Package ‘alakazam’

November 1, 2021

Type Package

Version 1.2.0

Date 2021-10-31

Title Immunoglobulin Clonal Lineage and Diversity Analysis

Description Provides methods for high-throughput adaptive immune receptor repertoire sequencing (AIRR-Seq; Rep-Seq) analysis. In particular, immunoglobulin (Ig) sequence lineage reconstruction, lineage topology analysis, diversity profiling, amino acid property analysis and gene usage.
Citations:
Gupta and Vander Heiden, et al (2017) <doi:10.1093/bioinformatics/btv359>,
Stern, Yaari and Vander Heiden, et al (2014) <doi:10.1126/scitranslmed.3008879>.

License AGPL-3

URL `https://alakazam.readthedocs.io/`

BugReports `https://bitbucket.org/kleinstein/alakazam/issues`

LazyData true

BuildVignettes true

VignetteBuilder knitr, rmarkdown

Encoding UTF-8

SystemRequirements C++11

LinkingTo Rcpp

biocViews Software, AnnotationData

Depends R (>= 4.0), ggplot2 (>= 3.3.4)

Imports airr (>= 1.3), ape, dplyr (>= 1.0), graphics, grid, igraph (>= 1.0), Matrix, methods, progress, Rcpp (>= 0.12.12), readr, rlang, scales, seqinr, stats, stringi, tibble, tidyr (>= 1.0), utils, Biostrings (>= 2.56.0), GenomicAlignments (>= 1.24.0), IRanges (>= 2.22.2)

Suggests knitr, rmarkdown, testthat

RoxygenNote 7.1.2

Collate 'Alakazam.R' 'AminoAcids.R' 'Classes.R' 'Core.R' 'Data.R' 'Diversity.R' 'Deprecated.R' 'Fastq.R' 'Gene.R' 'Lineage.R' 'RcppExports.R' 'Sequence.R' 'Topology.R'

NeedsCompilation yes

Author Namita Gupta [aut],
Susanna Marquez [aut],
Nima Nouri [aut],
Ruoyi Jiang [aut],
Julian Zhou [aut],
Kenneth Hoehn [aut],
Daniel Gadala-Maria [ctb],
Edel Aron [ctb],
Jason Vander Heiden [cre, aut],
Steven Kleinstein [aut, cph]

Maintainer Jason Vander Heiden <jason.vanderheiden@gmail.com>

Repository CRAN

Date/Publication 2021-11-01 16:10:16 UTC

R topics documented:

ABBREV_AA *Amino acid abbreviation translations*

Description

Mappings of amino acid abbreviations.

Usage

```
ABBREV_AA
```

Format

Named character vector defining single-letter character codes to three-letter abbreviation mappings.

Examples

```
aa <- c("Ala", "Ile", "Trp")
translateStrings(aa, ABBREV_AA)
```

AbundanceCurve-class *S4 class defining a clonal abundance curve*

Description

AbundanceCurve defines clonal abundance values.

Usage

```
## S4 method for signature 'AbundanceCurve'
print(x)

## S4 method for signature 'AbundanceCurve,missing'
plot(x, y, ...)
```

Arguments

x AbundanceCurve object

y ignored.

... arguments to pass to plotDiversityCurve.

Slots

abundance data.frame with relative clonal abundance data and confidence intervals, containing the following columns:

- group: group identifier.
- clone_id or CLONE: clone identifier.
- p: relative abundance of the clone.
- lower: lower confidence inverval bound.
- upper: upper confidence interval bound.
- rank: the rank of the clone abundance.

bootstrap data.frame of bootstrapped clonal distributions.

clone_by string specifying the name of the clone column.

group_by string specifying the name of the grouping column.

groups vector specifying the names of unique groups in group column.

n numeric vector indication the number of sequences sampled in each group.

nboot numeric specifying the number of bootstrap iterations to use.

ci confidence interval defining the upper and lower bounds (a value between 0 and 1).

---

alakazam *The alakazam package*

Description

alakazam in a member of the Immcantation framework of tools and serves five main purposes:

- Providing core functionality for other R packages in Immcantation. This includes common tasks such as file I/O, basic DNA sequence manipulation, and interacting with V(D)J segment and gene annotations.
- Providing an R interface for interacting with the output of the pRESTO and Change-O tool suites.
- Performing clonal abundance and diversity analysis on lymphocyte repertoires.
- Performing lineage reconstruction on clonal populations of immunoglobulin (Ig) sequences.
- Performing physicochemical property analyses of lymphocyte receptor sequences.

For additional details regarding the use of the alakazam package see the vignettes:
browseVignettes("alakazam")

File I/O

- readChangeoDb: Input Change-O style files.
- writeChangeoDb: Output Change-O style files.

Sequence cleaning

- maskSeqEnds: Mask ragged ends.
- maskSeqGaps: Mask gap characters.
- collapseDuplicates: Remove duplicate sequences.

Lineage reconstruction

- makeChangeoClone: Clean sequences for lineage reconstruction.
- buildPhylipLineage: Perform lineage reconstruction of Ig sequences.

Lineage topology analysis

- tableEdges: Tabulate annotation relationships over edges.
- testEdges: Significance testing of annotation edges.
- testMRCA: Significance testing of MRCA annotations.
- summarizeSubtrees: Various summary statistics for subtrees.
- plotSubtrees: Plot distributions of summary statistics for a population of trees.

Diversity analysis

- countClones: Calculate clonal abundance.
- estimateAbundance: Bootstrap clonal abundance curves.
- alphaDiversity: Generate clonal alpha diversity curves.
- plotAbundanceCurve: Plot clone size distribution as a rank-abundance
- plotDiversityCurve: Plot clonal diversity curves.
- plotDiversityTest: Plot testing at given diversity hill indicex.

Ig and TCR sequence annotation

- countGenes: Calculate Ig and TCR allele, gene and family usage.
- extractVRegion: Extract CDRs and FWRs sub-sequences.
- getAllele: Get V(D)J allele names.
- getGene: Get V(D)J gene names.
- getFamily: Get V(D)J family names.
- junctionAlignment: Junction alignment properties

Sequence distance calculation

- seqDist: Calculate Hamming distance between two sequences.
- seqEqual: Test two sequences for equivalence.
- pairwiseDist: Calculate a matrix of pairwise Hamming distances for a set of sequences.
- pairwiseEqual: Calculate a logical matrix of pairwise equivalence for a set of sequences.

Amino acid propertes

- translateDNA: Translate DNA sequences to amino acid sequences.
- aminoAcidProperties: Calculate various physicochemical properties of amino acid sequences.
- countPatterns: Count patterns in sequences.

References

1. Vander Heiden JA, Yaari G, et al. pRESTO: a toolkit for processing high-throughput sequencing raw reads of lymphocyte receptor repertoires. Bioinformatics. 2014 30(13):1930-2.
2. Stern JNH, Yaari G, Vander Heiden JA, et al. B cells populating the multiple sclerosis brain mature in the draining cervical lymph nodes. Sci Transl Med. 2014 6(248):248ra107.
3. Wu Y-CB, et al. Influence of seasonal exposure to grass pollen on local and peripheral blood IgE repertoires in patients with allergic rhinitis. J Allergy Clin Immunol. 2014 134(3):604-12.
4. Gupta NT, Vander Heiden JA, et al. Change-O: a toolkit for analyzing large-scale B cell immunoglobulin repertoire sequencing data. Bioinformatics. 2015 Oct 15;31(20):3356-8.

`aliphatic` *Calculates the aliphatic index of amino acid sequences*

Description

`aliphatic` calculates the aliphatic index of amino acid sequences using the method of Ikai. Non-informative positions are excluded, where non-informative is defined as any character in `c("X","-",".","*")`.

Usage

```
aliphatic(seq, normalize = TRUE)
```

Arguments

`seq`	vector of strings containing amino acid sequences.
`normalize`	if `TRUE` then divide the aliphatic index of each amino acid sequence by the number of informative positions. Non-informative position are defined by the presence any character in `c("X","-",".","*")`. If `FALSE` then return the raw aliphatic index.

Value

A vector of the aliphatic indices for the sequence(s).

References

1. Ikai AJ. Thermostability and aliphatic index of globular proteins. J Biochem. 88, 1895-1898 (1980).

Examples

```
seq <- c("CARDRSTPWRRGIASTTVRTSW", NA, "XXTQMYVRT")
aliphatic(seq)
```

---

alphaDiversity *Calculate clonal alpha diversity*

Description

`alphaDiversity` takes in a data.frame or AbundanceCurve and computes diversity scores (D) over an interval of diversity orders (q).

Usage

```
alphaDiversity(data, min_q = 0, max_q = 4, step_q = 0.1, ci = 0.95, ...)
```

Arguments

`data`	data.frame with Change-O style columns containing clonal assignments or a AbundanceCurve generate by estimateAbundance object containing a previously calculated bootstrap distributions of clonal abundance.
`min_q`	minimum value of q.
`max_q`	maximum value of q.
`step_q`	value by which to increment q.
`ci`	confidence interval to calculate; the value must be between 0 and 1.
`...`	additional arguments to pass to estimateAbundance. Additional arguments are ignored if a AbundanceCurve is provided as input.

Details

Clonal diversity is calculated using the generalized diversity index (Hill numbers) proposed by Hill (Hill, 1973). See calcDiversity for further details.

To generate a smooth curve, D is calculated for each value of q from `min_q` to `max_q` incremented by `step_q`. When `uniform=TRUE` variability in total sequence counts across unique values in the `group` column is corrected by repeated resampling from the estimated complete clonal distribution to a common number of sequences. The complete clonal abundance distribution that is resampled from is inferred by using the Chao1 estimator to infer the number of unseen clones, followed by applying the relative abundance correction and unseen clone frequencies described in Chao et al, 2015.

The diversity index (D) for each group is the mean value of over all resampling realizations. Confidence intervals are derived using the standard deviation of the resampling realizations, as described in Chao et al, 2015.

Significance of the difference in diversity index (`D`) between groups is tested by constructing a bootstrap delta distribution for each pair of unique values in the `group` column. The bootstrap delta distribution is built by subtracting the diversity index `Da` in group `a` from the corresponding value Db in group `b`, for all bootstrap realizations, yielding a distribution of `nboot` total deltas; where group `a` is the group with the greater mean `D`. The p-value for hypothesis `Da != Db` is the value of `P(0)` from the empirical cumulative distribution function of the bootstrap delta distribution, multiplied by 2 for the two-tailed correction.

Note, this method may inflate statistical significance when clone sizes are uniformly small, such as when most clones sizes are 1, sample size is small, and `max_n` is near the total count of the smallest data group. Use caution when interpreting the results in such cases.

Value

A DiversityCurve object summarizing the diversity scores.

References

1. Hill M. Diversity and evenness: a unifying notation and its consequences. Ecology. 1973 54(2):427-32.
2. Chao A. Nonparametric Estimation of the Number of Classes in a Population. Scand J Stat. 1984 11, 265270.
3. Chao A, et al. Rarefaction and extrapolation with Hill numbers: A framework for sampling and estimation in species diversity studies. Ecol Monogr. 2014 84:45-67.
4. Chao A, et al. Unveiling the species-rank abundance distribution by generalizing the Good-Turing sample coverage theory. Ecology. 2015 96, 11891201.

See Also

See calcDiversity for the basic calculation and DiversityCurve for the return object. See plotDiversityCurve for plotting the return object.

Examples

```
# Group by sample identifier in two steps
abund <- estimateAbundance(ExampleDb, group="sample_id", nboot=100)
div <- alphaDiversity(abund, step_q=1, max_q=10)
plotDiversityCurve(div, legend_title="Sample")

# Grouping by isotype rather than sample identifier in one step
div <- alphaDiversity(ExampleDb, group="c_call", min_n=40, step_q=1, max_q=10,
                      nboot=100)
plotDiversityCurve(div, legend_title="Isotype")
```

aminoAcidProperties *Calculates amino acid chemical properties for sequence data*

Description

`aminoAcidProperties` calculates amino acid sequence physicochemical properties, including length, hydrophobicity, bulkiness, polarity, aliphatic index, net charge, acidic residue content, basic residue content, and aromatic residue content.

Usage

```
aminoAcidProperties(
  data,
  property = c("length", "gravy", "bulk", "aliphatic", "polarity", "charge", "basic",
    "acidic", "aromatic"),
  seq = "junction",
  nt = TRUE,
  trim = FALSE,
  label = NULL,
  ...
)
```

Arguments

`data`	`data.frame` containing sequence data.
`property`	vector strings specifying the properties to be calculated. Defaults to calculating all defined properties.
`seq`	`character` name of the column containing input sequences.
`nt`	boolean, TRUE if the sequences (or sequence) are DNA and will be translated.
`trim`	if `TRUE` remove the first and last codon/amino acids from each sequence before calculating properties. If `FALSE` do not modify input sequences.
`label`	name of sequence region to add as prefix to output column names.
`...`	additional named arguments to pass to the functions gravy, bulk, aliphatic, polar or charge.

Details

For all properties except for length, non-informative positions are excluded, where non-informative is defined as any character in `c("X","-",".","*")`.

The scores for gravy, bulkiness and polarity are calculated as simple averages of the scores for each informative positions. The basic, acid and aromatic indices are calculated as the fraction of informative positions falling into the given category.

The aliphatic index is calculated using the Ikai, 1980 method.

The net charge is calculated using the method of Moore, 1985, excluding the N-terminus and C-terminus charges, and normalizing by the number of informative positions. The default pH for the calculation is 7.4.

The following data sources were used for the default property scores:

- hydropathy: Kyte & Doolittle, 1982.
- bulkiness: Zimmerman et al, 1968.
- polarity: Grantham, 1974.
- pK: EMBOSS.

Value

A modified `data` data.frame with the following columns:

- `*_aa_length`: number of amino acids.
- `*_aa_gravy`: grand average of hydrophobicity (gravy) index.
- `*_aa_bulk`: average bulkiness of amino acids.
- `*_aa_aliphatic`: aliphatic index.
- `*_aa_polarity`: average polarity of amino acids.
- `*_aa_charge`: net charge.
- `*_aa_basic`: fraction of informative positions that are Arg, His or Lys.
- `*_aa_acidic`: fraction of informative positions that are Asp or Glu.
- `*_aa_aromatic`: fraction of informative positions that are His, Phe, Trp or Tyr.

Where `*` is the value from `label` or the name specified for `seq` if `label=NULL`.

References

1. Zimmerman JM, Eliezer N, Simha R. The characterization of amino acid sequences in proteins by statistical methods. J Theor Biol 21, 170-201 (1968).
2. Grantham R. Amino acid difference formula to help explain protein evolution. Science 185, 862-864 (1974).
3. Ikai AJ. Thermostability and aliphatic index of globular proteins. J Biochem 88, 1895-1898 (1980).
4. Kyte J, Doolittle RF. A simple method for displaying the hydropathic character of a protein. J Mol Biol 157, 105-32 (1982).
5. Moore DS. Amino acid and peptide net charges: A simple calculational procedure. Biochem Educ 13, 10-11 (1985).
6. Wu YC, et al. High-throughput immunoglobulin repertoire analysis distinguishes between human IgM memory and switched memory B-cell populations. Blood 116, 1070-8 (2010).
7. Wu YC, et al. The relationship between CD27 negative and positive B cell populations in human peripheral blood. Front Immunol 2, 1-12 (2011).
8. http://emboss.sourceforge.net/apps/cvs/emboss/apps/iep.html

See Also

See countPatterns for counting the occurance of specific amino acid subsequences. See gravy, bulk, aliphatic, polar and charge for functions that calculate the included properties individually.

Examples

```
# Subset example data
db <- ExampleDb[c(1,10,100), c("sequence_id", "junction")]

# Calculate default amino acid properties from DNA sequences
aminoAcidProperties(db, seq="junction")
# Calculate default amino acid properties from amino acid sequences
# Use a custom output column prefix
db$junction_aa <- translateDNA(db$junction)
aminoAcidProperties(db, seq="junction_aa", label="junction", nt=FALSE)

# Use the Grantham, 1974 side chain volume scores from the seqinr package
# Set pH=7.0 for the charge calculation
# Calculate only average volume and charge
# Remove the head and tail amino acids from the junction, thus making it the CDR3
library(seqinr)
data(aaindex)
x <- aaindex[["GRAR740103"]]$I
# Rename the score vector to use single-letter codes
names(x) <- translateStrings(names(x), ABBREV_AA)
# Calculate properties
aminoAcidProperties(db, property=c("bulk", "charge"), seq="junction",
                    trim=TRUE, label="cdr3", bulkiness=x, pH=7.0)
```

---

baseTheme *Standard ggplot settings*

Description

baseTheme defines common ggplot theme settings for plotting.

