Package ‘shazam’

January 8, 2026

Type Package

Version 1.3.1

Date 2025-12-26

Title Immunoglobulin Somatic Hypermutation Analysis

Description Provides a computational framework for analyzing mutations in immunoglobulin (Ig) sequences. Includes methods for Bayesian estimation of antigen-driven selection pressure, mutational load quantification, building of somatic hypermutation (SHM) models, and model-dependent distance calculations. Also includes empirically derived models of SHM for both mice and humans.
Citations:
Gupta and Vander Heiden, et al (2015) <doi:10.1093/bioinformatics/btv359>,
Yaari, et al (2012) <doi:10.1093/nar/gks457>,
Yaari, et al (2013) <doi:10.3389/fimmu.2013.00358>,
Cui, et al (2016) <doi:10.4049/jimmunol.1502263>.

License AGPL-3

URL `http://shazam.readthedocs.io`

BugReports `https://github.com/immcantation/shazam/issues`

LazyData true

BuildVignettes true

VignetteBuilder knitr, rmarkdown

Encoding UTF-8

Depends R (>= 4.0), ggplot2 (>= 3.4.0)

Imports alakazam (>= 1.4.1), ape, diptest, doParallel, dplyr (>= 1.0), foreach, graphics, grid, igraph (>= 1.5.0), iterators, KernSmooth, lazyeval, MASS, methods, parallel, progress, rlang, scales, seqinr, stats, stringi (>= 1.1.3), tidyr, tidyselect, utils

Suggests knitr, rmarkdown, testthat, tibble

Collate 'Shazam.R' 'Core.R' 'RegionDefinitions.R' 'Baseline.R' 'ConvertNumbering.R' 'MutationProfiling.R' 'Deprecated.R' 'DistToNearest.R' 'MutationDefinitions.R' 'RegionsExtend.R' 'Shmulate.R' 'TargetingModels.R' 'kedd.R'

RoxygenNote 7.3.3

NeedsCompilation no

Author Mohamed Uduman [aut],
Namita Gupta [aut],
Susanna Marquez [aut, cre],
Julian Zhou [aut],
Nima Nouri [aut],
Noah Yann Lee [aut],
Ang Cui [ctb],
Jason Vander Heiden [aut],
Gur Yaari [aut],
Steven Kleinstein [aut, cph]

Maintainer Susanna Marquez <susanna.marquez@yale.edu>

Repository CRAN

Date/Publication 2026-01-08 08:40:31 UTC

Contents

`Baseline-class` *S4 class defining a BASELINe (selection) object*

Description

`Baseline` defines a common data structure the results of selection analysis using the BASELINe method.

Usage

```
## S4 method for signature 'Baseline,character'
plot(x, y, ...)

## S4 method for signature 'Baseline'
summary(object, nproc = 1)
```

Arguments

`x`	`Baseline` object.
`y`	name of the column in the `db` slot of `baseline` containing primary identifiers.
`...`	arguments to pass to plotBaselineDensity.
`object`	`Baseline` object.
`nproc`	number of cores to distribute the operation over.

Slots

- `description` `character` providing general information regarding the sequences, selection analysis and/or object.
- `db` `data.frame` containing annotation information about the sequences and selection results.
- `regionDefinition` RegionDefinition object defining the regions and boundaries of the Ig sequences.
- `testStatistic` `character` indicating the statistical framework used to test for selection. For example, `"local"` or `"focused"`.
- `regions` `character` vector defining the regions the BASELINe analysis was carried out on. For `"cdr"` and `"fwr"` or `"cdr1"`, `"cdr2"`, `"cdr3"`, etc.
- `numbOfSeqs` `matrix` of dimensions `r x c` containing the number of sequences or PDFs in each region, where:
  `r` = number of rows = number of groups or sequences.
  `c` = number of columns = number of regions.
- `binomK` `matrix` of dimensions `r x c` containing the number of successes in the binomial trials in each region, where:
  `r` = number of rows = number of groups or sequences.
  `c` = number of columns = number of regions.
- `binomN` `matrix` of dimensions `r x c` containing the total number of trials in the binomial in each region, where:
  `r` = number of rows = number of groups or sequences.
  `c` = number of columns = number of regions.
- `binomP` `matrix` of dimensions `r x c` containing the probability of success in one binomial trial in each region, where:
  `r` = number of rows = number of groups or sequences.
  `c` = number of columns = number of regions.
- `pdfs` `list` of matrices containing PDFs with one item for each defined region (e.g. `cdr` and `fwr`). Matrices have dimensions `r x c` dimensions, where:
  `r` = number of rows = number of sequences or groups.
  `c` = number of columns = length of the PDF (default 4001).

