100k.gff")
unzip(exampleArchive, files)

# Read a fasta file, ENr334-100k.fa
# this file represents a 4-way alignment of the encode region
```

```
# ENr334 starting from hg18 chr6 position 41405894
idx.offset <- 41405894
m1 <- read.msa("ENr334-100k.fa", offset=idx.offset)
m1

# Now read in only a subset represented in a feature file
f <- read.feat("gencode.ENr334-100k.gff")
f$seqname <- "hg18"  # need to tweak source name to match name in alignment
m1 <- read.msa("ENr334-100k.fa", features=f, offset=idx.offset)

# Can also subset on certain features
do.cats <- c("CDS", "5'flank", "3'flank")
m1 <- read.msa("ENr334-100k.fa", features=f, offset=idx.offset,
               do.cats=do.cats)

# Can read MAFs similarly, but don't need offset because
# MAF file is annotated with coordinates
m2 <- read.msa("ENr334-100k.maf", features=f, do.cats=do.cats)
# Also, note that when features is given and the file is
# in MAF format, the first sequence is automatically
# stripped of gaps
ncol.msa(m1)
ncol.msa(m2)
ncol.msa(m1, "hg18")

unlink(files) # clean up
```

`read.newick.tree` *Read a Newick Tree from a File*

Description

Read a tree from a file

Usage

```
read.newick.tree(filename)
```

Arguments

`filename`	The file containing the tree.

Details

Reads a tree in newick format

Value

a character string representing the tree in newick format

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
cat(c("((hg18:0.142679,(mm9:0.083220,rn4:0.090564):0.269385):0.020666,canFam2:0.193569);",
      "(human, (mouse, rat));",
      sep="\n"), file="test.nh")
read.newick.tree("test.nh")
unlink("test.nh")
```

---

read.tm *Read a Tree Model*

Description

Read a tree model from a file

Usage

```
read.tm(filename)
```

Arguments

filename	The file containing a tree model

Value

An object of class "tm"

Author(s)

Melissa J. Hubisz and Adam Siepel

See Also

tm

Examples

```
exampleArchive <- system.file("extdata", "examples.zip", package="rphast")
filename <- "rev.mod"
unzip(exampleArchive, filename)
tm <- read.tm(filename)
tm
unlink(filename)
```

read.wig *Read a wig file*

Description

Reads fixed or variable step wig files. Stores them as a features object.

Usage

```
read.wig(file, pointer.only = FALSE)
```

Arguments

file	The file to read
pointer.only	If TRUE, store as a pointer to a C structure

Value

A GFF object representing data in wig file

reflect.phylo.hmm *Reflect a phylo-hmm across a strand*

Description

Reflect a phylo-hmm across a strand

Usage

```
reflect.phylo.hmm(x, pivot.states, mods = NULL)
```

Arguments

x	An object of type hmm
pivot.states	The list of states to "reflect" across; these should be the states that are not strand-specific. Can be an integer vector containing state indices, or a character vector corresponding to state names (in row.names(x$trans.mat))
mods	A list of objects of type tm representing phylogenetic models corresponding to each state in the hmm. If given, then the models will also be reflected and the return value will be a list with a new hmm and a new list of models.

Value

If mods==NULL then a new hmm will be returned. Otherwise a list containing the new hmm and the corresponding models will be returned.

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
#state.names <- c("neutral", "conserved", "codon1", "codon2", "codon3")
#h <- hmm(t(matrix(c(0.95, 0.04, 0.01, 0, 0,
#                    0.04, 0.95, 0.01, 0, 0,
#                    0,    0,    0,    1, 0,
#                    0,    0,    0,    0, 1,
#                    0.005, 0.005,  0.99,   0, 0), nrow=5,
#                  dimnames=list(state.names, state.names))))
#         eq.freq=c(0.6, 0.3, 0.1/3, 0.1/3, 0.1/3))
#reflect.phylo.hmm(h, c("neutral", "conserved"))
```

---

register.rphast *Register RPHAST*

Description

If you are making use of RPHAST, we would appreciate if you would let us know. This will send your name, email, and institution (all optional), as well as your IP address and rphast version to our server. We will not share your information with anyone. If you choose to send your email address, we may use it (very rarely) to let you know about major new releases and bug fixes.

Usage

```
register.rphast(name = "", email = "", institution = "", comments = "")
```

Arguments

name	Your Name (Optional). Let us know who you are, if you want.
email	Your Email Address (Optional). If given, we may use it very rarely to let you know about major new releases and bug fixes. We will not share your email address with anyone.
institution	Your Institution (Optional). Let us know where you are from.
comments	Anything else you'd like to tell us!

Details

Once you register, an empty file called "registered" will be created in ~/.rphast (non-Windows) or %appData%\rphast (Windows) which will indicate to us that you have registered, and you will no longer receive any reminders to register when rphast is loaded.

Author(s)

Nicholas Peterson

See Also

nothanks.rphast to get rid of registration reminders without actually registering.

`rename.tree` *Tree Node Renaming*

Description

Rename nodes of trees

Usage

```
rename.tree(tree, old.names, new.names)
```

Arguments

`tree`	A vector of character strings, each containing a newick tree
`old.names`	A vector of current names to be substituted
`new.names`	A vector of equal length to old.names giving the substitutions

Value

A vector of character strings, in which all nodes with names given in old.names are replaced with values from new.names

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
trees <- c("((hg18:1.0, panTro2:2.0):3.0, mm9:4.0);",
           "((hg18:0.142679,(mm9:0.083220,rn4:0.090564):0.269385):
                0.020666,canFam2:0.193569);")
rename.tree(trees,
            old.names=c("hg18", "panTro2", "mm9", "rn4", "canFam2"),
            new.names=c("human", "chimp", "mouse", "rat", "dog"))
```

rescale.tree *Scale a Tree or Subtree*

Description

Rescale a tree

Usage

```
rescale.tree(tree, scale, subtree = NULL, include.leading = FALSE)
```

Arguments

tree	A vector of character strings, each containing a newick tree
scale	A vector of scale factors for each tree (will be recycled as necessary if shorter than trees)
subtree	If not NULL, scaling will be on subtree defined by the named node. Subtrees will be recycled as necessary if shorter than trees.
include.leading	(Only applicable when subtree used) If TRUE, include the branch leading to the named node in the subtree.