Usage

```
baseTheme(sizing = c("figure", "window"))
```

Arguments

sizing	defines the style and sizing of the theme. One of c("figure","window") where sizing="figure" is appropriately sized for pdf export at 7 to 7.5 inch width, and sizing="window" is sized for an interactive session.

Value

A ggplot2 object.

See Also

theme.

---

buildPhylipLineage *Infer an Ig lineage using PHYLIP*

Description

buildPhylipLineage reconstructs an Ig lineage via maximum parsimony using the dnapars application, or maximum liklihood using the dnaml application of the PHYLIP package.

Usage

```
buildPhylipLineage(
  clone,
  phylip_exec,
  dist_mat = getDNAMatrix(gap = 0),
  rm_temp = FALSE,
  verbose = FALSE,
  temp_path = NULL,
  onetree = FALSE,
  branch_length = c("mutations", "distance")
)
```

Arguments

clone	ChangeoClone object containing clone data.
phylip_exec	absolute path to the PHYLIP dnapars executable.
dist_mat	character distance matrix to use for reassigning edge weights. Defaults to a Hamming distance matrix returned by getDNAMatrix with gap=0. If gap characters, c("-","."), are assigned a value of -1 in dist_mat then contiguous gaps of any run length, which are not present in both sequences, will be counted as a distance of 1. Meaning, indels of any length will increase the sequence distance by 1. Gap values other than -1 will return a distance that does not consider indels as a special case.
rm_temp	if TRUE delete the temporary directory after running dnapars; if FALSE keep the temporary directory.
verbose	if FALSE suppress the output of dnapars; if TRUE STDOUT and STDERR of dnapars will be passed to the console.
temp_path	specific path to temp directory if desired.
onetree	if TRUE save only one tree.
branch_length	specifies how to define branch lengths; one of "mutations" or "distance". If set to "mutations" (default), then branch lengths represent the number of mutations between nodes. If set to "distance", then branch lengths represent the expected number of mutations per site, unaltered from PHYLIP output.

Details

buildPhylipLineage builds the lineage tree of a set of unique Ig sequences via maximum parsimony through an external call to the dnapars application of the PHYLIP package. dnapars is called with default algorithm options, except for the search option, which is set to "Rearrange on one best tree". The germline sequence of the clone is used for the outgroup.

Following tree construction using dnapars, the dnapars output is modified to allow input sequences to appear as internal nodes of the tree. Intermediate sequences inferred by dnapars are replaced by children within the tree having a Hamming distance of zero from their parent node. With the default dist_mat, the distance calculation allows IUPAC ambiguous character matches, where an ambiguous character has distance zero to any character in the set of characters it represents. Distance calculation and movement of child nodes up the tree is repeated until all parent-child pairs have a distance greater than zero between them. The germline sequence (outgroup) is moved to the root of the tree and excluded from the node replacement processes, which permits the trunk of the tree to be the only edge with a distance of zero. Edge weights of the resultant tree are assigned as the distance between each sequence.

Value

An igraph graph object defining the Ig lineage tree. Each unique input sequence in clone is a vertex of the tree, with additional vertices being either the germline (root) sequences or inferred intermediates. The graph object has the following attributes.

Vertex attributes:

- name: value in the sequence_id column of the data slot of the input clone for observed sequences. The germline (root) vertex is assigned the name "Germline" and inferred intermediates are assigned names with the format "Inferred1", "Inferred2", ...
- sequence: value in the sequence column of the data slot of the input clone for observed sequences. The germline (root) vertex is assigned the sequence in the germline slot of the input clone. The sequence of inferred intermediates are extracted from the dnapars output.
- label: same as the name attribute.

Additionally, each other column in the data slot of the input clone is added as a vertex attribute with the attribute name set to the source column name. For the germline and inferred intermediate vertices, these additional vertex attributes are all assigned a value of NA.

Edge attributes:

- weight: Hamming distance between the sequence attributes of the two vertices.
- label: same as the weight attribute.

Graph attributes:

- clone: clone identifier from the clone slot of the input ChangeoClone.
- v_gene: V-segment gene call from the v_gene slot of the input ChangeoClone.
- j_gene: J-segment gene call from the j_gene slot of the input ChangeoClone.
- junc_len: junction length (nucleotide count) from the junc_len slot of the input ChangeoClone.

  Alternatively, this function will return an phylo object, which is compatible with the ape package. This object will contain reconstructed ancestral sequences in nodes attribute.

References

1. Felsenstein J. PHYLIP - Phylogeny Inference Package (Version 3.2). Cladistics. 1989 5:164-166.
2. Stern JNH, Yaari G, Vander Heiden JA, et al. B cells populating the multiple sclerosis brain mature in the draining cervical lymph nodes. Sci Transl Med. 2014 6(248):248ra107.

See Also

Takes as input a ChangeoClone. Temporary directories are created with makeTempDir. Distance is calculated using seqDist. See the igraph graph documentation for how to work with graph objects.

Examples

```
## Not run:
# Preprocess clone
db <- subset(ExampleDb, clone_id == 3138)
clone <- makeChangeoClone(db, text_fields=c("sample_id", "c_call"),
                          num_fields="duplicate_count")

# Run PHYLIP and process output
phylip_exec <- "~/apps/phylip-3.695/bin/dnapars"
graph <- buildPhylipLineage(clone, phylip_exec, rm_temp=TRUE)

# Plot graph with a tree layout
library(igraph)
plot(graph, layout=layout_as_tree, vertex.label=V(graph)$c_call,
     vertex.size=50, edge.arrow.mode=0, vertex.color="grey80")

# To consider each indel event as a mutation, change the masking character
# and distance matrix
clone <- makeChangeoClone(db, text_fields=c("sample_id", "c_call"),
                          num_fields="duplicate_count", mask_char="-")
graph <- buildPhylipLineage(clone, phylip_exec, dist_mat=getDNAMatrix(gap=-1),
                            rm_temp=TRUE)

## End(Not run)
```

---

bulk — *Calculates the average bulkiness of amino acid sequences*

Description

bulk calculates the average bulkiness score of amino acid sequences. Non-informative positions are excluded, where non-informative is defined as any character in `c("X","-",".","*")`.

Usage

```
bulk(seq, bulkiness = NULL)
```

Arguments

seq	vector of strings containing amino acid sequences.
bulkiness	named numerical vector defining bulkiness scores for each amino acid, where names are single-letter amino acid character codes. If NULL, then the Zimmerman et al, 1968 scale is used.

Value

A vector of bulkiness scores for the sequence(s).

References

1. Zimmerman JM, Eliezer N, Simha R. The characterization of amino acid sequences in proteins by statistical methods. J Theor Biol 21, 170-201 (1968).

See Also

For additional size related indices see aaindex.

Examples

```
# Default bulkiness scale
seq <- c("CARDRSTPWRRGIASTTVRTSW", "XXTQMYVRT")
bulk(seq)

# Use the Grantham, 1974 side chain volumn scores from the seqinr package
library(seqinr)
data(aaindex)
x <- aaindex[["GRAR740103"]]$I
# Rename the score vector to use single-letter codes
names(x) <- translateStrings(names(x), ABBREV_AA)
# Calculate average volume
bulk(seq, bulkiness=x)
```

---

calcCoverage *Calculate sample coverage*

Description

calcCoverage calculates the sample coverage estimate, a measure of sample completeness, for varying orders using the method of Chao et al, 2015, falling back to the Chao1 method in the first order case.

Usage

```
calcCoverage(x, r = 1)
```

Arguments

x	numeric vector of abundance counts.
r	coverage order to calculate.

Value

The sample coverage of the given order `r`.

References

1. Chao A. Nonparametric Estimation of the Number of Classes in a Population. Scand J Stat. 1984 11, 265270.
2. Chao A, et al. Unveiling the species-rank abundance distribution by generalizing the Good-Turing sample coverage theory. Ecology. 2015 96, 11891201.

See Also

Used by alphaDiversity.

Examples

```
# Calculate clone sizes
clones <- countClones(ExampleDb, groups="sample_id")

# Calculate 1first order coverage for a single sample
calcCoverage(clones$seq_count[clones$sample_id == "+7d"])
```

`calcDiversity` *Calculate the diversity index*

Description

`calcDiversity` calculates the clonal diversity index for a vector of diversity orders.

Usage

```
calcDiversity(p, q)
```

Arguments

p	numeric vector of clone (species) counts or proportions.
q	numeric vector of diversity orders.

Details

This method, proposed by Hill (Hill, 1973), quantifies diversity as a smooth function (D) of a single parameter q. Special cases of the generalized diversity index correspond to the most popular diversity measures in ecology: species richness ($q = 0$), the exponential of the Shannon-Weiner index (q approaches 1), the inverse of the Simpson index ($q = 2$), and the reciprocal abundance of the largest clone (q approaches $+\infty$). At $q = 0$ different clones weight equally, regardless of their size. As the parameter q increase from 0 to $+\infty$ the diversity index (D) depends less on rare clones and more on common (abundant) ones, thus encompassing a range of definitions that can be visualized as a single curve.

Values of $q < 0$ are valid, but are generally not meaningful. The value of D at $q = 1$ is estimated by D at $q = 0.9999$.

Value

A vector of diversity scores D for each q.

References

1. Hill M. Diversity and evenness: a unifying notation and its consequences. Ecology. 1973 54(2):427-32.

See Also

Used by alphaDiversity.

Examples

```
# May define p as clonal member counts
p <- c(1, 1, 3, 10)
q <- c(0, 1, 2)
calcDiversity(p, q)

# Or proportional abundance
p <- c(1/15, 1/15, 1/5, 2/3)
calcDiversity(p, q)
```

ChangeoClone-class *S4 class defining a clone*

Description

ChangeoClone defines a common data structure for perform lineage recontruction from Change-O data.

Slots

`data` data.frame containing sequences and annotations. Contains the columns `SEQUENCE_ID` and `SEQUENCE`, as well as any additional sequence-specific annotation columns.

`clone` string defining the clone identifier.

`germline` string containing the germline sequence for the clone.

`v_gene` string defining the V segment gene call.

`j_gene` string defining the J segment gene call.

`junc_len` numeric junction length (nucleotide count).

See Also

See makeChangeoClone and buildPhylipLineage for use.

`charge` *Calculates the net charge of amino acid sequences.*

Description

`charge` calculates the net charge of amino acid sequences using the method of Moore, 1985, with exclusion of the C-terminus and N-terminus charges.

Usage

```
charge(seq, pH = 7.4, pK = NULL, normalize = FALSE)
```

Arguments

`seq`	vector strings defining of amino acid sequences.
`pH`	environmental pH.
`pK`	named vector defining pK values for each charged amino acid, where names are the single-letter amino acid character codes `c("R","H","K","D","E","C","Y")`. If `NULL`, then the EMBOSS scale is used.
`normalize`	if `TRUE` then divide the net charge of each amino acid sequence by the number of informative positions. Non-informative position are defined by the presence any character in `c("X","-",".","*")`. If `FALSE` then return the raw net charge.

Value

A vector of net charges for the sequence(s).

References

1. Moore DS. Amino acid and peptide net charges: A simple calculational procedure. Biochem Educ. 13, 10-11 (1985).
2. http://emboss.sourceforge.net/apps/cvs/emboss/apps/iep.html

See Also

For additional pK scales see pK.

Examples

```
seq <- c("CARDRSTPWRRGIASTTVRTSW", "XXTQMYVRT")
# Unnormalized charge
charge(seq)
# Normalized charge
charge(seq, normalize=TRUE)

# Use the Murray et al, 2006 scores from the seqinr package
library(seqinr)
data(pK)
x <- setNames(pK[["Murray"]], rownames(pK))
# Calculate charge
charge(seq, pK=x)
```

---

checkColumns *Check data.frame for valid columns and issue message if invalid*

Description

Check data.frame for valid columns and issue message if invalid

Usage

```
checkColumns(data, columns, logic = c("all", "any"))
```

Arguments

`data`	data.frame to check.
`columns`	vector of column names to check.
`logic`	one of `"all"` or `"any"` controlling whether all, or at least one, of the columns must be valid, respectively.

Value

`TRUE` if columns are valid and a string message if not.

Examples

```
df <- data.frame(A=1:3, B=4:6, C=rep(NA, 3))
checkColumns(df, c("A", "B"), logic="all")
checkColumns(df, c("A", "B"), logic="any")
checkColumns(df, c("A", "C"), logic="all")
checkColumns(df, c("A", "C"), logic="any")
```

```
checkColumns(df, c("A", "D"), logic="all")
checkColumns(df, c("A", "D"), logic="any")
```

`collapseDuplicates` *Remove duplicate DNA sequences and combine annotations*

Description

`collapseDuplicates` identifies duplicate DNA sequences, allowing for ambiguous characters, removes the duplicate entries, and combines any associated annotations.

Usage

```
collapseDuplicates(
  data,
  id = "sequence_id",
  seq = "sequence_alignment",
  text_fields = NULL,
  num_fields = NULL,
  seq_fields = NULL,
  add_count = FALSE,
  ignore = c("N", "-", ".", "?"),
  sep = ",",
  dry = FALSE,
  verbose = FALSE
)
```

Arguments

`data`	data.frame containing Change-O columns. The data.frame must contain, at a minimum, a unique identifier column and a column containg a character vector of DNA sequences.
`id`	name of the column containing sequence identifiers.
`seq`	name of the column containing DNA sequences.
`text_fields`	character vector of textual columns to collapse. The textual annotations of duplicate sequences will be merged into a single string with each unique value alphabetized and delimited by `sep`.
`num_fields`	vector of numeric columns to collapse. The numeric annotations of duplicate sequences will be summed.
`seq_fields`	vector of nucletoide sequence columns to collapse. The sequence with the fewest numer of non-informative characters will be retained. Where a non-informative character is one of `c("N","-",".","?")`. Note, this is distinct from the `seq` parameter which is used to determine duplicates.
`add_count`	if `TRUE` add the column `collpase_count` that indicates the number of sequences that were collapsed to build each unique entry.

ignore	vector of characters to ignore when testing for equality.
sep	character to use for delimiting collapsed annotations in the text_fields columns. Defines both the input and output delimiter.
dry	if TRUE perform dry run. Only labels the sequences without collapsing them.
verbose	if TRUE report the number input, discarded and output sequences; if FALSE process sequences silently.

Details

collapseDuplicates identifies duplicate sequences in the seq column by testing for character identity, with consideration of IUPAC ambiguous nucleotide codes. A cluster of sequences are considered duplicates if they are all equivalent, and no member of the cluster is equivalent to a sequence in a different cluster.

Textual annotations, specified by text_fields, are collapsed by taking the unique set of values within in each duplicate cluster and delimiting those values by sep. Numeric annotations, specified by num_fields, are collapsed by summing all values in the duplicate cluster. Sequence annotations, specified by seq_fields, are collapsed by retaining the first sequence with the fewest number of N characters.

Columns that are not specified in either text_fields, num_fields, or seq_fields will be retained, but the value will be chosen from a random entry amongst all sequences in a cluster of duplicates.

An ambiguous sequence is one that can be assigned to two different clusters, wherein the ambiguous sequence is equivalent to two sequences which are themselves non-equivalent. Ambiguous sequences arise due to ambiguous characters at positions that vary across sequences, and are discarded along with their annotations when dry=FALSE. Thus, ambiguous sequences are removed as duplicates of some sequence, but do not create a potential false-positive annotation merger. Ambiguous sequences are not included in the collapse_count annotation that is added when add_count=TRUE.

If dry=TRUE sequences will not be removed from the input. Instead, the following columns will be appended to the input defining the collapse action that would have been performed in the dry=FALSE case.

- collapse_id: an identifer for the group of identical sequences.
- collapse_class: string defining how the sequence matches to the other in the set. one of "duplicated" (has duplicates), "unique" (no duplicates), "ambiguous_duplicate" (no duplicates after ambiguous sequences are removed), or "ambiguous" (matches multiple non-duplicate sequences).
- collapse_pass: TRUE for the sequences that would be retained.

Value

A modified data data.frame with duplicate sequences removed and annotation fields collapsed if dry=FALSE. If dry=TRUE, sequences will be labeled with the collapse action, but the input will be otherwise unmodifed (see Details).

See Also

Equality is tested with seqEqual and pairwiseEqual. For IUPAC ambiguous character codes see IUPAC_DNA.