`stats`	`data.frame` of BASELINe statistics, including: mean selection strength (mean Sigma), 95% confidence intervals, and p-values with positive signs for the presence of positive selection and/or p-values with negative signs for the presence of negative selection.

See Also

See summarizeBaseline for more information on `@stats`.

---

calcBaseline — *Calculate the BASELINe PDFs (including for regions that include CDR3 and FWR4)*

Description

`calcBaseline` calculates the BASELINe posterior probability density functions (PDFs) for sequences in the given Change-O `data.frame`.

Usage

```
calcBaseline(
  db,
  sequenceColumn = "clonal_sequence",
  germlineColumn = "clonal_germline",
  testStatistic = c("local", "focused", "imbalanced"),
  regionDefinition = NULL,
  targetingModel = HH_S5F,
  mutationDefinition = NULL,
  calcStats = FALSE,
  nproc = 1,
  cloneColumn = NULL,
  juncLengthColumn = NULL
)
```

Arguments

`db`	`data.frame` containing sequence data and annotations.
`sequenceColumn`	`character` name of the column in db containing input sequences.
`germlineColumn`	`character` name of the column in db containing germline sequences.
`testStatistic`	`character` indicating the statistical framework used to test for selection. One of `c("local", "focused", "imbalanced")`.
`regionDefinition`	RegionDefinition object defining the regions and boundaries of the Ig sequences.
`targetingModel`	TargetingModel object. Default is HH_S5F.

`mutationDefinition`	MutationDefinition object defining replacement and silent mutation criteria. If `NULL` then replacement and silent are determined by exact amino acid identity. Note, if the input data.frame already contains observed and expected mutation frequency columns then mutations will not be recalculated and this argument will be ignored.
`calcStats`	`logical` indicating whether or not to calculate the summary statistics `data.frame` stored in the `stats` slot of a Baseline object.
`nproc`	number of cores to distribute the operation over. If `nproc=0` then the `cluster` has already been set and will not be reset.
`cloneColumn`	`character` name of the column in `db` containing clonal identifiers. Relevant only for when regionDefinition includes CDR and FWR4 (else this value can be `NULL`)
`juncLengthColumn`	`character` name of the column in `db` containing the junction length. Relevant only for when regionDefinition includes CDR and FWR4 (else this value can be `NULL`)

Details

Calculates the BASELINe posterior probability density function (PDF) for sequences in the provided `db`.

Note: Individual sequences within clonal groups are not, strictly speaking, independent events and it is generally appropriate to only analyze selection pressures on an effective sequence for each clonal group. For this reason, it is strongly recommended that the input `db` contains one effective sequence per clone. Effective clonal sequences can be obtained by calling the collapseClones function.

If the `db` does not contain the required columns to calculate the PDFs (namely mu_count & mu_expected) then the function will:

1. Calculate the numbers of observed mutations.
2. Calculate the expected frequencies of mutations and modify the provided `db`. The modified `db` will be included as part of the returned `Baseline` object.

The `testStatistic` indicates the statistical framework used to test for selection. E.g.

- `local` = CDR_R / (CDR_R + CDR_S).
- `focused` = CDR_R / (CDR_R + CDR_S + FWR_S).
- `imbalanced` = CDR_R + CDR_S / (CDR_R + CDR_S + FWR_S + FRW_R).

For `focused` the `regionDefinition` must only contain two regions. If more than two regions are defined the `local` test statistic will be used. For further information on the frame of these tests see Uduman et al. (2011).

Value

A Baseline object containing the modified `db` and BASELINe posterior probability density functions (PDF) for each of the sequences.

References

1. Hershberg U, et al. Improved methods for detecting selection by mutation analysis of Ig V region sequences. Int Immunol. 2008 20(5):683-94.
2. Uduman M, et al. Detecting selection in immunoglobulin sequences. Nucleic Acids Res. 2011 39(Web Server issue):W499-504.
3. Yaari G, et al. Models of somatic hypermutation targeting and substitution based on synonymous mutations from high-throughput immunoglobulin sequencing data. Front Immunol. 2013 4(November):358.

See Also

See Baseline for the return object. See groupBaseline and summarizeBaseline for further processing. See plotBaselineSummary and plotBaselineDensity for plotting results.