Value

A vector of trees whose branches have been scaled

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
trees <- c("((hg18:1.0, panTro2:2.0):3.0, mm9:4.0);",
           "((hg18:0.142679,(mm9:0.083220,rn4:0.090564):0.269385):
                0.020666,canFam2:0.193569);")
rescale.tree(trees, 0.5)
rescale.tree(trees, c(0.5, 2.0))
trees <- name.ancestors(trees)
rescale.tree(trees, 0.5, c("hg18-panTro2", "hg18-mm9"))
```

`reverse.complement.msa` *Reverse complement a multiple sequence alignment*

Description

Reverse complement a multiple sequence alignment

Usage

```
reverse.complement.msa(x)
```

Arguments

`x`	An object of type `msa`.

Value

The reverse complement of msa.

Note

If x is stored as a pointer to an object in C, x will be changed to its reverse complement. Use reverse.complement(copy.msa(x)) to avoid this behavior. The return value will be a pointer if the input value was stored as a pointer.

Author(s)

Melissa J. Hubisz and Adam Siepel

`sample.msa` *Sample columns from an MSA*

Description

Sample columns from an MSA

Usage

```
## S3 method for class 'msa'
sample(x, size, replace = FALSE, prob = NULL,
  pointer.only = FALSE)
```

Arguments

x	An object of type msa
size	The number of columns to sample
replace	Whether to sample with replacement
prob	A vector of probability weights for sampling each column; prob=NULL implies equal probability for all columns. Probabilities need not sum to one but should be non-negative and can not all be zero.
pointer.only	If TRUE, return only a pointer to an alignment object stored in C (useful for large objects; advanced use only).

Value

An object of type msa with columns randomly re-sampled from the original

Note

This function is implemented using R's sample function in conjunction with "[.msa". It will not alter the value of x even if it is stored as a pointer.

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
m <- msa(seqs=c("AAAAAAAAAACCCCCGGT", "GGGGGGGGGGTTTTTCCA", "CCCCCCCCCCAAAAAGGA"),
         names=c("human", "mouse", "rat"))
sample.msa(m, 10, replace=TRUE)
sample.msa(m, 10, replace=TRUE, prob=c(rep(1, 10), rep(2, 5), rep(5, 2), 10))
```

---

score.hmm *Score an alignment using a general phylo-HMM*

Description

Produce likelihood of an alignment given a phylo-HMM, posterior probabilities of phylo-HMM states across an alignment, and predict states using Viterbi algorithm

Usage

```
score.hmm(msa, mod, hmm, states = NULL, viterbi = TRUE, ref.idx = 1,
  reflect.strand = NULL, features = NULL, quiet = (!is.null(features)))
```

Arguments

msa	An object of type msa
mod	A list of tree model objects, corresponding to each state in the phylo-HMM
hmm	An object of type hmm describing transitions between states, equilbrium frequencies, initial frequencies, and optionally end frequencies
states	A vector of characters naming the states of interest in the phylo-HMM, or a vector of integers corresponding to states in the transition matrix. The post.probs will give the probability of any of these states, and the viterbi regions reflect regions where the state is predicted to be any of these states. If NULL, the post.probs will be a data frame with probabilities of each state at each site, and the viterbi algorithm will give the predicted state at every site.
viterbi	A logical value indicating whether to predict a path through the phylo-HMM using the Viterbi algorithm.
ref.idx	An integer value. Use the coordinate frame of the given sequence. Default is 1, indicating the first sequence in the alignment. A value of 0 indicates the coordinate frame of the entire alignment.
reflect.strand	Given an hmm describing the forward strand, create a larger HMM that allows for features on both strands by "reflecting" the original HMM about the specified states. States can be described as a vector of integers or characters in the same manner as states argument (above). The new hmm will be used for prediction on both strands. NOTE: if reflect.strand is provided, the first state is treated as a "default" state and is implicitly included in the reflect.strand list! Also, reflection is done assuming a reversible model.
features	If non-NULL, compute the likelihood of each feature under the phylo-HMM.
quiet	If TRUE, suppress printing of progress information.

Value

If features is not NULL, returns a numeric vector with one value per feature, giving the likelihood of the feature under the phylo-HMM.

Otherwise, returns a list with some or all of the following arguments (depending on options):

in.states	An object of type feat which describes regions which fall within the interesting states specified in the states parameter, as determined by the Viterbi algorithm.
post.prob.wig	A data frame giving a coordinate and posterior probibility that each site falls within an interesting state.
likelihood	The likelihood of the data under the estimated model.

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
exampleArchive <- system.file("extdata", "examples.zip", package="rphast")
files <- c("ENr334-100k.maf", "rev.mod", "gencode.ENr334-100k.gff")
unzip(exampleArchive, files)
# make "conserved" and "neutral" models and a phylo-HMM that describes
# transitions between them, and predict conserved elements (this
# is the same thing that phastCons does, but can be extended to general
# phylo-HMMs)
align <- read.msa("ENr334-100k.maf")
neutralMod <- read.tm("rev.mod")