Examples

```
# Example data.frame
db <- data.frame(sequence_id=LETTERS[1:4],
                 sequence_alignment=c("CCCCTGGG", "CCCCTGGN", "NAACTGGN", "NNNCTGNN"),
                 c_call=c("IGHM", "IGHG", "IGHG", "IGHA"),
                 sample_id=c("S1", "S1", "S2", "S2"),
                 duplicate_count=1:4,
                 stringsAsFactors=FALSE)

# Annotations are not parsed if neither text_fields nor num_fields is specified
# The retained sequence annotations will be random
collapseDuplicates(db, verbose=TRUE)

# Unique text_fields annotations are combined into a single string with ","
# num_fields annotations are summed
# Ambiguous duplicates are discarded
collapseDuplicates(db, text_fields=c("c_call", "sample_id"), num_fields="duplicate_count",
                   verbose=TRUE)

# Use alternate delimiter for collapsing textual annotations
collapseDuplicates(db, text_fields=c("c_call", "sample_id"), num_fields="duplicate_count",
                   sep="/", verbose=TRUE)

# Add count of duplicates
collapseDuplicates(db, text_fields=c("c_call", "sample_id"), num_fields="duplicate_count",
                   add_count=TRUE, verbose=TRUE)

# Masking ragged ends may impact duplicate removal
db$sequence_alignment <- maskSeqEnds(db$sequence_alignment)
collapseDuplicates(db, text_fields=c("c_call", "sample_id"), num_fields="duplicate_count",
                   add_count=TRUE, verbose=TRUE)
```

---

combineIgphyml *Combine IgPhyML object parameters into a dataframe*

Description

combineIgphyml combines IgPhyML object parameters into a data.frame.

Usage

```
combineIgphyml(iglist, format = c("wide", "long"))
```

Arguments

iglist	list of igphyml objects returned by readIgphyml. Each must have an id column in its param attribute, which can be added automatically using the id option of readIgphyml.

`format` string specifying whether each column of the resulting data.frame should represent a parameter (`wide`) or if there should only be three columns; i.e. id, varable, and value (`long`).

Details

`combineIgphyml` combines repertoire-wide parameter estimates from mutliple igphyml objects produced by readIgphyml into a dataframe that can be easily used for plotting and other hypothesis testing analyses.

All igphyml objects used must have an "id" column in their `param` attribute, which can be added automatically from the `id` flag of `readIgphyml`.

Value

A data.frame containing HLP model parameter estimates for all igphyml objects. Only parameters shared among all objects will be returned.

References

1. Hoehn KB, Lunter G, Pybus OG - A Phylogenetic Codon Substitution Model for Antibody Lineages. Genetics 2017 206(1):417-427 https://doi.org/10.1534/genetics.116.196303
2. Hoehn KB, Vander Heiden JA, Zhou JQ, Lunter G, Pybus OG, Kleinstein SHK - Repertoire-wide phylogenetic models of B cell molecular evolution reveal evolutionary signatures of aging and vaccination. bioRxiv 2019 https://doi.org/10.1101/558825

See Also

readIgphyml

Examples

```
## Not run:
   # Read in and combine two igphyml runs
   s1 <- readIgphyml("IB+7d_lineages_gy.tsv_igphyml_stats_hlp.tab", id="+7d")
   s2 <- readIgphyml("IB+7d_lineages_gy.tsv_igphyml_stats_hlp.tab", id="s2")
   combineIgphyml(list(s1, s2))

## End(Not run)
```

---

countClones *Tabulates clones sizes*

Description

`countClones` determines the number of sequences and total copy number of clonal groups.

Usage

```
countClones(
  data,
  groups = NULL,
  copy = NULL,
  clone = "clone_id",
  remove_na = TRUE
)
```

Arguments

`data`	data.frame with columns containing clonal assignments.
`groups`	character vector defining `data` columns containing grouping variables. If `groups=NULL`, then do not group data.
`copy`	name of the `data` column containing copy numbers for each sequence. If this value is specified, then total copy abundance is determined by the sum of copy numbers within each clonal group.
`clone`	name of the `data` column containing clone identifiers.
`remove_na`	removes rows with `NA` values in the clone column if `TRUE` and issues a warning. Otherwise, keeps those rows and considers `NA` as a clone in the final counts and relative abundances.

Value

A data.frame summarizing clone counts and frequencies with columns:

- `clone_id`: clone identifier. This is the default column name, specified with `clone='clone_id'`. If the function call uses Change-O formatted data and `clone='CLONE'`, this column will have name `CLONE`.
- `seq_count`: total number of sequences for the clone.
- `seq_freq`: frequency of the clone as a fraction of the total number of sequences within each group.
- `copy_count`: sum of the copy counts in the `copy` column. Only present if the `copy` argument is specified.
- `copy_freq`: frequency of the clone as a fraction of the total copy number within each group. Only present if the `copy` argument is specified.

Also includes additional columns specified in the `groups` argument.

Examples

```
# Without copy numbers
clones <- countClones(ExampleDb, groups="sample_id")

# With copy numbers and multiple groups
clones <- countClones(ExampleDb, groups=c("sample_id", "c_call"), copy="duplicate_count")
```

countGenes — *Tabulates V(D)J allele, gene or family usage.*

Description

Determines the count and relative abundance of V(D)J alleles, genes or families within groups.

Usage

```
countGenes(
  data,
  gene,
  groups = NULL,
  copy = NULL,
  clone = NULL,
  fill = FALSE,
  mode = c("gene", "allele", "family", "asis"),
  remove_na = TRUE
)
```

Arguments

`data`	data.frame with AIRR-format or Change-O style columns.
`gene`	column containing allele assignments. Only the first allele in the column will be considered when `mode` is "gene", "family" or "allele". The value will be used as it is with `mode="asis"`.
`groups`	columns containing grouping variables. If `NULL` do not group.
`copy`	name of the `data` column containing copy numbers for each sequence. If this value is specified, then total copy abundance is determined by the sum of copy numbers within each gene. This argument is ignored if `clone` is specified.
`clone`	name of the `data` column containing clone identifiers for each sequence. If this value is specified, then one gene will be considered for each clone. Note, this is accomplished by using the most common gene within each `clone` identifier. As such, ambiguous alleles within a clone will not be accurately represented.
`fill`	logical of `c(TRUE,FALSE)` specifying when if groups (when specified) lacking a particular gene should be counted as 0 if TRUE or not (omitted)
`mode`	one of `c("gene","family","allele","asis")` defining the degree of specificity regarding allele calls. Determines whether to return counts for genes (calling `getGene`), families (calling `getFamily`), alleles (calling `getAllele`) or using the value as it is in the column `gene`, without any processing.
`remove_na`	removes rows with `NA` values in the gene column if `TRUE` and issues a warning. Otherwise, keeps those rows and considers `NA` as a gene in the final counts and relative abundances.

Value

A data.frame summarizing family, gene or allele counts and frequencies with columns:

- gene: name of the family, gene or allele.
- seq_count: total number of sequences for the gene.
- seq_freq: frequency of the gene as a fraction of the total number of sequences within each grouping.
- copy_count: sum of the copy counts in the copy column. for each gene. Only present if the copy argument is specified.
- copy_freq: frequency of the gene as a fraction of the total copy number within each group. Only present if the copy argument is specified.
- clone_count: total number of clones for the gene. Only present if the clone argument is specified.
- clone_freq: frequency of the gene as a fraction of the total number of clones within each grouping. Only present if the clone argument is specified.

Additional columns defined by the groups argument will also be present.

Examples

```
# Without copy numbers
genes <- countGenes(ExampleDb, gene="v_call", groups="sample_id", mode="family")
genes <- countGenes(ExampleDb, gene="v_call", groups="sample_id", mode="gene")
genes <- countGenes(ExampleDb, gene="v_call", groups="sample_id", mode="allele")

# With copy numbers and multiple groups
genes <- countGenes(ExampleDb, gene="v_call", groups=c("sample_id", "c_call"),
                    copy="duplicate_count", mode="family")

# Count by clone
genes <- countGenes(ExampleDb, gene="v_call", groups=c("sample_id", "c_call"),
                    clone="clone_id", mode="family")

# Count absent genes
genes <- countGenes(ExampleDb, gene="v_call", groups="sample_id",
                    mode="allele", fill=TRUE)
```

---

countPatterns *Count sequence patterns*

Description

countPatterns counts the fraction of times a set of character patterns occur in a set of sequences.

Usage

```
countPatterns(seq, patterns, nt = TRUE, trim = FALSE, label = "region")
```

Arguments

seq	character vector of either DNA or amino acid sequences.
patterns	list of sequence patterns to count in each sequence. If the list is named, then names will be assigned as the column names of output data.frame.
nt	if TRUE then seq are DNA sequences and and will be translated before performing the pattern search.
trim	if TRUE remove the first and last codon or amino acid from each sequence before the pattern search. If FALSE do not modify the input sequences.
label	string defining a label to add as a prefix to the output column names.

Value

A data.frame containing the fraction of times each sequence pattern was found.

Examples

```
seq <- c("TGTCAACAGGCTAACAGTTTCCGGACGTTC",
         "TGTCAGCAATATTATATTGCTCCCTTCACTTTC",
         "TGTCAAAAGTATAACAGTGCCCCCTGGACGTTC")
patterns <- c("A", "V", "[LI]")
names(patterns) <- c("arg", "val", "iso_leu")
countPatterns(seq, patterns, trim=TRUE, label="cdr3")
```

---

cpuCount *Available CPU cores*

Description

cpuCount determines the number of CPU cores available.

Usage

```
cpuCount()
```

Value

Count of available cores. Returns 1 if undeterminable.

Examples

```
cpuCount()
```

DEFAULT_COLORS *Default colors*

Description

Default color palettes for DNA characters, Ig isotypes, and TCR chains.

Usage

```
DNA_COLORS

IG_COLORS

TR_COLORS
```

Format

Named character vectors with hexcode colors as values.

- `DNA_COLORS`: DNA character colors `c("A","C","G","T")`.
- `IG_COLORS`: Ig isotype colors `c("IGHA","IGHD","IGHE","IGHG","IGHM","IGHK","IGHL")`.
- `TR_COLORS`: TCR chain colors `c("TRA","TRB","TRD","TRG")`.

An object of class `character` of length 4.

An object of class `character` of length 7.

An object of class `character` of length 4.

Examples

```
# IG_COLORS as an isotype color set for ggplot
isotype <- c("IGHG", "IGHM", "IGHM", "IGHA")
db <- data.frame(x=1:4, y=1:4, iso=isotype)
g1 <- ggplot(db, aes(x=x, y=y, color=iso)) +
    scale_color_manual(name="Isotype", values=IG_COLORS) +
    geom_point(size=10)
plot(g1)

# DNA_COLORS to translate nucleotide values to a vector of colors
# for use in base graphics plots
seq <- c("A", "T", "T", "C")
colors <- translateStrings(seq, setNames(names(DNA_COLORS), DNA_COLORS))
plot(1:4, 1:4, col=colors, pch=16, cex=6)
```

`DiversityCurve-class` *S4 class defining a diversity curve*

Description

`DiversityCurve` defines diversity (D) scores over multiple diversity orders (Q).

Usage

```
## S4 method for signature 'DiversityCurve'
print(x)

## S4 method for signature 'DiversityCurve,missing'
plot(x, y, ...)

## S4 method for signature 'DiversityCurve,numeric'
plot(x, y, ...)
```

Arguments

`x`	DiversityCurve object
`y`	diversity order to plot (q).
`...`	arguments to pass to plotDiversityCurve or plotDiversityTest.

Slots

`diversity` data.frame defining the diversity curve with the following columns:

- `group`: group label.
- `q`: diversity order.
- `d`: mean diversity index over all bootstrap realizations.
- `d_sd`: standard deviation of the diversity index over all bootstrap realizations.
- `d_lower`: diversity lower confidence inverval bound.
- `d_upper`: diversity upper confidence interval bound.
- `e`: evenness index calculated as `D` divided by `D` at `Q=0`.
- `e_lower`: evenness lower confidence inverval bound.
- `e_upper`: eveness upper confidence interval bound.

`tests` data.frame describing the significance test results with columns:

- `test`: string listing the two groups tested.
- `delta_mean`: mean of the D bootstrap delta distribution for the test.
- `delta_sd`: standard deviation of the D bootstrap delta distribution for the test.
- `pvalue`: p-value for the test.

`group_by` string specifying the name of the grouping column in diversity calculation.

`groups` vector specifying the names of unique groups in group column in diversity calculation.

`method` string specifying the type of diversity calculated.

`q` vector of diversity hill diversity indices used for computing diversity.

`n` numeric vector indication the number of sequences sampled in each group.

`ci` confidence interval defining the upper and lower bounds (a value between 0 and 1).

`EdgeTest-class` *S4 class defining edge significance*

Description

`EdgeTest` defines the significance of parent-child annotation enrichment.

Usage

```
## S4 method for signature 'EdgeTest'
print(x)

## S4 method for signature 'EdgeTest,missing'
plot(x, y, ...)
```

Arguments

`x`	EdgeTest object.
`y`	ignored.
`...`	arguments to pass to plotEdgeTest.

Slots

`tests` data.frame describing the significance test results with columns:

- `parent`: parent node annotation.
- `child`: child node annotation
- `count`: count of observed edges with the given parent-child annotation set.
- `expected`: mean count of expected edges for the given parent-child relationship.
- `pvalue`: one-sided p-value for the hypothesis that the observed edge abundance is greater than expected.

`permutations` data.frame containing the raw permutation test data with columns:

- `parent`: parent node annotation.
- `child`: child node annotation
- `count`: count of edges with the given parent-child annotation set.
- `iter`: numerical index define which permutation realization each observation corresponds to.

`nperm` number of permutation realizations.

estimateAbundance *Estimates the complete clonal relative abundance distribution*

Description

estimateAbundance estimates the complete clonal relative abundance distribution and confidence intervals on clone sizes using bootstrapping.

Usage

```
estimateAbundance(
  data,
  clone = "clone_id",
  copy = NULL,
  group = NULL,
  min_n = 30,
  max_n = NULL,
  uniform = TRUE,
  ci = 0.95,
  nboot = 200,
  progress = FALSE
)
```

Arguments

data	data.frame with Change-O style columns containing clonal assignments.
clone	name of the data column containing clone identifiers.
copy	name of the data column containing copy numbers for each sequence. If copy=NULL (the default), then clone abundance is determined by the number of sequences. If a copy column is specified, then clone abundances is determined by the sum of copy numbers within each clonal group.
group	name of the data column containing group identifiers. If NULL then no grouping is performed and the group column of the output will contain the value NA for each row.
min_n	minimum number of observations to sample. A group with less observations than the minimum is excluded.
max_n	maximum number of observations to sample. If NULL then no maximum is set.
uniform	if TRUE then uniformly resample each group to the same number of observations. If FALSE then allow each group to be resampled to its original size or, if specified, max_size.
ci	confidence interval to calculate; the value must be between 0 and 1.
nboot	number of bootstrap realizations to generate.
progress	if TRUE show a progress bar.

Value

A AbundanceCurve object summarizing the abundances.

References

1. Chao A. Nonparametric Estimation of the Number of Classes in a Population. Scand J Stat. 1984 11, 265270.
2. Chao A, et al. Rarefaction and extrapolation with Hill numbers: A framework for sampling and estimation in species diversity studies. Ecol Monogr. 2014 84:45-67.
3. Chao A, et al. Unveiling the species-rank abundance distribution by generalizing the Good-Turing sample coverage theory. Ecology. 2015 96, 11891201.

See Also

See plotAbundanceCurve for plotting of the abundance distribution. See alphaDiversity for a similar application to clonal diversity.

Examples

```
abund <- estimateAbundance(ExampleDb, group="sample_id", nboot=100)
```

Example10x — *Small example 10x Genomics Ig V(D)J sequences from CD19+ B cells isolated from PBMCs of a healthy human donor. Down-sampled from data provided by 10x Genomics under a Creative Commons Attribute license, and processed with their Cell Ranger pipeline.*

Description

Small example 10x Genomics Ig V(D)J sequences from CD19+ B cells isolated from PBMCs of a healthy human donor. Down-sampled from data provided by 10x Genomics under a Creative Commons Attribute license, and processed with their Cell Ranger pipeline.

Usage

```
Example10x
```

Format

A data.frame with the following AIRR style columns:

- `sequence_id`: Sequence identifier
- `sequence_alignment`: IMGT-gapped observed sequence.
- `germline_alignment`: IMGT-gapped germline sequence.
- `v_call`: V region allele assignments.

- `d_call`: D region allele assignments.
- `j_call`: J region allele assignments.
- `c_call`: Isotype (C region) assignment.
- `junction`: Junction region sequence.
- `junction_length`: Length of the junction region in nucleotides.
- `np1_length`: Combined length of the N and P regions proximal to the V region.
- `np2_length`: Combined length of the N and P regions proximal to the J region.
- `umi_count`: Number of unique molecular identifies atttributed to sequence.
- `cell_id`: Cell identifier.
- `locus`: Genomic locus of sequence.