Examples

```
# Load and subset example data
data(ExampleDb, package="alakazam")
db <- subset(ExampleDb, c_call == "IGHG" & sample_id == "+7d")

# Collapse clones
db <- collapseClones(db, cloneColumn="clone_id",
                     sequenceColumn="sequence_alignment",
                     germlineColumn="germline_alignment_d_mask",
                     method="thresholdedFreq", minimumFrequency=0.6,
                     includeAmbiguous=FALSE, breakTiesStochastic=FALSE)

# Calculate BASELINe
baseline <- calcBaseline(db,
                         sequenceColumn="clonal_sequence",
                         germlineColumn="clonal_germline",
                         testStatistic="focused",
                         regionDefinition=IMGT_V,
                         targetingModel=HH_S5F,
                         nproc=1)
```

`calcExpectedMutations` *Calculate expected mutation frequencies of a sequence*

Description

`calcExpectedMutations` calculates the expected mutation frequencies of a given sequence. This is primarily a helper function for expectedMutations.

Usage

```
calcExpectedMutations(
  germlineSeq,
  inputSeq = NULL,
  targetingModel = HH_S5F,
  regionDefinition = NULL,
  mutationDefinition = NULL
)
```

Arguments

`germlineSeq`	germline (reference) sequence.
`inputSeq`	input (observed) sequence. If this is not `NULL`, then `germlineSeq` will be processed to be the same same length as `inputSeq` and positions in `germlineSeq` corresponding to positions with Ns in `inputSeq` will also be assigned an N.
`targetingModel`	TargetingModel object. Default is HH_S5F.
`regionDefinition`	RegionDefinition object defining the regions and boundaries of the Ig sequences.
`mutationDefinition`	MutationDefinition object defining replacement and silent mutation criteria. If `NULL` then replacement and silent are determined by exact amino acid identity.

Details

`calcExpectedMutations` calculates the expected mutation frequencies of a given sequence and its germline.

Note, only the part of the sequences defined in `regionDefinition` are analyzed. For example, when using the default IMGT_V definition, mutations in positions beyond 312 will be ignored.

Value

A `numeric` vector of the expected frequencies of mutations in the regions in the `regionDefinition`. For example, when using the default IMGT_V definition, which defines positions for CDR and FWR, the following columns are calculated:

- `mu_expected_cdr_r`: number of replacement mutations in CDR1 and CDR2 of the V-segment.
- `mu_expected_cdr_s`: number of silent mutations in CDR1 and CDR2 of the V-segment.
- `mu_expected_fwr_r`: number of replacement mutations in FWR1, FWR2 and FWR3 of the V-segment.
- `mu_expected_fwr_s`: number of silent mutations in FWR1, FWR2 and FWR3 of the V-segment.

See Also

expectedMutations calls this function. To create a custom `targetingModel` see createTargetingModel. See calcObservedMutations for getting observed mutation counts.

Examples

```
# Load example data
data(ExampleDb, package="alakazam")

# Use first entry in the exampled data for input and germline sequence
in_seq <- ExampleDb[["sequence_alignment"]][1]
germ_seq <-  ExampleDb[["germline_alignment_d_mask"]][1]

# Identify all mutations in the sequence
calcExpectedMutations(germ_seq,in_seq)

# Identify only mutations the V segment minus CDR3
calcExpectedMutations(germ_seq, in_seq, regionDefinition=IMGT_V)

# Define mutations based on hydropathy
calcExpectedMutations(germ_seq, in_seq, regionDefinition=IMGT_V,
                      mutationDefinition=HYDROPATHY_MUTATIONS)
```

`calcObservedMutations` *Count the number of observed mutations in a sequence.*

Description

`calcObservedMutations` determines all the mutations in a given input sequence compared to its germline sequence.

Usage

```
calcObservedMutations(
  inputSeq,
  germlineSeq,
  regionDefinition = NULL,
  mutationDefinition = NULL,
  ambiguousMode = c("eitherOr", "and"),
  returnRaw = FALSE,
  frequency = FALSE
)
```

Arguments

`inputSeq`	input sequence. IUPAC ambiguous characters for DNA are supported.
`germlineSeq`	germline sequence. IUPAC ambiguous characters for DNA are supported.
`regionDefinition`	RegionDefinition object defining the regions and boundaries of the Ig sequences. Note, only the part of sequences defined in `regionDefinition` are analyzed. If NULL, mutations are counted for entire sequence.