# create a conserved model
conservedMod <- neutralMod
conservedMod$tree <- rescale.tree(neutralMod$tree, 0.3)

# create a simple phylo-HMM
state.names <- c("neutral", "conserved")
h <- hmm(matrix(c(0.99, 0.01, 0.01, 0.99), nrow=2,
                dimnames=list(state.names, state.names)),
                eq.freq=c(neutral=0.9, conserved=0.1))
scores <- score.hmm(align, mod=list(neutral=neutralMod,
                            conserved=conservedMod),
                    hmm=h, states="conserved")
# try an alternate approach of comparing likelihoods of genes
feats <- read.feat("gencode.ENr334-100k.gff")
# plot in a region with some genes
plot.track(list(as.track.feat(scores$in.states, name="hmmScores"),
                as.track.feat(feats[feats$feature=="CDS",], name="genes")),
           xlim=c(41650000, 41680000))
unlink(files)
```

`set.rate.matrix.tm` *Set the rate matrix of a tree model using model-specific parameters.*

Description

The params argument is a numeric vector with the parameters specific to the model being used. Here is the meaning of params for each model:

- "JC69","F81": These models have no parameters; params should be NULL.
- "K80","HKY85","HKY_CODON": params should be a single value representing the transition-transversion ratio (kappa).
- "HKY85+Gap": params should be a numeric vector of length 2; the first element represents the transition/transversion ratio (kappa), and the second is the "gap parameter", the factor by which substitution rates are multiplied if they involve an indel event.
- "REV": params should be a numeric vector of length 6 (assuming a model with 4 states). With n states the vector should be of length n*(n-1)/2. The first parameter applies to the entry in the 1st row, 2nd column; the next to the 1st row, 3rd column, etc until the end of the first row; the next parameter applies to the 2nd row, 3rd column; etc.

- "SSREV": params should be a numeric vector of length 4. Assuming an alphabet "ACGT", the first parameter is the substitution rate from A->C, C->A, T->G, and G->T. The second is the rate from A->G, G->A, T->C, and C->T, The third is the rate from A->T and T->A, and the last is C->G and G->C.
- "UNREST": params should be a numeric vector of length n*n-n (where n is the number of states. params fills in the rate matrix starting at the first row going across, skipping diagonals.
- "R2": Parameters should be a numeric vector of length 48. There are 16 states in order AA, AC, AG, AT, CA, ..., TT (assuming alphabet order ACGT). Parameters are filled in starting row 1, column 2, going across and then down, filling in the matrix above the diagonal, and reflecting into the matrix below the diagonal. Only cells which represent a substitution which requires exactly one mutation are filled in; cells requiring greater than 1 mutation have rate 0.
- "U2": Similar to R2, except parameters are a numeric vector of length 96. Parameters are filled in starting row 1, column to, going across and then down, filling entire matrix (rather than refelcting across the diagonal)
- "R2S": Similar to R2, but with strand symmetry. params should be a vector of length 24. Parameters are filled in a similar fashion as R2, except the same parameter applies to substitutions which are strand symmetric.
- "U2S": Similar to U2, but with strand symmetry. params should be a vector of length 48. Parameters are filled in a similar fashion as U2, except the same parameter applies to substitutions which are strand symmetric.
- "R3","R3S","U3","U3S": Similar to R2, R2S, U2, and U2S, except there are 64 states instead of 16. params should be numeric vector of length 288, 148, 576, or 288, for models R3, R3S, U3, and U3S respectively.

Usage

```
set.rate.matrix.tm(x, params = NULL, scale = TRUE)
```

Arguments

x	An object of type tm.
params	Parameters specific to the substitution model. Should be a numeric vector of length appropriate for the model. See details below.
scale	A logical value. If TRUE, scale the matrix so that the expected number of mutations per unit time is one per base pair.

Value

An object of type tm with a rate matrix set according to params.

Author(s)

Melissa J. Hubisz and Adam Siepel

`setup.branch.site.tm` *Set up a tree model for branch site selection analysis*

Description

Set up a tree model for branch site selection analysis

Usage

```
setup.branch.site.tm(mod, foreground, bgc = FALSE, altModel = TRUE,
  init.sel.neg = 0, init.sel.pos = 0, init.bgc = 0, init.weights = NULL)
```

Arguments

`mod`	an object of type `tm`
`foreground`	a character string giving a tree branch name or label identifying foreground branches
`bgc`	If `TRUE`, then use 8 categories of sites; four with bgc in the foreground and four without.
`altModel`	If `TRUE`, then optimize the foreground positive selection parameter (constrained > 0). Otherwise hold constant at 0.
`init.sel.neg`	Initial value for negative selection parameter
`init.sel.pos`	Initial value for positive selection paramter
`init.bgc`	Initial value for bgc parameter (Ignored if bgc==FALSE)
`init.weights`	Numeric vector of length three giving the initial weight parameters. The first two values determine the relative frequencies of negatively, neutral, and positively selected sites. The last parameter determines the frequency of sites affected by bgc, and is ignored if bgc==FALSE. All values should be >= 0.

Value

An object of type `tm` which can be used as the init.mod argument to phyloFit to perform the branch-site test.

Author(s)

Melissa J. Hubisz

simulate.msa *Simulate a MSA given a tree model and HMM.*

Description

Simulates a multiple sequence alignment of specified length. Deals with base-substitution only, not indels. If one tree model is given, simply simulates a sequence from this model. If an HMM is provided, then the mod parameter should be a list of tree models with the same length as the number of states in the HMM.

Usage

```
## S3 method for class 'msa'
simulate(object, nsim, seed = NULL, hmm = NULL,
  get.features = FALSE, pointer.only = FALSE, ...)
```

Arguments

object	An object of type tm (or a list of these objects) describing the phylogenetic model from which to simulate. If it is a list of tree models then an HMM should be provided to describe transition rates between models. Currently only models of order zero are supported, and if multiple models are given, they are currently assumed to have the same topology.
nsim	The number of columns in the simulated alignment.
seed	A random number seed. Either NULL (the default; do not re-seed random number generator), or an integer to be sent to set.seed.
hmm	an object of type HMM describing transitions between the tree models across the columns of the alignment.
get.features	(For use with hmm). If TRUE, return object will be a list of length two. The first element will be the alignment, and the second will be an object of type feat describing the path through the phylo-hmm in the simulated alignment.
pointer.only	(Advanced use only). If TRUE, return only a pointer to the simulated alignment. Possibly useful for very (very) large alignments.
...	Currently not used (for S3 compatibility)

Value

An object of type MSA containing the simulated alignment.

Note

Currently only supports HMMs in which the models for each state have the same topologies.

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
filename <- "rev.mod"
exampleArchive <- system.file("extdata", "examples.zip", package="rphast")
unzip(exampleArchive, filename)
m <- matrix(nrow=3, ncol=3)
m[1,] <- c(1,2,3)
m[2,] <- c(1,5,10)
m[3,] <- c(10,4,2)
eq.freq <- c(1,2,3)
h <- hmm(m, eq.freq)
mod <- read.tm(filename)
mod2 <- mod
mod2$backgd <- rep(0.25, 4)
mod3 <- mod
mod3$backgd <- c(0.6, 0.1, 0.2, 0.1)
m <- simulate.msa(mod, 20)
m <- simulate.msa(list(mod, mod2, mod3), 20, hmm=h)
m <- matrix(1, nrow=3, ncol=3)
h <- hmm(m)
l <- simulate.msa(list(mod, mod2, mod3), 100, get.features=TRUE, hmm=h)
names(l)
l$msa
l$feats
coverage.feat(l$feats[l$feats$feature=="state1",])
unlink(filename)
```