References

1. Data source: https://support.10xgenomics.com/single-cell-vdj/datasets/2.2.0/vdj_v1_hs_cd19_b
2. License: https://creativecommons.org/licenses/by/4.0/

`ExampleDb` *Example AIRR database*

Description

A small example database subset from Laserson and Vigneault et al, 2014.

Usage

```
ExampleDb
```

Format

A data.frame with the following AIRR style columns:

- `sequence_id`: Sequence identifier
- `sequence_alignment`: IMGT-gapped observed sequence.
- `germline_alignment`: IMGT-gapped germline sequence.
- `germline_alignment_d_mask`: IMGT-gapped germline sequence with N, P and D regions masked.
- `v_call`: V region allele assignments.
- `v_call_genotyped`: TIgGER corrected V region allele assignment.
- `d_call`: D region allele assignments.
- `j_call`: J region allele assignments.
- `c_call`: Isotype (C region) assignment.
- `junction`: Junction region sequence.

- junction_length: Length of the junction region in nucleotides.
- np1_length: Combined length of the N and P regions proximal to the V region.
- np2_length: Combined length of the N and P regions proximal to the J region.
- duplicate_count: Copy count (number of duplicates) of the sequence.
- clone_id: Change-O assignment clonal group identifier.
- sample_id: Sample identifier. Time in relation to vaccination.

References

1. Laserson U and Vigneault F, et al. High-resolution antibody dynamics of vaccine-induced immune responses. Proc Natl Acad Sci USA. 2014 111:4928-33.

See Also

ExampleDbChangeo ExampleTrees

ExampleDbChangeo *Example Change-O database*

Description

A small example database subset from Laserson and Vigneault et al, 2014.

Usage

```
ExampleDbChangeo
```

Format

A data.frame with the following Change-O style columns:

- SEQUENCE_ID: Sequence identifier
- SEQUENCE_IMGT: IMGT-gapped observed sequence.
- GERMLINE_IMGT_D_MASK: IMGT-gapped germline sequence with N, P and D regions masked.
- V_CALL: V region allele assignments.
- V_CALL_GENOTYPED: TIgGER corrected V region allele assignment.
- D_CALL: D region allele assignments.
- J_CALL: J region allele assignments.
- JUNCTION: Junction region sequence.
- JUNCTION_LENGTH: Length of the junction region in nucleotides.
- NP1_LENGTH: Combined length of the N and P regions proximal to the V region.
- NP2_LENGTH: Combined length of the N and P regions proximal to the J region.
- SAMPLE: Sample identifier. Time in relation to vaccination.
- ISOTYPE: Isotype assignment.
- DUPCOUNT: Copy count (number of duplicates) of the sequence.
- CLONE: Change-O assignment clonal group identifier.

References

1. Laserson U and Vigneault F, et al. High-resolution antibody dynamics of vaccine-induced immune responses. Proc Natl Acad Sci USA. 2014 111:4928-33.

See Also

ExampleDb ExampleTrees

`ExampleTrees` *Example Ig lineage trees*

Description

A set of Ig lineage trees generated from the `ExampleDb` file, subset to only those trees with at least four nodes.

Usage

```
ExampleTrees
```

Format

A list of igraph objects output by buildPhylipLineage. Each node of each tree has the following annotations (vertex attributes):

- `sample_id`: Sample identifier(s). Time in relation to vaccination.
- `c_call`: Isotype assignment(s).
- `duplication_count`: Copy count (number of duplicates) of the sequence.

See Also

ExampleTrees

`extractVRegion` *Extracts FWRs and CDRs from IMGT-gapped sequences*

Description

`extractVRegion` extracts the framework and complementarity determining regions of the V segment for IMGT-gapped immunoglobulin (Ig) nucleotide sequences according to the IMGT numbering scheme.

Usage

```
extractVRegion(sequences, region = c("fwr1", "cdr1", "fwr2", "cdr2", "fwr3"))
```

Arguments

`sequences`	character vector of IMGT-gapped nucleotide sequences.
`region`	string defining the region(s) of the V segment to extract. May be a single region or multiple regions (as a vector) from `c("fwr1","cdr1","fwr2","cdr2","fwr3")`. By default, all regions will be returned.

Value

If only one region is specified in the `region` argument, a character vector of the extracted subsequences will be returned. If multiple regions are specified, then a character matrix will be returned with columns corresponding to the specified regions and a row for each entry in `sequences`.

References

1. Lefranc M-P, et al. IMGT unique numbering for immunoglobulin and T cell receptor variable domains and Ig superfamily V-like domains. Dev Comp Immunol. 2003 27(1):55-77.

See Also

IMGT-gapped region boundaries are defined in IMGT_REGIONS.

Examples

```
# Assign example clone
clone <- subset(ExampleDb, clone_id == 3138)

# Get all regions
extractVRegion(clone$sequence_alignment)

# Get single region
extractVRegion(clone$sequence_alignment, "fwr1")

# Get all CDRs
extractVRegion(clone$sequence_alignment, c("cdr1", "cdr2"))

# Get all FWRs
extractVRegion(clone$sequence_alignment, c("fwr1", "fwr2", "fwr3"))
```

`getAAMatrix` *Build an AA distance matrix*

Description

`getAAMatrix` returns a Hamming distance matrix for IUPAC ambiguous amino acid characters.

Usage

```
getAAMatrix(gap = 0)
```

Arguments

`gap`	value to assign to characters in the set `c("-",".")`.

Value

A `matrix` of amino acid character distances with row and column names indicating the character pair.

See Also

Creates an amino acid distance matrix for seqDist. See getDNAMatrix for nucleotide distances.

Examples

```
getAAMatrix()
```

`getDNAMatrix` *Build a DNA distance matrix*

Description

`getDNAMatrix` returns a Hamming distance matrix for IUPAC ambiguous DNA characters with modifications for gap, `c("-",".")`, and missing, `c("?")`, character values.

Usage

```
getDNAMatrix(gap = -1)
```

Arguments

`gap`	value to assign to characters in the set `c("-",".")`.

Value

A `matrix` of DNA character distances with row and column names indicating the character pair. By default, distances will be either 0 (equivalent), 1 (non-equivalent or missing), or -1 (gap).

See Also

Creates DNA distance matrix for seqDist. See getAAMatrix for amino acid distances.

Examples

```
# Set gap characters to Inf distance
# Distinguishes gaps from Ns
getDNAMatrix()

# Set gap characters to 0 distance
# Makes gap characters equivalent to Ns
getDNAMatrix(gap=0)
```

`getMRCA` *Retrieve the first non-root node of a lineage tree*

Description

`getMRCA` returns the set of lineage tree nodes with the minimum weighted or unweighted path length from the root (germline) of the lineage tree, allowing for exclusion of specific groups of nodes.

Usage

```
getMRCA(
  graph,
  path = c("distance", "steps"),
  root = "Germline",
  field = NULL,
  exclude = NULL
)
```

Arguments

`graph`	igraph object containing an annotated lineage tree.
`path`	string defining whether to use unweighted (steps) or weighted (distance) measures for determining the founder node set..
`root`	name of the root (germline) node.
`field`	annotation field to use for both unweighted path length exclusion and consideration as an MRCA node. If `NULL` do not exclude any nodes.
`exclude`	vector of annotation values in `field` to exclude from the potential MRCA set. If `NULL` do not exclude any nodes. Has no effect if `field=NULL`.

Value

A data.frame of the MRCA node(s) containing the columns:

- `name`: node name
- `steps`: path length as the number of nodes traversed
- `distance`: path length as the sum of edge weights

Along with additional columns corresponding to the annotations of the input graph.

See Also

Path lengths are determined with getPathLengths.

Examples

```
# Define example graph
graph <- ExampleTrees[[23]]

# Use unweighted path length and do not exclude any nodes
getMRCA(graph, path="steps", root="Germline")

# Exclude nodes without an isotype annotation and use weighted path length
getMRCA(graph, path="distance", root="Germline", field="c_call", exclude=NA)
```

---

getPathLengths *Calculate path lengths from the tree root*

Description

`getPathLengths` calculates the unweighted (number of steps) and weighted (distance) path lengths from the root of a lineage tree.

Usage

```
getPathLengths(graph, root = "Germline", field = NULL, exclude = NULL)
```

Arguments

`graph`	igraph object containing an annotated lineage tree.
`root`	name of the root (germline) node.
`field`	annotation field to use for exclusion of nodes from step count.
`exclude`	annotation values specifying which nodes to exclude from step count. If `NULL` consider all nodes. This does not affect the weighted (distance) path length calculation.

Value

A data.frame with columns:

- `name`: node name
- `steps`: path length as the number of nodes traversed
- `distance`: path length as the sum of edge weights

See Also

See buildPhylipLineage for generating input trees.

Examples

```
# Define example graph
graph <- ExampleTrees[[24]]

# Consider all nodes
getPathLengths(graph, root="Germline")

# Exclude nodes without an isotype annotation from step count
getPathLengths(graph, root="Germline", field="c_call", exclude=NA)
```

---

getPositionQuality — *Get a data.frame with sequencing qualities per position*

Description

getPositionQuality takes a data.frame with sequence quality scores in the form of a strings of comma separated numeric values, split the quality scores values by ",", and returns a data.frame with the values for each position.

Usage

```
getPositionQuality(
  data,
  sequence_id = "sequence_id",
  sequence = "sequence_alignment",
  quality_num = "quality_alignment_num"
)
```

Arguments

data	data.frame containing sequence data.
sequence_id	column in data with sequence identifiers.
sequence	column in data with sequence data.
quality_num	column in data with quality scores (as strings of numeric values, comma separated) for sequence.

Value

data with one additional field with masked sequences. The name of this field is created concatenating sequence and '_masked'.

See Also

readFastqDb and maskPositionsByQuality

Examples

```
db <- airr::read_rearrangement(system.file("extdata", "example_quality.tsv", package="alakazam"))
fastq_file <- system.file("extdata", "example_quality.fastq", package="alakazam")
db <- readFastqDb(db, fastq_file, quality_offset=-33)
head(getPositionQuality(db))
```

---

getSegment *Get Ig segment allele, gene and family names*

Description

getSegment performs generic matching of delimited segment calls with a custom regular expression. getAllele, getGene and getFamily extract the allele, gene and family names, respectively, from a character vector of immunoglobulin (Ig) or TCR segment allele calls in IMGT format.

Usage

```
getSegment(
  segment_call,
  segment_regex,
  first = TRUE,
  collapse = TRUE,
  strip_d = TRUE,
  omit_nl = FALSE,
  sep = ","
)

getAllele(
  segment_call,
  first = TRUE,
  collapse = TRUE,
  strip_d = TRUE,
  omit_nl = FALSE,
  sep = ","
)

getGene(
  segment_call,
  first = TRUE,
  collapse = TRUE,
  strip_d = TRUE,
  omit_nl = FALSE,
  sep = ","
)

getFamily(
  segment_call,
```

```
    first = TRUE,
    collapse = TRUE,
    strip_d = TRUE,
    omit_nl = FALSE,
    sep = ","
  )

  getLocus(
    segment_call,
    first = TRUE,
    collapse = TRUE,
    strip_d = TRUE,
    omit_nl = FALSE,
    sep = ","
  )

  getChain(
    segment_call,
    first = TRUE,
    collapse = TRUE,
    strip_d = TRUE,
    omit_nl = FALSE,
    sep = ","
  )
```

Arguments

segment_call	character vector containing segment calls delimited by commas.
segment_regex	string defining the segment match regular expression.
first	if TRUE return only the first call in segment_call; if FALSE return all calls delimited by commas.
collapse	if TRUE check for duplicates and return only unique segment assignments; if FALSE return all assignments (faster). Has no effect if first=TRUE.
strip_d	if TRUE remove the "D" from the end of gene annotations (denoting a duplicate gene in the locus); if FALSE do not alter gene names.
omit_nl	if TRUE remove non-localized (NL) genes from the result. Only applies at the gene or allele level.
sep	character defining both the input and output segment call delimiter.

Value

A character vector containing allele, gene or family names.

References

http://imgt.org

See Also

countGenes

Examples

```
# Light chain examples
kappa_call <- c("Homsap IGKV1D-39*01 F,Homsap IGKV1-39*02 F,Homsap IGKV1-39*01",
                "Homsap IGKJ5*01 F")

getAllele(kappa_call)
getAllele(kappa_call, first=FALSE)
getAllele(kappa_call, first=FALSE, strip_d=FALSE)

getGene(kappa_call)
getGene(kappa_call, first=FALSE)
getGene(kappa_call, first=FALSE, strip_d=FALSE)

getFamily(kappa_call)
getFamily(kappa_call, first=FALSE)
getFamily(kappa_call, first=FALSE, collapse=FALSE)
getFamily(kappa_call, first=FALSE, strip_d=FALSE)

getLocus(kappa_call)
getChain(kappa_call)

# Heavy chain examples
heavy_call <- c("Homsap IGHV1-69*01 F,Homsap IGHV1-69D*01 F",
                "Homsap IGHD1-1*01 F",
                "Homsap IGHJ1*01 F")

getAllele(heavy_call, first=FALSE)
getAllele(heavy_call, first=FALSE, strip_d=FALSE)

getGene(heavy_call, first=FALSE)
getGene(heavy_call, first=FALSE, strip_d=FALSE)

getFamily(heavy_call)
getLocus(heavy_call)
getChain(heavy_call)

# Filtering non-localized genes
nl_call <- c("IGHV3-NL1*01,IGHV3-30-3*01,IGHV3-30*01",
             "Homosap IGHV3-30*01 F,Homsap IGHV3-NL1*01 F",
             "IGHV1-NL1*01")

getAllele(nl_call, first=FALSE, omit_nl=TRUE)
getGene(nl_call, first=FALSE, omit_nl=TRUE)
getFamily(nl_call, first=FALSE, omit_nl=TRUE)

# Temporary designation examples
tmp_call <- c("IGHV9S3*01", "IGKV10S12*01")
```

```
getAllele(tmp_call)
getGene(tmp_call)
getFamily(tmp_call)
```

---

graphToPhylo *Convert a tree in igraph* `graph` *format to ape* `phylo` *format.*

Description

`graphToPhylo` a tree in igraph `graph` format to ape `phylo` format.

Usage

```
graphToPhylo(graph)
```

Arguments

`graph`	An igraph `graph` object.

Details

Convert from igraph `graph` object to ape `phylo` object. If `graph` object was previously rooted with the germline as the direct ancestor, this will re-attach the germline as a descendant node with a zero branch length to a new universal common ancestor (UCA) node and store the germline node ID in the `germid` attribute and UCA node number in the `uca` attribute. Otherwise these attributes will not be specified in the `phylo` object. Using `phyloToGraph(phylo,germline=phylo$germid)` creates a `graph` object with the germline back as the direct ancestor. Tip and internal node names are stored in the `tip.label` and `node.label` vectors, respectively.

Value

A `phylo` object representing the input tree. Tip and internal node names are stored in the `tip.label` and `node.label` vectors, respectively.

References

1. Hoehn KB, Lunter G, Pybus OG - A Phylogenetic Codon Substitution Model for Antibody Lineages. Genetics 2017 206(1):417-427 https://doi.org/10.1534/genetics.116.196303
2. Hoehn KB, Vander Heiden JA, Zhou JQ, Lunter G, Pybus OG, Kleinstein SHK - Repertoire-wide phylogenetic models of B cell molecular evolution reveal evolutionary signatures of aging and vaccination. bioRxiv 2019 https://doi.org/10.1101/558825

Examples

```
## Not run:
   library(igraph)
   library(ape)

   #convert to phylo
   phylo = graphToPhylo(graph)

   #plot tree using ape
   plot(phylo,show.node.label=TRUE)

   #store as newick tree
   write.tree(phylo,file="tree.newick")

   #read in tree from newick file
   phylo_r = read.tree("tree.newick")

   #convert to igraph
   graph_r = phyloToGraph(phylo_r,germline="Germline")

   #plot graph - same as before, possibly rotated
   plot(graph_r,layout=layout_as_tree)

## End(Not run)
```

---

gravy — *Calculates the hydrophobicity of amino acid sequences*

Description

`gravy` calculates the Grand Average of Hydrophobicity (gravy) index of amino acid sequences using the method of Kyte & Doolittle. Non-informative positions are excluded, where non-informative is defined as any character in `c("X","-",".","*")`.

Usage

```
gravy(seq, hydropathy = NULL)
```

Arguments

`seq`	vector of strings containing amino acid sequences.
`hydropathy`	named numerical vector defining hydropathy index values for each amino acid, where names are single-letter amino acid character codes. If `NULL`, then the Kyte & Doolittle scale is used.

Value

A vector of gravy scores for the sequence(s).