`mutationDefinition`	MutationDefinition object defining replacement and silent mutation criteria. If `NULL` then replacement and silent are determined by exact amino acid identity.
`ambiguousMode`	whether to consider ambiguous characters as `"either or"` or `"and"` when determining and counting the type(s) of mutations. Applicable only if `inputSeq` and/or `germlineSeq` contain(s) ambiguous characters. One of `c("eitherOr", "and")`. Default is `"eitherOr"`.
`returnRaw`	return the positions of point mutations and their corresponding mutation types, as opposed to counts of mutations across positions. Also returns the number of bases used as the denominator when calculating frequency. Default is `FALSE`.
`frequency`	`logical` indicating whether or not to calculate mutation frequencies. The denominator used is the number of bases that are not one of "N", "-", or "." in either the input or the germline sequences. If set, this overwrites `returnRaw`. Default is `FALSE`.

Details

Each mutation is considered independently in the germline context. For illustration, consider the case where the germline is `TGG` and the observed is `TAC`. When determining the mutation type at position 2, which sees a change from `G` to `A`, we compare the codon `TGG` (germline) to `TAG` (mutation at position 2 independent of other mutations in the germline context). Similarly, when determining the mutation type at position 3, which sees a change from `G` to `C`, we compare the codon `TGG` (germline) to `TGC` (mutation at position 3 independent of other mutations in the germline context).

If specified, only the part of `inputSeq` defined in `regionDefinition` is analyzed. For example, when using the default IMGT_V definition, then mutations in positions beyond 312 will be ignored. Additionally, non-triplet overhang at the sequence end is ignored.

Only replacement (R) and silent (S) mutations are included in the results. **Excluded** are:

- Stop mutations

  E.g.: the case where `TAGTGG` is observed for the germline `TGGTGG`.
- Mutations occurring in codons where one or both of the observed and the germline involve(s) one or more of "N", "-", or ".".

  E.g.: the case where `TTG` is observed for the germline being any one of `TNG`, `.TG`, or `-TG`. Similarly, the case where any one of `TTN`, `TT.`, or `TT-` is observed for the germline `TTG`.

In other words, a result that is `NA` or zero indicates absence of R and S mutations, not necessarily all types of mutations, such as the excluded ones mentioned above.

`NA` is also returned if `inputSeq` or `germlineSeq` is shorter than 3 nucleotides.

Value

For `returnRaw=FALSE`, an `array` with the numbers of replacement (R) and silent (S) mutations.

For `returnRaw=TRUE`, a list containing

- `$pos`: A data frame whose columns (`position`, `r`, `s`, and `region`) indicate, respectively, the nucleotide position, the number of R mutations at that position, the number of S mutations at that position, and the region in which that nucleotide is in.

- `$nonN`: A vector indicating the number of bases in regions defined by `regionDefinition` (excluding non-triplet overhang, if any) that are not one of "N", "-", or "." in either the `inputSeq` or `germlineSeq`.

For `frequency=TRUE`, regardless of `returnRaw`, an `array` with the frequencies of replacement (R) and silent (S) mutations.

Ambiguous characters

When there are ambiguous characters present, the user could choose how mutations involving ambiguous characters are counted through `ambiguousMode`. The two available modes are `"eitherOr"` and `"and"`.

- With `"eitherOr"`, ambiguous characters are each expanded but only 1 mutation is recorded. When determining the type of mutation, the priority for different types of mutations, in decreasing order, is as follows: no mutation, replacement mutation, silent mutation, and stop mutation.

  When counting the number of non-N, non-dash, and non-dot positions, each position is counted only once, regardless of the presence of ambiguous characters.

  As an example, consider the case where `germlineSeq` is `"TST"` and `inputSeq` is `"THT"`. Expanding `"H"` at position 2 in `inputSeq` into `"A"`, `"C"`, and `"T"`, as well as expanding `"S"` at position 2 in `germlineSeq` into `"C"` and `"G"`, one gets:

  - `"TCT"` (germline) to `"TAT"` (observed): replacement
  - `"TCT"` (germline) to `"TCT"` (observed): no mutation
  - `"TCT"` (germline) to `"TTT"` (observed): replacement
  - `"TGT"` (germline) to `"TAT"` (observed): replacement
  - `"TGT"` (germline) to `"TCT"` (observed): replacement
  - `"TGT"` (germline) to `"TTT"` (observed): replacement

  Because "no mutation" takes priority over replacement mutation, the final mutation count returned for this example is `NA` (recall that only R and S mutations are returned). The number of non-N, non-dash, and non-dot positions is 3.