---

smooth.wig *Smooth a wig plot in rphast*

Description

Smooth a wig plot in rphast

Usage

```
smooth.wig(coord, score, numpoints = 300)
```

Arguments

coord	The x coordinates of un-smoothed plot
score	The scores cooresponding to the x coordinates (should be same length as coord)
numpoints	The number of points to use in the new plot

Value

A data frame with numpoints rows and columns "coord" and "score" with smoothed values. If `length(coord) <= numpoints`, it will contain the original data

Author(s)

Melissa J. Hubisz

---

sort.feat *Sort a GFF*

Description

Sort a GFF

Usage

```
## S3 method for class 'feat'
sort(x, decreasing = FALSE, ...)
```

Arguments

x	An object of type feat
decreasing	Set to TRUE to sort from highest to lowest coordinates
...	Currently not used

Value

An object of type feat sorted primarily by seqname, then by start position, then by end position.

Note

If x is stored as a pointer to an object in C, the object will be modified to the return value.

Author(s)

Melissa J. Hubisz and Adam Siepel

---

split.by.feature.msa *Split an MSA by feature*

Description

Split an MSA by feature

Usage

```
## S3 method for class 'by.feature.msa'
split(x, f, drop = FALSE, pointer.only = FALSE,
  ...)
```

Arguments

x	An object of type msa
f	An object of type feat
drop	Not currently used
pointer.only	If TRUE, returned list elements are pointers to objects stored in C (advanced use only).
...	Not currently used

Value

A list of msa objects, representing the sub-alignments for each element in f

Note

Neither x nor f will be altered by this function if they are stored as pointers.

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
require("rphast")
exampleArchive <- system.file("extdata", "examples.zip", package="rphast")
files <- c("ENr334-100k.maf", "gencode.ENr334-100k.gff")
unzip(exampleArchive, files)
m <- read.msa("ENr334-100k.maf")
feats <- read.feat("gencode.ENr334-100k.gff")
feats$seqname <- "hg18"
cdsAlign <- split.by.feature.msa(m, feats[feats$feature=="CDS",])
unlink(files)
```

---

split.feat *Split features by length*

Description

Split features by length

Usage

```
## S3 method for class 'feat'
split(x, f, drop = FALSE, start.from = "left",
  pointer.only = FALSE, ...)
```

Arguments

x	An object of type feat
f	The maximum length of features in new object. Can be a vector giving a different length for each row, or a single numeric value. Values will be recycled to the same length as nrow.feat(x).
drop	A logical value saying whether to drop "left-over" elements which do not have exactly length f.
start.from	A character string, current valid values are "left" (start split at smallest coordinate for each feature), or "right" (start splitting at the last coordinate and work down). Values will be recycled to the length of nrow.feat(x)
pointer.only	If TRUE, return an object which is a pointer to a features object stored in C (advanced use only).
...	Currently not used (for S3 compatibility).

Value

An object of type feat with the same features as x but with all features of length > max.length broken into segments (starting from the first position in feature). The last piece of each split segment may be smaller than max.length

Author(s)

Melissa J. Hubisz

---

state.freq.msa *Get the observed frequencies of states in an alignment*

Description

Get the observed frequencies of states in an alignment

Usage

```
state.freq.msa(align, mod)
```

Arguments

align	An object of type msa.
mod	An object of type tm representing a tree model.

Value

A numeric vector giving the observed frequencies of each state in the model

Author(s)

Melissa J. Hubisz and Adam Siepel

`strip.gaps.msa` *MSA Strip Gaps*

Description

Strip gaps from an alignment.

Usage

```
strip.gaps.msa(x, strip.mode = 1)
```

Arguments

`x`	MSA object
`strip.mode`	Determines which gaps to strip. See Details

Details

If strip.mode can be a vector of integers or a vector of character strings. If it is a vector of integers, these are the indices of the sequences from which to strip gaps. If strip.mode is vector of character strings, each string names a sequence from which to strip gaps.

strip.mode can also be the string "all.gaps" or "any.gaps". The former will strip columns containing only gaps, whereas the latter strips columns containing even a single gap.

Value

an MSA object, with gaps stripped according to strip.mode.

Note

If x is passed as a pointer to a C structure (ie, it was created with pointer.only=TRUE), then this function will directly modify x. Use strip.gaps.msa(copy.msa(x)) to avoid this behavior. Also, the return value will be stored as a pointer if x is stored as a pointer; otherwise the return value will be stored in R.

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
m <- msa(seqs=c("A--ACGTAT-", "AG-AGGTAA-", "AGGAGGTA--"),
         names=c("human", "mouse", "rat"))
print(strip.gaps.msa(m, c("human", "mouse")), print.seq=TRUE)
print(strip.gaps.msa(m, strip.mode="any.gaps"), print.seq=TRUE)
print(strip.gaps.msa(m, strip.mode="all.gaps"), print.seq=TRUE)
print(m, print.seq=TRUE)
#' NOTE if msa stored as pointer, original object is changed
```

```
m <- as.pointer.msa(m)
temp <- strip.gaps.msa(m, "any.gaps")
print(m, print.seq=TRUE)
```

---

sub.msa *MSA Subset*

Description

Get a subset of an alignment

Usage

```
sub.msa(x, seqs = NULL, keep = TRUE, start.col = NULL, end.col = NULL,
  refseq = NULL, pointer.only = FALSE)
```

Arguments

x	An object of type msa
seqs	The sequence names to keep (or to remove if keep is FALSE)
keep	Whether to keep the named sequences or remove them
start.col	the first column to keep (columns indices start at 1)
end.col	the last column to keep (inclusive)
refseq	A character string naming the sequence in the alignment which determines the coordinates for start.col and end.col. If NULL, start.col and end.col are column indices in the multiple alignment.
pointer.only	If TRUE, return an msa object which is only a pointer to a C structure (advanced use only).