References

1. Kyte J, Doolittle RF. A simple method for displaying the hydropathic character of a protein. J Mol Biol. 157, 105-32 (1982).

See Also

For additional hydrophobicity indices see `aaindex`.

Examples

```
# Default scale
seq <- c("CARDRSTPWRRGIASTTVRTSW", "XXTQMYVRT")
gravy(seq)

# Use the Kidera et al, 1985 scores from the seqinr package
library(seqinr)
data(aaindex)
x <- aaindex[["KIDA850101"]]$I
# Rename the score vector to use single-letter codes
names(x) <- translateStrings(names(x), ABBREV_AA)
# Calculate hydrophobicity
gravy(seq, hydropathy=x)
```

---

gridPlot — *Plot multiple ggplot objects*

Description

Plots multiple ggplot objects in an equally sized grid.

Usage

```
gridPlot(..., ncol = 1)
```

Arguments

`...`	ggplot objects to plot.
`ncol`	number of columns in the plot.

References

Modified from: http://www.cookbook-r.com/Graphs/Multiple_graphs_on_one_page_(ggplot2)

See Also

ggplot.

groupGenes *Group sequences by gene assignment*

Description

groupGenes will group rows by shared V and J gene assignments, and optionally also by junction lengths. IGH:IGK/IGL, TRB:TRA, and TRD:TRG paired single-cell BCR/TCR sequencing and unpaired bulk sequencing (IGH, TRB, TRD chain only) are supported. In the case of ambiguous (multiple) gene assignments, the grouping may be specified to be a union across all ambiguous V and J gene pairs, analogous to single-linkage clustering (i.e., allowing for chaining).

Usage

```
groupGenes(
  data,
  v_call = "v_call",
  j_call = "j_call",
  junc_len = NULL,
  cell_id = NULL,
  locus = "locus",
  only_heavy = TRUE,
  first = FALSE
)
```

Arguments

data	data.frame containing sequence data.
v_call	name of the column containing the heavy/long chain V-segment allele calls.
j_call	name of the column containing the heavy/long chain J-segment allele calls.
junc_len	name of column containing the junction length. If NULL then 1-stage partitioning is perform considering only the V and J genes is performed. See Details for further clarification.
cell_id	name of the column containing cell identifiers or barcodes. If specified, grouping will be performed in single-cell mode with the behavior governed by the locus and only_heavy arguments. If set to NULL then the bulk sequencing data is assumed.
locus	name of the column containing locus information. Only applicable to single-cell data. Ignored if cell_id=NULL.
only_heavy	use only the IGH (BCR) or TRB/TRD (TCR) sequences for grouping. Only applicable to single-cell data. Ignored if cell_id=NULL.
first	if TRUE only the first call of the gene assignments is used. if FALSE the union of ambiguous gene assignments is used to group all sequences with any overlapping gene calls.

Details

To invoke single-cell mode the cell_id argument must be specified and the locus column must be correct. Otherwise, groupGenes will be run with bulk sequencing assumptions, using all input sequences regardless of the values in the locus column.

Values in the locus column must be one of c("IGH","IGI","IGK","IGL") for BCR or c("TRA","TRB","TRD","TRG") for TCR sequences. Otherwise, the function returns an error message and stops.

Under single-cell mode with paired chained sequences, there is a choice of whether grouping should be done by (a) using IGH (BCR) or TRB/TRD (TCR) sequences only or (b) using IGH plus IGK/IGL (BCR) or TRB/TRD plus TRA/TRG (TCR). This is governed by the only_heavy argument.

Specifying junc_len will force groupGenes to perform a 1-stage partitioning of the sequences/cells based on V gene, J gene, and junction length simultaneously. If junc_len=NULL (no column specified), then groupGenes performs only the first stage of a 2-stage partitioning in which sequences/cells are partitioned in the first stage based on V gene and J gene, and then in the second stage further splits the groups based on junction length (the second stage must be performed independently, as this only returns the first stage results).

In the input data, the v_call, j_call, cell_id, and locus columns, if present, must be of type character (as opposed to factor).

It is assumed that ambiguous gene assignments are separated by commas.

All rows containing NA values in any of the v_call, j_call, and junc_len (if junc_len != NULL) columns will be removed. A warning will be issued when a row containing an NA is removed.

Value

Returns a modified data.frame with disjoint union indices in a new vj_group column.

If junc_len is supplied, the grouping this vj_group will have been based on V, J, and junction length simultaneously. However, the output column name will remain vj_group.

The output v_call, j_call, cell_id, and locus columns will be converted to type character if they were of type factor in the input data.

Expectations for single-cell data

Single-cell paired chain data assumptions:

- every row represents a sequence (chain).
- heavy/long and light/short chains of the same cell are linked by cell_id.
- the value in locus column indicates whether the chain is the heavy/long or light/short chain.
- each cell possibly contains multiple heavy/long and/or light/short chains.
- every chain has its own V(D)J annotation, in which ambiguous V(D)J annotations, if any, are separated by a comma.

Single-cell example:

- A cell has 1 heavy chain and 2 light chains.
- There should be 3 rows corresponding to this cell.
- One of the light chains may have an ambiguous V annotation which looks like "Homsap IGKV1-39*01 F,Homsap IGKV1D-39*01 F".

Examples

```
# Group by genes
db <- groupGenes(ExampleDb)
head(db$vj_group)
```

IMGT_REGIONS *IMGT V-segment regions*

Description

A list defining the boundaries of V-segment framework regions (FWRs) and complementarity determining regions (CDRs) for IMGT-gapped immunoglobulin (Ig) nucleotide sequences according to the IMGT numbering scheme.

Usage

```
IMGT_REGIONS
```

Format

A list with regions named one of `c("fwr1","cdr1","fwr2","cdr2","fwr3")` with values containing a numeric vector of length two defining the `c(start,end)` positions of the named region.

References

http://imgt.org

---

isValidAASeq *Validate amino acid sequences*

Description

`isValidAASeq` checks that a set of sequences are valid non-ambiguous amino acid sequences. A sequence is considered valid if it contains only characters in the the non-ambiguous IUPAC character set or any characters in `c("X",".","-","*")`.

Usage

```
isValidAASeq(seq)
```

Arguments

seq	character vector of sequences to check.

Value

A logical vector with `TRUE` for each valid amino acid sequences and `FALSE` for each invalid sequence.

See Also

See ABBREV_AA for the set of non-ambiguous amino acid characters. See IUPAC_AA for the full set of ambiguous amino acid characters.

Examples

```
seq <- c("CARDRSTPWRRGIASTTVRTSW", "XXTQMYVR--XX", "CARJ", "10")
isValidAASeq(seq)
```

IUPAC_CODES *IUPAC ambiguous characters*

Description

A translation list mapping IUPAC ambiguous characters code to corresponding nucleotide amino acid characters.

Usage

```
IUPAC_DNA

IUPAC_AA

DNA_IUPAC
```

Format

A list with single character codes as names and values containing character vectors that define the set of standard characters that match to each each ambiguous character.

- `IUPAC_DNA`: DNA ambiguous character translations.
- `IUPAC_AA`: Amino acid ambiguous character translations.
- `DNA_IUPAC`: Ordered DNA to ambiguous characters

An object of class `list` of length 15.

An object of class `list` of length 25.

An object of class `list` of length 15.

junctionAlignment *Calculate junction region alignment properties*

Description

junctionAlignment determines the number of deleted germline nucleotides in the junction region and the number of V gene and J gene nucleotides in the CDR3.

Usage

```
junctionAlignment(
  data,
  germline_db,
  v_call = "v_call",
  d_call = "d_call",
  j_call = "j_call",
  v_germline_start = "v_germline_start",
  v_germline_end = "v_germline_end",
  d_germline_start = "d_germline_start",
  d_germline_end = "d_germline_end",
  j_germline_start = "j_germline_start",
  j_germline_end = "j_germline_end",
  np1_length = "np1_length",
  np2_length = "np2_length",
  junction = "junction",
  junction_length = "junction_length",
  sequence_alignment = "sequence_alignment"
)
```

Arguments

data	data.frame containing sequence data.
germline_db	reference germline database for the V, D and J genes. in data
v_call	V gene assignment column.
d_call	D gene assignment column.
j_call	J gene assignment column.
v_germline_start	column containing the start position of the alignment in the V reference germline.
v_germline_end	column containing the end position of the alignment in the V reference germline.
d_germline_start	column containing the start position of the alignment in the D reference germline.
d_germline_end	column containing the start position of the alignment in the D reference germline.
j_germline_start	column containing the start position of the alignment in the J reference germline.

`j_germline_end` column containing the start position of the alignment in the J reference germline.

`np1_length` combined length of the N and P regions between the V and D regions (heavy chain) or V and J regions (light chain).

`np2_length` combined length of the N and P regions between the D and J regions (heavy chain).

`junction` column containing the junction sequence.

`junction_length`
column containing the length of the junction region in nucleotides.

`sequence_alignment`
column containing the aligned sequence.

Value

A modified input `data.frame` with the following additional columns storing junction alignment information:

1. `e3v_length`: number of 3' V germline nucleotides deleted.
2. `e5d_length`: number of 5' D germline nucleotides deleted.
3. `e3d_length`: number of 3' D germline nucleotides deleted.
4. `e5j_length`: number of 5' J germline nucleotides deleted.
5. `v_cdr3_length`: number of sequence_alignment V nucleotides in the CDR3.
6. `j_cdr3_length`: number of sequence_alignment J nucleotides in the CDR3.

Examples

```
germline_db <- list(
"IGHV3-11*05"="CAGGTGCAGCTGGTGGAGTCTGGGGGA...GGCTTGGTCAAGCCTGGAGGGTCCCTGAGACT
CTCCTGTGCAGCCTCTGGATTCACCTTC............AGTGACTACTACATGAGCTGGATCCGCCAGGCTCCAG
GGAAGGGGCTGGAGTGGGTTTCATACATTAGTAGTAGT......AGTAGTTACACAAACTACGCAGACTCTGTGAAG
...GGCCGATTCACCATCTCCAGAGACAACGCCAAGAACTCACTGTATCTGCAAATGAACAGCCTGAGAGCCGAGGA
CACGGCCGTGTATTACTGTGCGAGAGA",
"IGHD3-10*01"="GTATTACTATGGTTCGGGGAGTTATTATAAC",
"IGHJ5*02"="ACAACTGGTTCGACCCCTGGGGCCAGGGAACCCTGGTCACCGTCTCCTCAG"
)

db <- junctionAlignment(SingleDb, germline_db)
```

`makeChangeoClone` *Generate a ChangeoClone object for lineage construction*

Description

`makeChangeoClone` takes a data.frame with AIRR or Change-O style columns as input and masks gap positions, masks ragged ends, removes duplicates sequences, and merges annotations associated with duplicate sequences. It returns a `ChangeoClone` object which serves as input for lineage reconstruction.

Usage

```
makeChangeoClone(
  data,
  id = "sequence_id",
  seq = "sequence_alignment",
  germ = "germline_alignment",
  v_call = "v_call",
  j_call = "j_call",
  junc_len = "junction_length",
  clone = "clone_id",
  mask_char = "N",
  max_mask = 0,
  pad_end = FALSE,
  text_fields = NULL,
  num_fields = NULL,
  seq_fields = NULL,
  add_count = TRUE,
  verbose = FALSE
)
```

Arguments

`data`	data.frame containing the AIRR or Change-O data for a clone. See Details for the list of required columns and their default values.
`id`	name of the column containing sequence identifiers.
`seq`	name of the column containing observed DNA sequences. All sequences in this column must be multiple aligned.
`germ`	name of the column containing germline DNA sequences. All entries in this column should be identical for any given clone, and they must be multiple aligned with the data in the `seq` column.
`v_call`	name of the column containing V-segment allele assignments. All entries in this column should be identical to the gene level.
`j_call`	name of the column containing J-segment allele assignments. All entries in this column should be identical to the gene level.
`junc_len`	name of the column containing the length of the junction as a numeric value. All entries in this column should be identical for any given clone.
`clone`	name of the column containing the identifier for the clone. All entries in this column should be identical.
`mask_char`	character to use for masking and padding.
`max_mask`	maximum number of characters to mask at the leading and trailing sequence ends. If `NULL` then the upper masking bound will be automatically determined from the maximum number of observed leading or trailing Ns amongst all sequences. If set to `0` (default) then masking will not be performed.
`pad_end`	if `TRUE` pad the end of each sequence with `mask_char` to make every sequence the same length.

`text_fields`	text annotation columns to retain and merge during duplicate removal.
`num_fields`	numeric annotation columns to retain and sum during duplicate removal.
`seq_fields`	sequence annotation columns to retain and collapse during duplicate removal. Note, this is distinct from the `seq` and `germ` arguments, which contain the primary sequence data for the clone and should not be repeated in this argument.
`add_count`	if `TRUE` add an additional annotation column called `collapse_count` during duplicate removal that indicates the number of sequences that were collapsed.
`verbose`	passed on to `collapseDuplicates`. If `TRUE`, report the numbers of input, discarded and output sequences; otherwise, process sequences silently.

Details

The input data.frame (`data`) must columns for each of the required column name arguments: `id`, `seq`, `germ`, `v_call`, `j_call`, `junc_len`, and `clone`. The default values are as follows:

- `id = "sequence_id"`: unique sequence identifier.
- `seq = "sequence_alignment"`: IMGT-gapped sample sequence.
- `germ = "germline_alignment"`: IMGT-gapped germline sequence.
- `v_call = "v_call"`: V segment allele call.
- `j_call = "j_call"`: J segment allele call.
- `junc_len = "junction_length"`: junction sequence length.
- `clone = "clone_id"`: clone identifier.

Additional annotation columns specified in the `text_fields`, `num_fields` or `seq_fields` arguments will be retained in the `data` slot of the return object, but are not required. If the input data.frame `data` already contains a column named `sequence`, which is not used as the `seq` argument, then that column will not be retained.

The default columns are IMGT-gapped sequence columns, but this is not a requirement. However, all sequences (both observed and germline) must be multiple aligned using some scheme for both proper duplicate removal and lineage reconstruction.

The value for the germline sequence, V-segment gene call, J-segment gene call, junction length, and clone identifier are determined from the first entry in the `germ`, `v_call`, `j_call`, `junc_len` and `clone` columns, respectively. For any given clone, each value in these columns should be identical.

Value

A ChangeoClone object containing the modified clone.

See Also

Executes in order maskSeqGaps, maskSeqEnds, padSeqEnds, and collapseDuplicates. Returns a ChangeoClone object which serves as input to buildPhylipLineage.

Examples

```
# Example data
db <- data.frame(sequence_id=LETTERS[1:4],
                 sequence_alignment=c("CCCCTGGG", "CCCCTGGN", "NAACTGGN", "NNNCTGNN"),
                 germline_alignment="CCCCAGGG",
                 v_call="Homsap IGKV1-39*01 F",
                 j_call="Homsap IGKJ5*01 F",
                 junction_length=2,
                 clone_id=1,
                 c_call=c("IGHM", "IGHG", "IGHG", "IGHA"),
                 duplicate_count=1:4,
                 stringsAsFactors=FALSE)


 # Without end masking
 makeChangeoClone(db, text_fields="c_call", num_fields="duplicate_count")

 # With end masking
 makeChangeoClone(db, max_mask=3, text_fields="c_call", num_fields="duplicate_count")
```

---

makeTempDir — *Create a temporary folder*

Description

`makeTempDir` creates a randomly named temporary folder in the system temp location.

Usage

```
makeTempDir(prefix)
```

Arguments

`prefix`	prefix name for the folder.

Value

The path to the temporary folder.

See Also

This is just a wrapper for tempfile and dir.create.

Examples

```
makeTempDir("Clone50")
```

maskPositionsByQuality

Mask sequence positions with low quality

Description

`maskPositionsByQuality` will replace positions that have a sequencing quality score lower that `min_quality` with an `"N"` character.

Usage

```
maskPositionsByQuality(
  data,
  min_quality = 70,
  sequence = "sequence_alignment",
  quality_num = "quality_alignment_num"
)
```

Arguments

`data`	`data.frame` containing sequence data.
`min_quality`	minimum quality score. Positions with sequencing quality less than `min_qual` will be masked.
`sequence`	column in `data` with sequence data to be masked.
`quality_num`	column in `data` with quality scores (a string of numeric values, comma separated) that can be used to mask sequence.

Value

Modified `data` data.frame with an additional field containing quality masked sequences. The name of this field is created concatenating the `sequence` name and `"_masked"`.