Value

A new MSA object containing a subset of the original MSA.

Note

If x is stored as a pointer and represents an unordered alignment, it may be ordered after this call. Otherwise it will not be changed.

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
m <- msa(seqs=c("ACGT---AT", "AGGTAGTAA", "AGGAAGTAG"),
         names=c("human", "mouse", "rat"))
print(sub.msa(m, c("human", "rat"), start.col=3, end.col=6),
      print.seq=TRUE)
print(sub.msa(m, c("mouse"), keep=FALSE, refseq="human",
              start.col=3, end.col=4),
      print.seq=TRUE)
```

---

subst.mods *List PHAST Substitution Models*

Description

List all valid substitution models

Usage

```
subst.mods()
```

Value

a character vector with the names of all valid substitution models

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
subst.mods()
```

---

subtree *Subtree*

Description

Get a subtree

Usage

```
subtree(tree, node, super.tree = FALSE)
```

Arguments

tree	A vector of character strings, each containing a newick tree
node	A vector of character strings, each representing the name of the node which will be the new root of the tree. If node is shorter than tree, values will be recycled, and a warning produced if `length(tree) %% length(node) != 0`
super.tree	A vector of logical values. If TRUE, then remove all nodes which are descendants of node, rather than keeping them.

Value

A vector of trees which have been pruned, removing all nodes which are not descendants of the given node.

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
trees <- c("((hg18, panTro2), mm9);",
           "((hg18:0.142679,(mm9:0.083220,rn4:0.090564):0.269385)
            :0.020666,canFam2:0.193569);")
trees <- name.ancestors(trees)
subtree(trees, c("hg18-panTro2", "mm9-rn4"))
```

---

summary.feat *Features Summary*

Description

Prints a brief summary of a features object.

Usage

```
## S3 method for class 'feat'
summary(object, ...)
```

Arguments

object	an object of type feat
...	further arguments to be passed to or from other methods

Author(s)

Melissa J. Hubisz

Examples

```
seq <- rep("hg18.chr6", 10)
src <- rep("fake_example", 10)
feature <- rep("CDS", 10)
start <- seq(1, 100, by=10)
end <- seq(10, 100, by=10)
f <- feat(seq, src, feature, start, end)
summary(f)  # this calls summary.data.frame
summary(as.pointer.feat(f))
```

`summary.msa` *MSA Summary*

Description

Prints a short description of an MSA (multiple sequence alignment) object.

Usage

```
## S3 method for class 'msa'
summary(object, ..., print.seq = ncol.msa(object) < 100 &&
  nrow.msa(object) < 30, format = "FASTA", pretty.print = FALSE)
```

Arguments

`object`	an MSA object
`...`	additional arguments sent to `print`
`print.seq`	whether to supress printing of the alignment
`format`	to print sequence in if printing alignment
`pretty.print`	whether to pretty.print pretty-print sequence if printing alignment

Author(s)

Melissa J. Hubisz

See Also

`print.msa`

Examples

```
# read in an MSA stored in R
m <- msa(seqs=c("ACGTAT", "AGGTAA", "AGGTAG"),
         names=c("human", "mouse", "rat"))
summary(m)
#'
# read in an MSA stored by reference in C
```

```
m <- msa(seqs=c("ACGTAT", "AGGTAA", "AGGTAG"),
         names=c("human", "mouse", "rat"),
         pointer.only=TRUE)
summary(m)
```

---

summary.tm *Tree Model Summary*

Description

Tree model summary

Usage

```
## S3 method for class 'tm'
summary(object, ...)
```

Arguments

object	An object of class tm
...	Parameters to be passed to/from other functions

Author(s)

Melissa J. Hubisz and Adam Siepel

See Also

tm

Examples

```
exampleArchive <- system.file("extdata", "examples.zip", package="rphast")
filename <- "rev.mod"
unzip(exampleArchive, filename)
read.tm(filename)
unlink(c(filename, "test.mod"))
```

summary.tree *Get a summary of a Newick-formatted tree, edge lengths, node names, etc*

Description

Get a summary of a Newick-formatted tree, edge lengths, node names, etc

Usage

```
## S3 method for class 'tree'
summary(object, ...)
```

Arguments

object	A character string containing a newick tree
...	Not currently used (exists for S3 compatibility)

Value

A data frame with a row for every node, containing columns: branch length (tparent), distance to root (troot), name, label (if tree labels present), and parent, rchild, lchild.

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
tree <- "(((hg18:0.01, panTro2:0.01)hg18-panTro2:0.07,
              (mm9:0.083220,rn4:0.090564)mm9-rn4:
            0.269385)hg18-rn4:0.020666,canFam2:0.193569);"
summary.tree(tree)
summary.tree(label.subtree(tree, "mm9-rn4", "rodent", include.leading=TRUE))
```

tagval *Extract value from tag-value formatted attributes*

Description

Extract value from tag-value formatted attributes

Usage

```
tagval(x, tag)
```

Arguments

x	A vector of character strings in tag-val format (as described in the GFF 2 standard; ie, "tag1 val1a val1b; tag2 val2 ; ...", or in the GFF 3 format "tag1=val1a,val1b; tag2=val2; ..."), where vals are in quotes if they are strings.
tag	The tag whose values are to be extracted.

Value

If there is at most one value per tag for each element of x, a character vector of the same length as x will be returned, containing the value for each element, or NA if the tag does not exist for that element. If some elements have multiple values, then the return value will be a list with the same length as x, each element being a character vector containing the values for the corresponding element of x (or NA for no value).

Author(s)

Melissa J. Hubisz

Examples

```
tags <- c("tag1 \"val 1a\"; tag2 \"val 2a\" \"val2a.1\" 123; tag3 \"val3a\"",
          "tag1 \"val 1b\"; tag2 \"val 2b\"; tag4 \"val4b\"",
          "tag3 \"val3a\" 1; tag4 2;")
tagval(tags, "tag1")
tagval(tags, "tag2")
tagval(tags, "tag3")
tagval(tags, "tag4")
tagval(tags, "notag")
rm(tags)
```

---

tagval.feat *Extract value from tag-value formatted attribute in features object*

Description

Extract value from tag-value formatted attribute in features object

Usage

```
tagval.feat(x, tag)
```

Arguments

x	A features object of type `feat`. The attribute field should be in tag-value format (either GFF 2 standard; ie, "tag1 val1a val1b; tag2 val2 ; ...", or, GFF 3 standard; ie, "tag1=val1a,val1b;tag2=val2; ...". where vals are in quotes if they are strings.
tag	The tag whose values are to be extracted.

Value

If there is at most one relevant value for each feature, a character vector of the same length as x will be returned, containing the value for each feature, or NA where the tag does not exist for that feature. If some elements have multiple values, then the return value will be a list with the same length as x, each element being a character vector containing the values for the corresponding element of x (or NA for no value).

Author(s)

Melissa J. Hubisz

Examples

```
exampleArchive <- system.file("extdata", "examples.zip", package="rphast")
featFile <- "sol1.gp"
unzip(exampleArchive, featFile)
f <- read.feat(featFile)
geneName <- tagval.feat(f, "transcript_id")
geneName[1:10]
length(unique(geneName)) # number of unique genes
unlink(featFile)
rm(f, geneName)
```

---

tm *Tree Models*

Description

Make a new tree model

Usage

```
tm(tree, subst.mod, rate.matrix = NULL, backgd = NULL, alphabet = "ACGT",
  nratecats = 1, alpha = 0, rate.consts = NULL, rate.weights = NULL,
  selection = NULL, root.leaf = NULL, likelihood = NULL)
```

Arguments

tree	A character string representing a phylogenetic tree in newick foromat
subst.mod	A character string giving a valid substitution mod. See subst.mods.
rate.matrix	A square matrix representing the rate of substitution from one state to the next.
backgd	A numeric vector giving the equilibrium frequencies for each state.
alphabet	A character vector containing all valid states, given in the order they are represented in rate.matrix and backgd. Defaults to "ACGT"
nratecats	The number of rate categories in the model. Defaults to 1.

alpha	If nratecats > 1, weight for each category is computed using a gamma distribution with shape parameter alpha.
rate.consts	The rate for each rate category. NULL if only one category.
rate.weights	Vector of numeric of length nratecats, determining the weight of each rate category. Must sum to 1 (will be normalized otherwise). May be defined implicitly by alpha.
selection	If not NULL, then this is a numeric value giving the selection parameter for this model. If NULL then there is no selection in the model. If selection==0.0, means that selection has no effect in the current model, but is part of the model, and by default the selection parameter will be optimized by phyloFit. The rate matrix is assumed to already be scaled by the selection parameter, if provided.
root.leaf	Usually NULL, but if set to the name of a leaf node in the tree, the tree will be re-rooted at this leaf node.
likelihood	an optional value giving the log likelihood of this model for some alignment.

Details

Tree models represent a substitution process along a phylogenetic tree. They are stored as a list, with components defined by the arguments to this function.

Value

An object of class `tm` representing a phylogenetic model.

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
tree <- "((human:0.01, chimp:0.01):0.03, mouse:0.3)"
subst.mod <- "JC69"
rate.mat <- matrix(runif(16), nrow=4, ncol=4)
for (i in 1:4)
  rate.mat[i,i] <- -sum(rate.mat[i,-i])
backgd <- runif(4)
backgd <- backgd/sum(backgd)
alphabet <- "ACGT"
t <- tm(tree, subst.mod, rate.mat, backgd, alphabet)
t

nratecats <- 3
alpha <- 1.5
rate.consts <- runif(nratecats, max=3.0)
root.leaf <- "human"
t <- tm(tree, subst.mod, rate.matrix=rate.mat,
        backgd=backgd, alphabet=alphabet,
        nratecats=nratecats, alpha=alpha,
        rate.consts=rate.consts, root.leaf=root.leaf)
t
```

`total.expected.subs.msa` *Obtain expected number of substitutions of each type on each branch*

Description

Obtain expected number of substitutions of each type on each branch

Usage

```
total.expected.subs.msa(x, tm)
```

Arguments

`x`	An object of type `msa`
`tm`	An object of type `tm`

Value

An array giving the expected number of substitutions on each branch, for each type of substitution.

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
exampleArchive <- system.file("extdata", "examples.zip", package="rphast")
unzip(exampleArchive, "ENr334-100k.maf")
m <- read.msa("ENr334-100k.maf")
mod <- phyloFit(m, tree="((hg18,(mm9,rn4)),canFam2)")
x <- total.expected.subs.msa(sub.msa(m, start.col=41447839, end.col=41448033, refseq="hg18"), mod)
dim(x)
dimnames(x)
x["mm9-rn4",,]
unlink("ENr334-100k.maf")
```

`translate.msa` *Get amino acid sequences from an alignment*

Description

Get amino acid sequences from an alignment

Usage

```
translate.msa(m, one.frame = TRUE, frame = 1)
```

Arguments

`m`	An object of type `msa` representing the alignment. The alignment is assumed to be coding sequence, already in frame.
`one.frame`	A logical value indicating whether to use the same frame for all species in the alignment, or a separate frame for each species. If `one.frame==TRUE` then every three columns of the alignment is translated into a codon, regardless of gaps within the alignment. If `one.frame==FALSE`, gaps will shift the frame in the species where they occur. In this case, the length of the seqeunces returned may not all be the same.
`frame`	An integer specifying an offset from the first column of the alignment where the coding region starts. The default 1 means start at the beginning. If `one.frame==FALSE`, frame can be a vector of integers, one for each species. Otherwise it should be a single value.

Value

A vector of character strings representing the translated alignment. The characters are amino acid codes, with '$' representing a stop codon, and '*' denoting missing data or a codon with 1 or 2 gaps, and '-' denoting a codon with all gaps.