See Also

readFastqDb and getPositionQuality

Examples

```
db <- airr::read_rearrangement(system.file("extdata", "example_quality.tsv", package="alakazam"))
fastq_file <- system.file("extdata", "example_quality.fastq", package="alakazam")
db <- readFastqDb(db, fastq_file, quality_offset=-33)
maskPositionsByQuality(db, min_quality=90, quality_num="quality_alignment_num")
```

maskSeqEnds *Masks ragged leading and trailing edges of aligned DNA sequences*

Description

maskSeqEnds takes a vector of DNA sequences, as character strings, and replaces the leading and trailing characters with "N" characters to create a sequence vector with uniformly masked outer sequence segments.

Usage

```
maskSeqEnds(seq, mask_char = "N", max_mask = NULL, trim = FALSE)
```

Arguments

seq	character vector of DNA sequence strings.
mask_char	character to use for masking.
max_mask	the maximum number of characters to mask. If set to 0 then no masking will be performed. If set to NULL then the upper masking bound will be automatically determined from the maximum number of observed leading or trailing "N" characters amongst all strings in seq.
trim	if TRUE leading and trailing characters will be cut rather than masked with "N" characters.

Value

A modified seq vector with masked (or optionally trimmed) sequences.

See Also

See maskSeqGaps for masking internal gaps. See padSeqEnds for padding sequence of unequal length.

Examples

```
# Default behavior uniformly masks ragged ends
seq <- c("CCCCTGGG", "NAACTGGN", "NNNCTGNN")
maskSeqEnds(seq)

# Does nothing
maskSeqEnds(seq, max_mask=0)

# Cut ragged sequence ends
maskSeqEnds(seq, trim=TRUE)

# Set max_mask to limit extent of masking and trimming
maskSeqEnds(seq, max_mask=1)
maskSeqEnds(seq, max_mask=1, trim=TRUE)
```

```
# Mask dashes instead of Ns
seq <- c("CCCCTGGG", "-AACTGG-", "---CTG--")
maskSeqEnds(seq, mask_char="-")
```

---

maskSeqGaps *Masks gap characters in DNA sequences*

Description

maskSeqGaps substitutes gap characters, c("-","."), with "N" in a vector of DNA sequences.

Usage

```
maskSeqGaps(seq, mask_char = "N", outer_only = FALSE)
```

Arguments

seq	character vector of DNA sequence strings.
mask_char	character to use for masking.
outer_only	if TRUE replace only contiguous leading and trailing gaps; if FALSE replace all gap characters.

Value

A modified seq vector with "N" in place of c("-",".") characters.

See Also

See maskSeqEnds for masking ragged edges.

Examples

```
# Mask with Ns
maskSeqGaps(c("ATG-C", "CC..C"))
maskSeqGaps("--ATG-C-")
maskSeqGaps("--ATG-C-", outer_only=TRUE)

# Mask with dashes
maskSeqGaps(c("ATG-C", "CC..C"), mask_char="-")
```

MRCATest-class S4 class defining edge significance

Description

`MRCATest` defines the significance of enrichment for annotations appearing at the MRCA of the tree.

Usage

```
## S4 method for signature 'MRCATest'
print(x)

## S4 method for signature 'MRCATest,missing'
plot(x, y, ...)
```

Arguments

x	MRCATest object.
y	ignored.
...	arguments to pass to plotMRCATest.

Slots

`tests` data.frame describing the significance test results with columns:

- `annotation`: annotation value.
- `count`: observed count of MRCA positions with the given annotation.
- `expected`: expected mean count of MRCA occurance for the annotation.
- `pvalue`: one-sided p-value for the hypothesis that the observed annotation abundance is greater than expected.

`permutations` data.frame containing the raw permutation test data with columns:

- `annotation`: annotation value.
- `count`: count of MRCA positions with the given annotation.
- `iter`: numerical index define which permutation realization each observation corresponds to.

`nperm` number of permutation realizations.

`nonsquareDist` *Calculate pairwise distances between sequences*

Description

`nonsquareDist` calculates all pairwise distance between a set of sequences and a subset of it.

Usage

```
nonsquareDist(seq, indx, dist_mat = getDNAMatrix())
```

Arguments

`seq`	character vector containing a DNA sequences. The sequence vector needs to be named.
`indx`	numeric vector contating the indices (a subset of indices of `seq`).
`dist_mat`	Character distance matrix. Defaults to a Hamming distance matrix returned by getDNAMatrix. If gap characters, `c("-",".")`, are assigned a value of -1 in `dist_mat` then contiguous gaps of any run length, which are not present in both sequences, will be counted as a distance of 1. Meaning, indels of any length will increase the sequence distance by 1. Gap values other than -1 will return a distance that does not consider indels as a special case.

Value

A matrix of numerical distance between each entry in `seq` and sequences specified by `indx` indices.

Note that the input subsampled indices will be ordered ascendingly. Therefore, it is necassary to assign unique names to the input sequences, `seq`, to recover the input order later. Row and columns names will be added accordingly.

Amino acid distance matrix may be built with getAAMatrix. Uses seqDist for calculating distances between pairs. See pairwiseEqual for generating an equivalence matrix.

Examples

```
# Gaps will be treated as Ns with a gap=0 distance matrix
seq <- c(A="ATGGC", B="ATGGG", C="ATGGG", D="AT--C")
pairwiseDist(seq,
             dist_mat=getDNAMatrix(gap=0))

nonsquareDist(seq, indx=c(1,3),
             dist_mat=getDNAMatrix(gap=0))
```

padSeqEnds	*Pads ragged ends of aligned DNA sequences*

Description

padSeqEnds takes a vector of DNA sequences, as character strings, and appends the ends of each sequence with an appropriate number of "N" characters to create a sequence vector with uniform lengths.

Usage

```
padSeqEnds(seq, len = NULL, start = FALSE, pad_char = "N", mod3 = TRUE)
```

Arguments

seq	character vector of DNA sequence strings.
len	length to pad to. Only applies if longer than the maximum length of the data in seq.
start	if TRUE pad the beginning of each sequence instead of the end.
pad_char	character to use for padding.
mod3	if TRUE pad sequences to be of length multiple three.

Value

A modified seq vector with padded sequences.

See Also

See maskSeqEnds for creating uniform masking from existing masking.

Examples

```
# Default behavior uniformly pads ragged ends
seq <- c("CCCCTGGG", "ACCCTG", "CCCC")
padSeqEnds(seq)

# Pad to fixed length
padSeqEnds(seq, len=15)

# Add padding to the beginning of the sequences instead of the ends
padSeqEnds(seq, start=TRUE)
padSeqEnds(seq, len=15, start=TRUE)
```

`pairwiseDist` *Calculate pairwise distances between sequences*

Description

`pairwiseDist` calculates all pairwise distance between a set of sequences.

Usage

```
pairwiseDist(seq, dist_mat = getDNAMatrix())
```

Arguments

`seq`	character vector containing a DNA sequences.
`dist_mat`	Character distance matrix. Defaults to a Hamming distance matrix returned by getDNAMatrix. If gap characters, `c("-",".")`, are assigned a value of -1 in `dist_mat` then contiguous gaps of any run length, which are not present in both sequences, will be counted as a distance of 1. Meaning, indels of any length will increase the sequence distance by 1. Gap values other than -1 will return a distance that does not consider indels as a special case.

Value

A matrix of numerical distance between each entry in `seq`. If `seq` is a named vector, row and columns names will be added accordingly.

Amino acid distance matrix may be built with getAAMatrix. Uses seqDist for calculating distances between pairs. See pairwiseEqual for generating an equivalence matrix.

Examples

```
# Gaps will be treated as Ns with a gap=0 distance matrix
pairwiseDist(c(A="ATGGC", B="ATGGG", C="ATGGG", D="AT--C"),
             dist_mat=getDNAMatrix(gap=0))

# Gaps will be treated as universally non-matching characters with gap=1
pairwiseDist(c(A="ATGGC", B="ATGGG", C="ATGGG", D="AT--C"),
             dist_mat=getDNAMatrix(gap=1))

# Gaps of any length will be treated as single mismatches with a gap=-1 distance matrix
pairwiseDist(c(A="ATGGC", B="ATGGG", C="ATGGG", D="AT--C"),
             dist_mat=getDNAMatrix(gap=-1))
```

pairwiseEqual *Calculate pairwise equivalence between sequences*

Description

`pairwiseEqual` determined pairwise equivalence between a pairs in a set of sequences, excluding ambiguous positions (Ns and gaps).

Usage

```
pairwiseEqual(seq)
```

Arguments

`seq`	character vector containing a DNA sequences.

Value

A logical matrix of equivalence between each entry in `seq`. Values are `TRUE` when sequences are equivalent and `FALSE` when they are not.

See Also

Uses seqEqual for testing equivalence between pairs. See pairwiseDist for generating a sequence distance matrix.

Examples

```
# Gaps and Ns will match any character
seq <- c(A="ATGGC", B="ATGGG", C="ATGGG", D="AT--C", E="NTGGG")
d <- pairwiseEqual(seq)
rownames(d) <- colnames(d) <- seq
d
```

permuteLabels *Permute the node labels of a tree*

Description

`permuteLabels` permutes the node annotations of a lineage tree.

Usage

```
permuteLabels(graph, field, exclude = c("Germline", NA))
```

Arguments

`graph`	igraph object containing an annotated lineage tree.
`field`	string defining the annotation field to permute.
`exclude`	vector of strings defining `field` values to exclude from permutation.

Value

A modified igraph object with vertex annotations permuted.

See Also

testEdges.

Examples

```
# Define and plot example graph
library(igraph)
graph <- ExampleTrees[[23]]
plot(graph, layout=layout_as_tree, vertex.label=V(graph)$c_call,
     vertex.size=50, edge.arrow.mode=0, vertex.color="grey80")

# Permute annotations and plot new tree
g <- permuteLabels(graph, "c_call")
plot(g, layout=layout_as_tree, vertex.label=V(g)$c_call,
     vertex.size=50, edge.arrow.mode=0, vertex.color="grey80")
```

`phyloToGraph` *Convert a tree in ape* `phylo` *format to igraph* `graph` *format.*

Description

`phyloToGraph` converts a tree in `phylo` format to and `graph` format.

Usage

```
phyloToGraph(phylo, germline = "Germline")
```

Arguments

`phylo`	An ape `phylo` object.
`germline`	If specified, places specified tip sequence as the direct ancestor of the tree

Details

Convert from phylo to graph object. Uses the node.label vector to label internal nodes. Nodes may rotate but overall topology will remain constant.

Value

A graph object representing the input tree.

References

1. Hoehn KB, Lunter G, Pybus OG - A Phylogenetic Codon Substitution Model for Antibody Lineages. Genetics 2017 206(1):417-427 https://doi.org/10.1534/genetics.116.196303
2. Hoehn KB, Vander Heiden JA, Zhou JQ, Lunter G, Pybus OG, Kleinstein SHK - Repertoire-wide phylogenetic models of B cell molecular evolution reveal evolutionary signatures of aging and vaccination. bioRxiv 2019 https://doi.org/10.1101/558825

Examples

```
## Not run:
   library(igraph)
   library(ape)

   #convert to phylo
   phylo = graphToPhylo(graph)

   #plot tree using ape
   plot(phylo,show.node.label=TRUE)

   #store as newick tree
   write.tree(phylo,file="tree.newick")

   #read in tree from newick file
   phylo_r = read.tree("tree.newick")

   #convert to igraph
   graph_r = phyloToGraph(phylo_r,germline="Germline")

   #plot graph - same as before, possibly rotated
   plot(graph_r,layout=layout_as_tree)

## End(Not run)
```

---

plotAbundanceCurve *Plots a clonal abundance distribution*

Description

plotAbundanceCurve plots the results from estimating the complete clonal relative abundance distribution. The distribution is plotted as a log rank abundance distribution.

Usage

```
plotAbundanceCurve(
  data,
  colors = NULL,
  main_title = "Rank Abundance",
  legend_title = NULL,
  xlim = NULL,
  ylim = NULL,
  annotate = c("none", "depth"),
  silent = FALSE,
  ...
)
```

Arguments

`data`	AbundanceCurve object returned by estimateAbundance.
`colors`	named character vector whose names are values in the `group` column of `data` and whose values are colors to assign to those group names.
`main_title`	string specifying the plot title.
`legend_title`	string specifying the legend title.
`xlim`	numeric vector of two values specifying the `c(lower,upper)` x-axis limits.
`ylim`	numeric vector of two values specifying the `c(lower,upper)` y-axis limits.
`annotate`	string defining whether to added values to the group labels of the legend. When `"none"` (default) is specified no annotations are added. Specifying (`"depth"`) adds sequence counts to the labels.
`silent`	if `TRUE` do not draw the plot and just return the ggplot2 object; if `FALSE` draw the plot.
`...`	additional arguments to pass to ggplot2::theme.

Value

A `ggplot` object defining the plot.

See Also

See AbundanceCurve for the input object and estimateAbundance for generating the input abundance distribution. Plotting is performed with ggplot.

Examples

```
# Estimate abundance by sample and plot
abund <- estimateAbundance(ExampleDb, group="sample_id", nboot=100)
plotAbundanceCurve(abund, legend_title="Sample")
```

plotDiversityCurve *Plot the results of alphaDiversity*

Description

plotDiversityCurve plots a DiversityCurve object.

Usage

```
plotDiversityCurve(
  data,
  colors = NULL,
  main_title = "Diversity",
  legend_title = "Group",
  log_x = FALSE,
  log_y = FALSE,
  xlim = NULL,
  ylim = NULL,
  annotate = c("none", "depth"),
  score = c("diversity", "evenness"),
  silent = FALSE,
  ...
)
```

Arguments

data	DiversityCurve object returned by alphaDiversity.
colors	named character vector whose names are values in the group column of the data slot of data, and whose values are colors to assign to those group names.
main_title	string specifying the plot title.
legend_title	string specifying the legend title.
log_x	if TRUE then plot q on a log scale; if FALSE plot on a linear scale.
log_y	if TRUE then plot the diversity/evenness scores on a log scale; if FALSE plot on a linear scale.
xlim	numeric vector of two values specifying the c(lower,upper) x-axis limits.
ylim	numeric vector of two values specifying the c(lower,upper) y-axis limits.
annotate	string defining whether to added values to the group labels of the legend. When "none" (default) is specified no annotations are added. Specifying ("depth") adds sequence counts to the labels.
score	one of "diversity" or "evenness" specifying which score to plot on the y-asis.
silent	if TRUE do not draw the plot and just return the ggplot2 object; if FALSE draw the plot.
...	additional arguments to pass to ggplot2::theme.

Value

A ggplot object defining the plot.

See Also

See alphaDiversity and alphaDiversity for generating DiversityCurve objects for input. Plotting is performed with ggplot.

Examples

```
# Calculate diversity
div <- alphaDiversity(ExampleDb, group="sample_id", nboot=100)

# Plot diversity
plotDiversityCurve(div, legend_title="Sample")

#' # Plot diversity
plotDiversityCurve(div, legend_title="Sample", score="evenness")
```

---

plotDiversityTest *Plot the results of diversity testing*

Description

plotDiversityTest plots summary data for a DiversityCurve object with mean and a line range indicating plus/minus one standard deviation.

Usage

```
plotDiversityTest(
  data,
  q,
  colors = NULL,
  main_title = "Diversity",
  legend_title = "Group",
  log_d = FALSE,
  annotate = c("none", "depth"),
  silent = FALSE,
  ...
)
```

Arguments

data	DiversityCurve object returned by alphaDiversity.
q	diversity order to plot the test for.

colors	named character vector whose names are values in the group column of the data slot of data, and whose values are colors to assign to those group names.
main_title	string specifying the plot title.
legend_title	string specifying the legend title.
log_d	if TRUE then plot the diversity scores D on a log scale; if FALSE plot on a linear scale.
annotate	string defining whether to added values to the group labels of the legend. When "none" (default) is specified no annotations are added. Specifying ("depth") adds sequence counts to the labels.
silent	if TRUE do not draw the plot and just return the ggplot2 object; if FALSE draw the plot.
...	additional arguments to pass to ggplot2::theme.

Value

A ggplot object defining the plot.

See Also

See alphaDiversity for generating input. Plotting is performed with ggplot.

Examples

```
# Calculate diversity
div <- alphaDiversity(ExampleDb, group="sample_id", min_q=0, max_q=2, step_q=1, nboot=100)

# Plot results at q=0 (equivalent to species richness)
plotDiversityTest(div, 0, legend_title="Sample")

# Plot results at q=2 (equivalent to Simpson's index)
plotDiversityTest(div, q=2, legend_title="Sample")
```

---

plotEdgeTest *Plot the results of an edge permutation test*

Description

plotEdgeTest plots the results of an edge permutation test performed with testEdges as either a histogram or cumulative distribution function.