Author(s)

Melissa J. Hubisz

Examples

```
# here is a little portion of the SOL1 gene
seqs <- c("ATGGCGACGAAGGCCGTGTGCGTGCTGAAGGGCGACGGCCCAGTGCAGG
           GCATCATCAATTTCGAGCAGAAGGCAAGGGCTGGGACGGAGGCTTGTTT
           GCGAGGCCGCTCCCACCCGCTCGTCCCCCCGCGCACCTTTGCTAGGAGC
           GGGTCGC----CCGCCAGGC-CTCGGGGCCGCCCTGGTCCAGCGCCCGG
           TCCCGGCCCGTGCCGCCCGGTCGGTGCCTTCGCCCCCAGCGGTGCGGTG
           CCCAAGTGCTGAGTCACCGGGCGGGCCCGGGC----GCGGGGCGTGGGA
           ---------CCGAGGCCGCCGCGGG",
          "ATGGCGACGAAGGCCGTGTGCGTGCTGAAGGGCGATGGCCCAGTGCAGG
           GCATCATCAATTTCGAGCAGAAGGCAAGGGCTGGGACGGAGGCTTGTTT
```

```
              GCGAGGCCGCTCCTACCCGCTCGTCCCCCCGCGCACCTTTGCTAGGAGC
              GGGTCGC----CCGCCAGGC-CTCGGGGCTGCCCTGGTCCAGCGCCCGG
              TCCCGGCCCGTGCCGCCCGGTCGGTGCCTTCGCCCCCAGCGGTGCGGTG
              CCCAAGTGCTGAGTCACCGGGCGGGCCCGGGC----GCGGGGTGTGGGA
              ---------CCGAGGCCGCCGCGGG",
             "ATGGCGATGAAAGCGGTGTGCGTGCTGAAGGGCGACGGTCCGGTGCAGG
              GAACCATCCACTTCGAGCAGAAGGCAAGGCCCGGGGC------------
              ----------------------------------------GCGGGGCGC
              AGGCCGCGGTGACGCGGCGCACCTGTGCGGGAGCACGCCACGCCCCCG-
              CCACGGCCTGAG----------------------CCCG-----------
              -CTAAGTGCTGAGTCACC--GTGGCCTGGGGCAGGGGCTGGGCGCCGGG
              AAGCGAGGCCCGGGGC-GCCGC***")
seqs <- gsub("\\s", "", seqs) #remove whitespace from seqs
align <- msa(seqs, names=c("hg19", "panTro2", "mm9"))

translate.msa(align)
translate.msa(msa(c("NNATGGCCACG")))
translate.msa(msa(c("NNATGGCCACG")), frame=3)
translate.msa(msa(c("NNATGGCCACG", "AT--GGCCACG")))
translate.msa(msa(c("NNATGGCCACG", "AT--GGCCACG")), one.frame=FALSE)
translate.msa(msa(c("NNATGGCCACG", "AT--GGCCACG")), one.frame=FALSE, frame=c(3,1))
```

`unapply.bgc.sel` *Unapply bgc+selection parameters from a matrix*

Description

Unapply bgc+selection parameters from a matrix

Usage

```
unapply.bgc.sel(m, bgc = 0, sel = 0, alphabet = "ACGT")
```

Arguments

`m`	A transition matrix
`bgc`	The bgc parameter which was used to calculate m
`sel`	The selection parameter which was used to calculate m
`alphabet`	The alphabet used for nucleotide states

Value

A matrix reflecting m before bgc and sel were applied.

Author(s)

Melissa J. Hubisz and Adam Siepel

`unique.feat` *Remove overlapping genes*

Description

Remove overlapping genes

Usage

```
## S3 method for class 'feat'
unique(x, incomparables = FALSE, ...)
```

Arguments

`x`	An object of type `feat`, usually read from a genepred file. Should have attributes labelled "transcript_id" which identify features belonging to the same gene.
`incomparables`	Not currently used (present for S3 compatibility).
`...`	Not currently used.

Value

An object of type `feat` with overlapping genes removed. If the features are scored, then the feature with the highest score is kept; otherwise the feature with the longest length. If x is a pointer to an object stored in C, the return value will also be a pointer (and x will be altered to the return value).

Note

- Long UTRs can have undesirable effects; may want to filter these out first.
- If x is a pointer to an object in C, it will be modified (to the return value).

VERY IMPORTANT: this function is not currently implemented to look at chromosomes (ie the seqname field of the feature). Therefore any genes which have overlapping coordinates REGARDLESS OF THE CHROMOSOME will be pruned to a single "non-overlapping" gene. To get around this, first subset the features by chromosome and call uniq.feat on each subset.

Also, this algorithm considers genes to be overlapping even if they are on different strands. If this is undesirable, then subset the features by strand as well as chromosome.

Author(s)

Melissa J. Hubisz and Adam Siepel

write.feat *Writing a features Object*

Description

Write a features object to a file in GFF format.

Usage

```
write.feat(x, file)
```

Arguments

x	an object of type feat
file	The name of the file to write to (will be overwritten)

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
seq <- rep("hg18.chr6", 10)
src <- rep("fake_example", 10)
feature <- rep("CDS", 10)
start <- seq(1, 100, by=10)
end <- seq(10, 100, by=10)
f <- feat(seq, src, feature, start, end)
write.feat(f, "test.gff")

unlink("test.gff") # clean up
```

write.hmm *Write an HMM object to a file*

Description

Write an HMM object to a file

Usage

```
write.hmm(x, filename, append = FALSE)
```

Arguments

x	An object of type hmm
filename	The name of the file to write to (if NULL, write to terminal)
append	If TRUE, append hmm to existing file, otherwise overwrite.

Author(s)

Melissa J. Hubisz and Adam Siepel

Examples

```
state.names <- c("neutral", "conserved")
h <- hmm(matrix(c(0.99, 0.01, 0.01, 0.99), nrow=2,
                dimnames=list(state.names, state.names)),
                eq.freq=c(neutral=0.9, conserved=0.1))
filename <- tempfile()
write.hmm(h, filename)
unlink(filename)
```

---

write.msa *Writing MSA Objects to Files*

Description

Writes a multiple sequence alignment (MSA) object to a file in one of several formats.

Usage