Usage

```
plotEdgeTest(
  data,
  color = "black",
  main_title = "Edge Test",
  style = c("histogram", "cdf"),
  silent = FALSE,
  ...
)
```

Arguments

`data`	EdgeTest object returned by testEdges.
`color`	color of the histogram or lines.
`main_title`	string specifying the plot title.
`style`	type of plot to draw. One of: • `"histogram"`: histogram of the edge count distribution with a red dotted line denoting the observed value. • `"cdf"`: cumulative distribution function of edge counts with a red dotted line denoting the observed value and a blue dotted line indicating the p-value.
`silent`	if `TRUE` do not draw the plot and just return the ggplot2 object; if `FALSE` draw the plot.
`...`	additional arguments to pass to ggplot2::theme.

Value

A `ggplot` object defining the plot.

See Also

See testEdges for performing the test.

Examples

```
# Define example tree set
graphs <- ExampleTrees[1-10]

# Perform edge test on isotypes
x <- testEdges(graphs, "c_call", nperm=10)

# Plot
plotEdgeTest(x, color="steelblue", style="hist")
plotEdgeTest(x, style="cdf")
```

plotMRCATest *Plot the results of a founder permutation test*

Description

plotMRCATest plots the results of a founder permutation test performed with testMRCA.

Usage

```
plotMRCATest(
  data,
  color = "black",
  main_title = "MRCA Test",
  style = c("histogram", "cdf"),
  silent = FALSE,
  ...
)
```

Arguments

data	MRCATest object returned by testMRCA.
color	color of the histogram or lines.
main_title	string specifying the plot title.
style	type of plot to draw. One of: • "histogram": histogram of the annotation count distribution with a red dotted line denoting the observed value. • "cdf": cumulative distribution function of annotation counts with a red dotted line denoting the observed value and a blue dotted line indicating the p-value.
silent	if TRUE do not draw the plot and just return the ggplot2 object; if FALSE draw the plot.
...	additional arguments to pass to ggplot2::theme.

Value

A ggplot object defining the plot.

See Also

See testEdges for performing the test.

Examples

```
# Define example tree set
graphs <- ExampleTrees[1-10]

# Perform MRCA test on isotypes
x <- testMRCA(graphs, "c_call", nperm=10)

# Plot
plotMRCATest(x, color="steelblue", style="hist")
plotMRCATest(x, style="cdf")
```

---

plotSubtrees *Plots subtree statistics for multiple trees*

Description

plotSubtree plots distributions of normalized subtree statistics for a set of lineage trees, broken down by annotation value.

Usage

```
plotSubtrees(
  graphs,
  field,
  stat,
  root = "Germline",
  exclude = c("Germline", NA),
  colors = NULL,
  main_title = "Subtrees",
  legend_title = "Annotation",
  style = c("box", "violin"),
  silent = FALSE,
  ...
)
```

Arguments

graphs list of igraph objects containing annotated lineage trees.

field string defining the annotation field.

stat string defining the subtree statistic to plot. One of:

- outdegree: distribution of normalized node outdegrees.
- size: distribution of normalized subtree sizes.
- depth: distribution of subtree depths.

- `pathlength`: distribution of maximum pathlength beneath nodes.

`root` name of the root (germline) node.

`exclude` vector of strings defining `field` values to exclude from plotting.

`colors` named vector of colors for values in `field`, with names defining annotation names `field` column and values being colors. Also controls the order in which values appear on the plot. If `NULL` alphabetical ordering and a default color palette will be used.

`main_title` string specifying the plot title.

`legend_title` string specifying the legend title.

`style` string specifying the style of plot to draw. One of:

- `"histogram"`: histogram of the annotation count distribution with a red dotted line denoting the observed value.
- `"cdf"`: cumulative distribution function of annotation counts with a red dotted line denoting the observed value and a blue dotted line indicating the p-value.

`silent` if `TRUE` do not draw the plot and just return the ggplot2 object; if `FALSE` draw the plot.

`...` additional arguments to pass to ggplot2::theme.

Value

A `ggplot` object defining the plot.

See Also

Subtree statistics are calculated with summarizeSubtrees.

Examples

```
# Define example tree set
graphs <- ExampleTrees[1-10]

# Violin plots of node outdegree by sample
plotSubtrees(graphs, "sample_id", "out", style="v")

# Violin plots of subtree size by sample
plotSubtrees(graphs, "sample_id", "size", style="v")

# Boxplot of node depth by isotype
plotSubtrees(graphs, "c_call", "depth", style="b")
```

polar — *Calculates the average polarity of amino acid sequences*

Description

`polar` calculates the average polarity score of amino acid sequences. Non-informative positions are excluded, where non-informative is defined as any character in `c("X","-",".","*")`.

Usage

```
polar(seq, polarity = NULL)
```

Arguments

`seq`	vector of strings containing amino acid sequences.
`polarity`	named numerical vector defining polarity scores for each amino acid, where names are single-letter amino acid character codes. If `NULL`, then the Grantham, 1974 scale is used.

Value

A vector of bulkiness scores for the sequence(s).

References

1. Grantham R. Amino acid difference formula to help explain protein evolution. Science 185, 862-864 (1974).

See Also

For additional size related indices see `aaindex`.

Examples

```
# Default scale
seq <- c("CARDRSTPWRRGIASTTVRTSW", "XXTQMYVRT")
polar(seq)

# Use the Zimmerman et al, 1968 polarity scale from the seqinr package
library(seqinr)
data(aaindex)
x <- aaindex[["ZIMJ680103"]]$I
# Rename the score vector to use single-letter codes
names(x) <- translateStrings(names(x), ABBREV_AA)
# Calculate polarity
polar(seq, polarity=x)
```

progressBar *Standard progress bar*

Description

`progressBar` defines a common progress bar format.

Usage

```
progressBar(n)
```

Arguments

n	maximum number of ticks

Value

A progress_bar object.

rarefyDiversity *Generate a clonal diversity index curve*

Description

`rarefyDiversity` divides a set of clones by a group annotation, resamples the sequences from each group, and calculates diversity scores (D) over an interval of diversity orders (q).

Usage

```
rarefyDiversity(
  data,
  group,
  clone = "CLONE",
  copy = NULL,
  min_q = 0,
  max_q = 4,
  step_q = 0.05,
  min_n = 30,
  max_n = NULL,
  ci = 0.95,
  nboot = 2000,
  uniform = TRUE,
  progress = FALSE
)
```

Arguments

data	data.frame with Change-O style columns containing clonal assignments.
group	name of the data column containing group identifiers.
clone	name of the data column containing clone identifiers.
copy	name of the data column containing copy numbers for each sequence. If copy=NULL (the default), then clone abundance is determined by the number of sequences. If a copy column is specified, then clone abundances is determined by the sum of copy numbers within each clonal group.
min_q	minimum value of q.
max_q	maximum value of q.
step_q	value by which to increment q.
min_n	minimum number of observations to sample. A group with less observations than the minimum is excluded.
max_n	maximum number of observations to sample. If NULL then no maximum is set.
ci	confidence interval to calculate; the value must be between 0 and 1.
nboot	number of bootstrap realizations to generate.
uniform	if TRUE then uniformly resample each group to the same number of observations. If FALSE then allow each group to be resampled to its original size or, if specified, max_size.
progress	if TRUE show a progress bar.

Details

Clonal diversity is calculated using the generalized diversity index (Hill numbers) proposed by Hill (Hill, 1973). See calcDiversity for further details.

Diversity is calculated on the estimated complete clonal abundance distribution. This distribution is inferred by using the Chao1 estimator to estimate the number of seen clones, and applying the relative abundance correction and unseen clone frequency described in Chao et al, 2015.

To generate a smooth curve, D is calculated for each value of q from min_q to max_q incremented by step_q. When uniform=TRUE variability in total sequence counts across unique values in the group column is corrected by repeated resampling from the estimated complete clonal distribution to a common number of sequences.

The diversity index (D) for each group is the mean value of over all resampling realizations. Confidence intervals are derived using the standard deviation of the resampling realizations, as described in Chao et al, 2015.

Value

A DiversityCurve object summarizing the diversity scores.

References

1. Hill M. Diversity and evenness: a unifying notation and its consequences. Ecology. 1973 54(2):427-32.
2. Chao A. Nonparametric Estimation of the Number of Classes in a Population. Scand J Stat. 1984 11, 265270.
3. Chao A, et al. Rarefaction and extrapolation with Hill numbers: A framework for sampling and estimation in species diversity studies. Ecol Monogr. 2014 84:45-67.
4. Chao A, et al. Unveiling the species-rank abundance distribution by generalizing the Good-Turing sample coverage theory. Ecology. 2015 96, 11891201.

See Also

alphaDiversity

Examples

```
## Not run:
# Group by sample identifier
div <- rarefyDiversity(ExampleDb, "sample_id", step_q=1, max_q=10, nboot=100)
plotDiversityCurve(div, legend_title="Sample")

# Grouping by isotype rather than sample identifier
div <- rarefyDiversity(ExampleDb, "c_call", min_n=40, step_q=1, max_q=10,
                       nboot=100)
plotDiversityCurve(div, legend_title="Isotype")

## End(Not run)
```

---

readChangeoDb *Read a Change-O tab-delimited database file*

Description

`readChangeoDb` reads a tab-delimited database file created by a Change-O tool into a data.frame.

Usage

```
readChangeoDb(file, select = NULL, drop = NULL, seq_upper = TRUE)
```

Arguments

`file`	tab-delimited database file output by a Change-O tool.
`select`	columns to select from database file.
`drop`	columns to drop from database file.
`seq_upper`	if `TRUE` convert sequence columns to upper case; if `FALSE` do not alter sequence columns. See Value for a list of which columns are effected.

Value

A data.frame of the database file. Columns will be imported as is, except for the following columns which will be explicitly converted into character values:

- SEQUENCE_ID
- CLONE
- SAMPLE

And the following sequence columns which will converted to upper case if `seq_upper=TRUE` (default).

- SEQUENCE_INPUT
- SEQUENCE_VDJ
- SEQUENCE_IMGT
- JUNCTION
- GERMLINE_IMGT
- GERMLINE_IMGT_D_MASK

See Also

Wraps read_delim. See writeChangeoDb for writing to Change-O files.

Examples

```
## Not run:
    # Read all columns in and convert sequence fields to upper case
    db <- readChangeoDb("changeo.tsv")

    # Subset columns and convert sequence fields to upper case
    db <- readChangeoDb("changeo.tsv", select=c("SEQUENCE_ID", "SEQUENCE_IMGT"))

    # Drop columns and do not alter sequence field case
    db <- readChangeoDb("changeo.tsv", drop=c("D_CALL", "DUPCOUNT"),
                        seq_upper=FALSE)

## End(Not run)
```

`readFastqDb` *Load sequencing quality scores from a FASTQ file*

Description

`readFastqDb` adds the sequencing quality scores to a data.frame from a FASTQ file. Matching is done by ‘sequence_id‘.

Usage

```
readFastqDb(
  data,
  fastq_file,
  quality_offset = -33,
  header = c("presto", "asis"),
  sequence_id = "sequence_id",
  sequence = "sequence",
  sequence_alignment = "sequence_alignment",
  v_cigar = "v_cigar",
  d_cigar = "d_cigar",
  j_cigar = "j_cigar",
  np1_length = "np1_length",
  np2_length = "np2_length",
  v_sequence_end = "v_sequence_end",
  d_sequence_end = "d_sequence_end",
  style = c("num", "ascii", "both"),
  quality_sequence = FALSE
)
```

Arguments

- `data` — `data.frame` containing sequence data.
- `fastq_file` — path to the fastq file
- `quality_offset` — offset value to be used by ape::read.fastq. It is the value to be added to the quality scores (the default -33 applies to the Sanger format and should work for most recent FASTQ files).
- `header` — FASTQ file header format; one of `"presto"` or `"asis"`. Use `"presto"` to specify that the fastq file headers are using the pRESTO format and can be parsed to extract the `sequence_id`. Use `"asis"` to skip any processing and use the sequence names as they are.
- `sequence_id` — column in `data` that contains sequence identifiers to be matched to sequence identifiers in `fastq_file`.
- `sequence` — column in `data` that contains sequence data.
- `sequence_alignment` — column in `data` that contains IMGT aligned sequence data.
- `v_cigar` — column in `data` that contains CIGAR strings for the V gene alignments.
- `d_cigar` — column in `data` that contains CIGAR strings for the D gene alignments.
- `j_cigar` — column in `data` that contains CIGAR strings for the J gene alignments.
- `np1_length` — column in `data` that contains the number of nucleotides between the V gene and first D gene alignments or between the V gene and J gene alignments.
- `np2_length` — column in `data` that contains the number of nucleotides between either the first D gene and J gene alignments or the first D gene and second D gene alignments.
- `v_sequence_end` — column in `data` that contains the end position of the V gene in `sequence`.

d_sequence_end column in data that contains the end position of the D gene in sequence.

style how the sequencing quality should be returned; one of "num", "phred", or "both". Specify "num" to store the quality scores as strings of comma separated numeric values. Use "phred" to have the function return the scores as Phred (ASCII) scores. Use "both" to retrieve both.

quality_sequence
specify TRUE to keep the quality scores for sequence. If false, only the quality score for sequence_alignment will be added to data.

Value

Modified data with additional fields:

1. quality_alignment: A character vector with ASCII Phred scores for sequence_alignment.
2. quality_alignment_num: A character vector, with comma separated numerical quality values for each position in sequence_alignment.
3. quality: A character vector with ASCII Phred scores for sequence.
4. quality_num: A character vector, with comma separated numerical quality values for each position in sequence.

See Also

maskPositionsByQuality and getPositionQuality

Examples

```
db <- airr::read_rearrangement(system.file("extdata", "example_quality.tsv", package="alakazam"))
fastq_file <- system.file("extdata", "example_quality.fastq", package="alakazam")
db <- readFastqDb(db, fastq_file, quality_offset=-33)
```

---

readIgphyml *Read in output from IgPhyML*

Description

readIgphyml reads output from the IgPhyML phylogenetics inference package for B cell repertoires

Usage

```
readIgphyml(
  file,
  id = NULL,
  format = c("graph", "phylo"),
  collapse = FALSE,
  branches = c("mutations", "distance")
)
```

Arguments

file	IgPhyML output file (.tab).
id	ID to assign to output object.
format	if "graph" return trees as igraph graph objects. if "phylo" return trees as ape phylo objects.
collapse	if TRUE transform branch lengths to units of substitutions, rather than substitutions per site, and collapse internal nodes separated by branches < 0.1 substitutions. Will also remove all internal node labels, as it makes them inconsistent.
branches	if "distance" branch lengths are in expected mutations per site. If "mutations" branches are in expected mutations.

Details

readIgphyml reads output from the IgPhyML repertoire phylogenetics inference package. The resulting object is divded between parameter estimates (usually under the HLP19 model), which provide information about mutation and selection pressure operating on the sequences.

Trees returned from this function are either igraph objects or phylo objects, and each may be visualized accordingly. Futher, branch lengths in tree may represent either the expected number of substitutions per site (codon, if estimated under HLP or GY94 models), or the total number of expected substitutions per site. If the latter, internal nodes - but not tips - separated by branch lengths less than 0.1 are collapsed to simplify viewing.

Value

A list containing IgPhyML model parameters and estimated lineage trees.

Object attributes:

- param: Data.frame of parameter estimates for each clonal lineage. Columns include: CLONE, which is the clone id; NSEQ, the total number of sequences in the lineage; NSITE, the number of codon sites; TREE_LENGTH, the sum of all branch lengths in the estimated lineage tree; and LHOOD, the log likelihood of the clone's sequences given the tree and parameters. Subsequent columns are parameter estimates from IgPhyML, which will depend on the model used. Parameter columns ending with _MLE are maximum likelihood estimates; those ending with _LCI are the lower 95 with _UCI are the upper 95 estimate. The first line of param is for clone REPERTOIRE, which is a summary of all lineages within the repertoire. For this row, NSEQ is the total number of sequences, NSITE is the average number of sites, and TREE_LENGTH is the mean tree length. For most applications, parameter values will be the same for all lineages within the repertoire, so access them simply by: <object>$param$OMEGA_CDR_MLE[1] to, for instance, get the estimate of dN/dS on the CDRs at the repertoire level.
- trees: List of tree objects estimated by IgPhyML. If format="graph" these are igraph graph objects. If format="phylo", these are ape phylo objects.
- command: Command used to run IgPhyML.