```
write.msa(x, file=NULL,
format=ifelse((f <- guess.format.msa(file, method="extension"))=="UNKNOWN", "FASTA", f),
pretty.print=FALSE)
```

Arguments

x	an object of class msa
file	File to write (will be overwritten). If NULL, output goes to terminal.
format	format to write MSA object. Valid values are "FASTA", "PHYLIP", "MPM", or "SS".
pretty.print	Whether to pretty-print alignment (turning bases which match the first base in the same column to ".").

Note

pretty.print does not work if format="SS".

Author(s)

Melissa J. Hubisz and Adam Siepel

write.tm *Wrting Tree Models*

Description

Write a tree model to a file (or to the terminal)

Usage

```
write.tm(tm, filename = NULL, append = FALSE)
```

Arguments

tm	An object of class "tm"
filename	The filename to write to (use NULL for output to terminal)
append	Whether to append the tree to the end of the file (if FALSE, overwrites file). Not used if filename is NULL

Author(s)

Melissa J. Hubisz and Adam Siepel

See Also

tm

Examples

```
exampleArchive <- system.file("extdata", "examples.zip", package="rphast")
filename <- "rev.mod"
unzip(exampleArchive, filename)
tm <- read.tm(filename)
tm
write.tm(tm, NULL)
write.tm(tm, "test.mod")
unlink(c(filename, "test.mod"))
```

write.wig *Writing a wig file*

Description

Write a fixedStep wig file

Usage

```
write.wig(chrom, start, score, span = 1, file = NULL, append = FALSE)
```

Arguments

chrom	A character vector giving chromosome name for each point. Will be recycled to length(start)
start	An integer vector giving start coordinate for each point.
score	A numeric vector giving score at each point Will be recycled to length(start)
span	An integer giving span (ie, length) of each element (all elements must have the same length, so only a single value is allowed).
file	The name of the file to write to (will be overwritten). A value of NULL implies write to console.
append	Whether to append to the file. If FALSE, file will be overwritten.

Author(s)

Melissa J. Hubisz

Examples

```
write.wig(chrom=c("chr1", "chr1", "chr1", "chr1", "chr2"),
          start=c(1, 11, 21, 100, 1),
          span=3,
          score=runif(5))
```

write.wig.feat *Write a features object in fixedStep wig format*

Description

Write a features object in fixedStep wig format

Usage

```
write.wig.feat(x, file = NULL, append = FALSE)
```

Arguments

x	An object of type feat
file	The name of the file to write to. A value of NULL implies write to console.
append	If TRUE, append to the file. Otherwise overwrite.

Note

Wig format only contains chromosome, coordinates, and score. Any other data will be lost.

This function will quit with an error if the elements of x are not all the same length (as required by fixedStep wig format).

If x is stored as a pointer to a C structure, the elements will be sorted by this function.

Author(s)

Melissa J. Hubisz

Examples

```
f <- feat(seqname=c("chr1", "chr1", "chr1", "chr1", "chr2"),
          start=c(1, 11, 21, 100, 1),
          end=c(3, 13, 23, 102, 3),
          score=runif(5))
write.wig.feat(f)
```

[.msa *Extract, replace, reorder MSA*

Description

Treat multiple sequence alignment as a matrix where each row corresponds to a sequence for one species, and each column is one position aligned across all species.

Usage

```
## S3 method for class 'msa'
x[rows, cols, pointer.only]
```

Arguments

x	An object of type msa
rows	A numeric vector of sequence indices, character vector (containing sequence name), or logical vector (containing sequences to keep). If logical vector it will be recycled as necessary to the same length as nrow.msa(x).
cols	A numeric vector of alignment columns, or a logical vector containing columns to keep. If logical vector it will be recycled as necessary to the same length as ncol.msa(x). Note that these are in coordinates with respect to the entire alignment. x$idx.offset is ignored here.

pointer.only — If TRUE, return an object which is only a pointer to a structure stored in C (useful for large alignments; advanced use only). In certain cases when the original alignment is stored in R, it may be more efficient return an object in R, in which case this argument will be ignored.

Details

The bracket notation can return a subset of the alignment, or re-order rows and columns.

Note

This function will not alter the value of x even if it is stored as a pointer to a C structure.

Author(s)

Melissa J. Hubisz and Adam Siepel

See Also

sub.msa which can subset columns based on genomic coordinates, and extract.feature.msa which can subset based on genomic coordinates denoted in a features object.

Examples

```
m <- msa(seqs=c("ACGTAT", "AGGTAA", "AGGTAG"),
         names=c("human", "mouse", "rat"))
print(m[c("rat", "rat", "human"), ], print.seq=TRUE)
print(m[c(3,3,1),], print.seq=TRUE)
print(m[c(TRUE, FALSE, TRUE),], print.seq=TRUE)
print(m[TRUE,], print.seq=TRUE)
print("[.msa"(m, "mouse",c(1,6,3,5)), print.seq=TRUE)
```

[<-.msa — *Replace subsets of an alignment*

Description

Replace subsets of an alignment

Usage

```
## S3 replacement method for class 'msa'
x[rows, cols] <- value
```

Arguments

x	An object of type msa
rows	A numeric vector of sequence indices, character vector (containing sequence names), or logical vector. If logical vector, it will be recycled as necessary to the length of nrow.msa(x). If not provided, all rows are selected.
cols	A numeric vector of alignment columns, or a logical vector. If logical vector it will be recycled to the same length as ncol.msa(x). Note that these are coordinates with respect to the entire alignment. x$idx.offset is ignored here. If cols is not provided, all columns are selected.
value	The value to replace in the indicated rows/columns. Should be a character representing a base (ie, "A", "C", "G", "T", "N", "-"). Can be a single value or a vector of values which match number of selected cells. This value will be recycled to the necessary length, and an error produced if the necessary length is not an even multiple of length(value). Can also give a single character string, in which case it will be expanded into a vector using strsplit.

Value

An object of type msa with the chosen rows/columns replaced by value.

Note

If x is stored as a pointer, x will be changed to the return value.

Author(s)

Melissa J. Hubisz

Examples

```
m <- msa(seqs=c("ACGTAT", "AGGTAA", "AGGTAG"),
         names=c("human", "mouse", "rat"))
m[1:2,4:6] <- "G"
m[1,] <- "A"
m[,4:5] <- "-"
m["rat",] <- "ABCDEF"
```

Index