References

1. Hoehn KB, Lunter G, Pybus OG - A Phylogenetic Codon Substitution Model for Antibody Lineages. Genetics 2017 206(1):417-427 https://doi.org/10.1534/genetics.116.196303

2. Hoehn KB, Vander Heiden JA, Zhou JQ, Lunter G, Pybus OG, Kleinstein SHK - Repertoire-wide phylogenetic models of B cell molecular evolution reveal evolutionary signatures of aging and vaccination. bioRxiv 2019 https://doi.org/10.1101/558825

Examples

```
## Not run:
   # Read in and plot a tree from an igphyml run
   library(igraph)
   s1 <- readIgphyml("IB+7d_lineages_gy.tsv_igphyml_stats_hlp.tab", id="+7d")
   print(s1$param$OMEGA_CDR_MLE[1])
   plot(s1$trees[[1]], layout=layout_as_tree, edge.label=E(s1$trees[[1]])$weight)

## End(Not run)
```

---

seqDist *Calculate distance between two sequences*

Description

`seqDist` calculates the distance between two DNA sequences.

Usage

```
seqDist(seq1, seq2, dist_mat = getDNAMatrix())
```

Arguments

seq1	character string containing a DNA sequence.
seq2	character string containing a DNA sequence.
dist_mat	Character distance matrix. Defaults to a Hamming distance matrix returned by getDNAMatrix. If gap characters, `c("-",".")`, are assigned a value of -1 in `dist_mat` then contiguous gaps of any run length, which are not present in both sequences, will be counted as a distance of 1. Meaning, indels of any length will increase the sequence distance by 1. Gap values other than -1 will return a distance that does not consider indels as a special case.

Value

Numerical distance between `seq1` and `seq2`.

See Also

Nucleotide distance matrix may be built with getDNAMatrix. Amino acid distance matrix may be built with getAAMatrix. Used by pairwiseDist for generating distance matrices. See seqEqual for testing sequence equivalence.

Examples

```
# Ungapped examples
seqDist("ATGGC", "ATGGG")
seqDist("ATGGC", "ATG??")

# Gaps will be treated as Ns with a gap=0 distance matrix
seqDist("ATGGC", "AT--C", dist_mat=getDNAMatrix(gap=0))

# Gaps will be treated as universally non-matching characters with gap=1
seqDist("ATGGC", "AT--C", dist_mat=getDNAMatrix(gap=1))

# Gaps of any length will be treated as single mismatches with a gap=-1 distance matrix
seqDist("ATGGC", "AT--C", dist_mat=getDNAMatrix(gap=-1))

# Gaps of equivalent run lengths are not counted as gaps
seqDist("ATG-C", "ATG-C", dist_mat=getDNAMatrix(gap=-1))

# Overlapping runs of gap characters are counted as a single gap
seqDist("ATG-C", "AT--C", dist_mat=getDNAMatrix(gap=-1))
seqDist("A-GGC", "AT--C", dist_mat=getDNAMatrix(gap=-1))
seqDist("AT--C", "AT--C", dist_mat=getDNAMatrix(gap=-1))

# Discontiguous runs of gap characters each count as separate gaps
seqDist("-TGGC", "AT--C", dist_mat=getDNAMatrix(gap=-1))
```

---

seqEqual *Test DNA sequences for equality.*

Description

`seqEqual` checks if two DNA sequences are identical.

Usage

```
seqEqual(seq1, seq2, ignore = as.character(c("N", "-", ".", "?")))
```

Arguments

seq1	character string containing a DNA sequence.
seq2	character string containing a DNA sequence.
ignore	vector of characters to ignore when testing for equality. Default is to ignore c("N",".","-","?")

Value

Returns `TRUE` if sequences are equal and `FALSE` if they are not. Sequences of unequal length will always return `FALSE` regardless of their character values.

See Also

Used by pairwiseEqual within collapseDuplicates. See seqDist for calculation Hamming distances between sequences.

Examples

```
# Ignore gaps
seqEqual("ATG-C", "AT--C")
seqEqual("ATGGC", "ATGGN")
seqEqual("AT--T", "ATGGC")

# Ignore only Ns
seqEqual("ATG-C", "AT--C", ignore="N")
seqEqual("ATGGC", "ATGGN", ignore="N")
seqEqual("AT--T", "ATGGC", ignore="N")
```

SingleDb *Single sequence AIRR database*

Description

A database with just one sequence from `ExampleDb` and additional AIRR Rearrangement fields containing alignment information. The sequence was reanalyzed with a recent versions of alignment software (IgBLAST 1.16.0) and reference germlines (IMGT 2020-08-12).

Usage

```
SingleDb
```

Format

An object of class `spec_tbl_df` (inherits from `tbl_df`, `tbl`, `data.frame`) with 1 rows and 32 columns.

See Also

ExampleDb

sortGenes — *Sort V(D)J genes*

Description

sortGenes sorts a vector of V(D)J gene names by either lexicographic ordering or locus position.

Usage

```
sortGenes(genes, method = c("name", "position"))
```

Arguments

genes	vector of strings respresenting V(D)J gene names.
method	string defining the method to use for sorting genes. One of: • "name": sort in lexicographic order. Order is by family first, then gene, and then allele. • "position": sort by position in the locus, as determined by the final two numbers in the gene name. Non-localized genes are assigned to the highest positions.

Value

A sorted character vector of gene names.

See Also

See getAllele, getGene and getFamily for parsing gene names.

Examples

```
# Create a list of allele names
genes <- c("IGHV1-69D*01","IGHV1-69*01","IGHV4-38-2*01","IGHV1-69-2*01",
           "IGHV2-5*01","IGHV1-NL1*01", "IGHV1-2*01,IGHV1-2*05",
           "IGHV1-2", "IGHV1-2*02", "IGHV1-69*02")

# Sort genes by name
sortGenes(genes)

# Sort genes by position in the locus
sortGenes(genes, method="pos")
```

`stoufferMeta` *Weighted meta-analysis of p-values via Stouffer's method*

Description

`stoufferMeta` combines multiple weighted p-values into a meta-analysis p-value using Stouffer's Z-score method.

Usage

```
stoufferMeta(p, w = NULL)
```

Arguments

`p`	numeric vector of p-values.
`w`	numeric vector of weights.

Value

A named numeric vector with the combined Z-score and p-value in the form `c(Z,pvalue)`.

Examples

```
# Define p-value and weight vectors
p <- c(0.1, 0.05, 0.3)
w <- c(5, 10, 1)

# Unweighted
stoufferMeta(p)

# Weighted
stoufferMeta(p, w)
```

`summarizeSubtrees` *Generate subtree summary statistics for a tree*

Description

`summarizeSubtrees` calculates summary statistics for each node of a tree. Includes both node properties and subtree properties.

Usage

```
summarizeSubtrees(graph, fields = NULL, root = "Germline")
```

Arguments

`graph`	igraph object containing an annotated lineage tree.
`fields`	annotation fields to add to the output.
`root`	name of the root (germline) node.

Value

A data.frame with columns:

- `name`: node name.
- `parent`: name of the parent node.
- `outdegree`: number of edges leading from the node.
- `size`: total number of nodes within the subtree rooted at the node.
- `depth`: the depth of the subtree that is rooted at the node.
- `pathlength`: the maximum pathlength beneath the node.
- `outdegree_norm`: `outdegree` normalized by the total number of edges.
- `size_norm`: `size` normalized by the largest subtree size (the germline).
- `depth_norm`: `depth` normalized by the largest subtree depth (the germline).
- `pathlength_norm`: `pathlength` normalized by the largest subtree pathlength (the germline).

An additional column corresponding to the value of `field` is added when specified.

See Also

See buildPhylipLineage for generating input trees. See getPathLengths for calculating path length to nodes.

Examples

```
# Summarize a tree
graph <- ExampleTrees[[23]]
summarizeSubtrees(graph, fields="c_call", root="Germline")
```

---

tableEdges *Tabulate the number of edges between annotations within a lineage tree*

Description

`tableEdges` creates a table of the total number of connections (edges) for each unique pair of annotations within a tree over all nodes.

Usage

```
tableEdges(graph, field, indirect = FALSE, exclude = NULL)
```

Arguments

- `graph` igraph object containing an annotated lineage tree.
- `field` string defining the annotation field to count.
- `indirect` if `FALSE` count direct connections (edges) only. If `TRUE` walk through any nodes with annotations specified in the `argument` to count indirect connections. Specifying `indirect=TRUE` with `exclude=NULL` will have no effect.
- `exclude` vector of strings defining `field` values to exclude from counts. Edges that either start or end with the specified annotations will not be counted. If `NULL` count all edges.

Value

A data.frame defining total annotation connections in the tree with columns:

- `parent`: parent annotation
- `child`: child annotation
- `count`: count of edges for the parent-child relationship

See Also

See testEdges for performed a permutation test on edge relationships.

Examples

```
# Define example graph
graph <- ExampleTrees[[23]]

# Count direct edges between isotypes including inferred nodes
tableEdges(graph, "c_call")

# Count direct edges excluding edges to and from germline and inferred nodes
tableEdges(graph, "c_call", exclude=c("Germline", NA))

# Count indirect edges walking through germline and inferred nodes
tableEdges(graph, "c_call", indirect=TRUE, exclude=c("Germline", NA))
```

`testDiversity` *Pairwise test of the diversity index*

Description

`testDiversity` performs pairwise significance tests of the diversity index (D) at a given diversity order (q) for a set of annotation groups via rarefaction and bootstrapping.

Usage

```
testDiversity(
  data,
  q,
  group,
  clone = "CLONE",
  copy = NULL,
  min_n = 30,
  max_n = NULL,
  nboot = 2000,
  progress = FALSE,
  ci = 0.95
)
```

Arguments

`data`	data.frame with Change-O style columns containing clonal assignments.
`q`	diversity order to test.
`group`	name of the `data` column containing group identifiers.
`clone`	name of the `data` column containing clone identifiers.
`copy`	name of the `data` column containing copy numbers for each sequence. If `copy=NULL` (the default), then clone abundance is determined by the number of sequences. If a `copy` column is specified, then clone abundances is determined by the sum of copy numbers within each clonal group.
`min_n`	minimum number of observations to sample. A group with less observations than the minimum is excluded.
`max_n`	maximum number of observations to sample. If `NULL` the maximum if automatically determined from the size of the largest group.
`nboot`	number of bootstrap realizations to perform.
`progress`	if `TRUE` show a progress bar.
`ci`	confidence interval to calculate; the value must be between 0 and 1.

Details

Clonal diversity is calculated using the generalized diversity index proposed by Hill (Hill, 1973). See calcDiversity for further details.

Diversity is calculated on the estimated complete clonal abundance distribution. This distribution is inferred by using the Chao1 estimator to estimate the number of seen clones, and applying the relative abundance correction and unseen clone frequency described in Chao et al, 2014.

Variability in total sequence counts across unique values in the `group` column is corrected by repeated resampling from the estimated complete clonal distribution to a common number of sequences. The diversity index estimate (D) for each group is the mean value of over all bootstrap realizations.

Significance of the difference in diversity index (D) between groups is tested by constructing a bootstrap delta distribution for each pair of unique values in the `group` column. The bootstrap delta distribution is built by subtracting the diversity index Da in $group - a$ from the corresponding value Db in $group - b$, for all bootstrap realizations, yeilding a distribution of `nboot` total deltas; where $group - a$ is the group with the greater mean D. The p-value for hypothesis $Da! = Db$ is the value of $P(0)$ from the empirical cumulative distribution function of the bootstrap delta distribution, multiplied by 2 for the two-tailed correction.

Value

A DiversityCurve object containing slot test with p-values and summary statistics.

Note

This method may inflate statistical significance when clone sizes are uniformly small, such as when most clones sizes are 1, sample size is small, and `max_n` is near the total count of the smallest data group. Use caution when interpreting the results in such cases. We are currently investigating this potential problem.

References

1. Hill M. Diversity and evenness: a unifying notation and its consequences. Ecology. 1973 54(2):427-32.
2. Chao A. Nonparametric Estimation of the Number of Classes in a Population. Scand J Stat. 1984 11, 265270.
3. Wu Y-CB, et al. Influence of seasonal exposure to grass pollen on local and peripheral blood IgE repertoires in patients with allergic rhinitis. J Allergy Clin Immunol. 2014 134(3):604-12.
4. Chao A, et al. Rarefaction and extrapolation with Hill numbers: A framework for sampling and estimation in species diversity studies. Ecol Monogr. 2014 84:45-67.
5. Chao A, et al. Unveiling the species-rank abundance distribution by generalizing the Good-Turing sample coverage theory. Ecology. 2015 96, 11891201.

See Also

alphaDiversity

Examples

```
## Not run:
# Groups under the size threshold are excluded and a warning message is issued.
testDiversity(ExampleDb, "sample_id", q=0, min_n=30, nboot=100)

## End(Not run)
```

testEdges *Tests for parent-child annotation enchrichment in lineage trees*

Description

testEdges performs a permutation test on a set of lineage trees to determine the significance of an annotation's association with parent-child relationships.

Usage

```
testEdges(
  graphs,
  field,
  indirect = FALSE,
  exclude = c("Germline", NA),
  nperm = 200,
  progress = FALSE
)
```

Arguments

graphs	list of igraph objects with vertex annotations.
field	string defining the annotation field to permute.
indirect	if FALSE count direct connections (edges) only. If TRUE walk through any nodes with annotations specified in the argument to count indirect connections. Specifying indirect=TRUE with exclude=NULL will have no effect.
exclude	vector of strings defining field values to exclude from permutation.
nperm	number of permutations to perform.
progress	if TRUE show a progress bar.

Value

An EdgeTest object containing the test results and permutation realizations.

See Also

Uses tableEdges and permuteLabels. See plotEdgeTest for plotting the permutation distributions.

Examples

```
# Define example tree set
graphs <- ExampleTrees[1-10]

# Perform edge test on isotypes
x <- testEdges(graphs, "c_call", nperm=10)
print(x)
```

testMRCA *Tests for MRCA annotation enrichment in lineage trees*

Description

testMRCA performs a permutation test on a set of lineage trees to determine the significance of an annotation's association with the MRCA position of the lineage trees.

Usage

```
testMRCA(
  graphs,
  field,
  root = "Germline",
  exclude = c("Germline", NA),
  nperm = 200,
  progress = FALSE
)
```

Arguments

graphs	list of igraph object containing annotated lineage trees.
field	string defining the annotation field to test.
root	name of the root (germline) node.
exclude	vector of strings defining field values to exclude from the set of potential founder annotations.
nperm	number of permutations to perform.
progress	if TRUE show a progress bar.

Value

An MRCATest object containing the test results and permutation realizations.

See Also

Uses getMRCA and getPathLengths. See plotMRCATest for plotting the permutation distributions.

Examples

```
# Define example tree set
graphs <- ExampleTrees[1-10]

# Perform MRCA test on isotypes
x <- testMRCA(graphs, "c_call", nperm=10)
print(x)
```

`translateDNA` *Translate nucleotide sequences to amino acids*

Description

`translateDNA` translates nucleotide sequences to amino acid sequences.

Usage

```
translateDNA(seq, trim = FALSE)
```

Arguments

`seq`	vector of strings defining DNA sequence(s) to be converted to translated.
`trim`	boolean flag to remove 3 nts from both ends of seq (converts IMGT junction to CDR3 region).

Value

A vector of translated sequence strings.

See Also

`translate`.

Examples

```
# Translate a single sequence
translateDNA("ACTGACTCGA")

# Translate a vector of sequences
translateDNA(ExampleDb$junction[1:3])

# Remove the first and last codon from the translation
translateDNA(ExampleDb$junction[1:3], trim=TRUE)
```

translateStrings *Translate a vector of strings*

Description

translateStrings modifies a character vector by substituting one or more strings with a replacement string.

Usage

```
translateStrings(strings, translation)
```

Arguments

strings	vector of character strings to modify.
translation	named character vector or a list of character vectors specifying the strings to replace (values) and their replacements (names).

Details

Does not perform partial replacements. Each translation value must match a complete strings value or it will not be replaced. Values that do not have a replacement named in the translation parameter will not be modified.

Replacement is accomplished using gsub.

Value

A modified strings vector.

See Also

See gsub for single value replacement in the base package.

Examples

```
# Using a vector translation
strings <- LETTERS[1:5]
translation <- c("POSITION1"="A", "POSITION5"="E")
translateStrings(strings, translation)

# Using a list translation
strings <- LETTERS[1:5]
translation <- list("1-3"=c("A","B","C"), "4-5"=c("D","E"))
translateStrings(strings, translation)
```

writeChangeoDb *Write a Change-O tab-delimited database file*

Description

writeChangeoDb is a simple wrapper around write_delim with defaults appropriate for writing a Change-O tab-delimited database file from a data.frame.

Usage

```
writeChangeoDb(data, file)
```

Arguments

data	data.frame of Change-O data.
file	output file name.

See Also

Wraps write_delim. See readChangeoDb for reading to Change-O files.

Examples

```
## Not run:
  # Write a database
  writeChangeoDb(data, "changeo.tsv")

## End(Not run)
```

Index