- With `"and"`, ambiguous characters are each expanded and mutation(s) from all expansions are recorded.

  When counting the number of non-N, non-dash, and non-dot positions, if a position contains ambiguous character(s) in `inputSeq` and/or `germlineSeq`, the count at that position is taken to be the total number of combinations of germline and observed codons after expansion.

  Using the same example from above, the final result returned for this example is that there are 5 R mutations at position 2. The number of non-N, non-dash, and non-dot positions is 8, since there are 6 combinations stemming from position 2 after expanding the germline codon (`"TST"`) and the observed codon (`"THT"`).

See Also

See observedMutations for counting the number of observed mutations in a `data.frame`.

Examples

```
# Use an entry in the example data for input and germline sequence
data(ExampleDb, package="alakazam")
in_seq <- ExampleDb[["sequence_alignment"]][100]
germ_seq <-  ExampleDb[["germline_alignment_d_mask"]][100]

# Identify all mutations in the sequence
ex1_raw <- calcObservedMutations(in_seq, germ_seq, returnRaw=TRUE)
# Count all mutations in the sequence
ex1_count <- calcObservedMutations(in_seq, germ_seq, returnRaw=FALSE)
ex1_freq <- calcObservedMutations(in_seq, germ_seq, returnRaw=FALSE, frequency=TRUE)
# Compare this with ex1_count
table(ex1_raw$pos$region, ex1_raw$pos$r)[, "1"]
table(ex1_raw$pos$region, ex1_raw$pos$s)[, "1"]
# Compare this with ex1_freq
table(ex1_raw$pos$region, ex1_raw$pos$r)[, "1"]/ex1_raw$nonN
table(ex1_raw$pos$region, ex1_raw$pos$s)[, "1"]/ex1_raw$nonN

# Identify only mutations the V segment minus CDR3
ex2_raw <- calcObservedMutations(in_seq, germ_seq,
                                regionDefinition=IMGT_V, returnRaw=TRUE)
# Count only mutations the V segment minus CDR3
ex2_count <- calcObservedMutations(in_seq, germ_seq,
                                  regionDefinition=IMGT_V, returnRaw=FALSE)
ex2_freq <- calcObservedMutations(in_seq, germ_seq,
                                 regionDefinition=IMGT_V, returnRaw=FALSE,
                                 frequency=TRUE)
# Compare this with ex2_count
table(ex2_raw$pos$region, ex2_raw$pos$r)[, "1"]
table(ex2_raw$pos$region, ex2_raw$pos$s)[, "1"]
# Compare this with ex2_freq
table(ex2_raw$pos$region, ex2_raw$pos$r)[, "1"]/ex2_raw$nonN
table(ex2_raw$pos$region, ex2_raw$pos$s)[, "1"]/ex2_raw$nonN

# Identify mutations by change in hydropathy class
ex3_raw <- calcObservedMutations(in_seq, germ_seq, regionDefinition=IMGT_V,
                                mutationDefinition=HYDROPATHY_MUTATIONS,
                                returnRaw=TRUE)
# Count mutations by change in hydropathy class
ex3_count <- calcObservedMutations(in_seq, germ_seq, regionDefinition=IMGT_V,
                                  mutationDefinition=HYDROPATHY_MUTATIONS,
                                  returnRaw=FALSE)
ex3_freq <- calcObservedMutations(in_seq, germ_seq, regionDefinition=IMGT_V,
                                 mutationDefinition=HYDROPATHY_MUTATIONS,
                                 returnRaw=FALSE, frequency=TRUE)
# Compare this with ex3_count
table(ex3_raw$pos$region, ex3_raw$pos$r)[, "1"]
table(ex3_raw$pos$region, ex3_raw$pos$s)[, "1"]
# Compare this with ex3_freq
table(ex3_raw$pos$region, ex3_raw$pos$r)[, "1"]/ex3_raw$nonN
table(ex3_raw$pos$region, ex3_raw$pos$s)[, "1"]/ex3_raw$nonN
```

calcTargetingDistance *Calculates a 5-mer distance matrix from a TargetingModel object*

Description

`calcTargetingDistance` converts either the targeting rates in a `TargetingModel` model to a matrix of 5-mer to single-nucleotide mutation distances, or the substitution rates in a 1-mer substitution model to a symmetric distance matrix.

Usage

```
calcTargetingDistance(model, places = 2)
```

Arguments

`model`	TargetingModel object with mutation likelihood information, or a 4x4 1-mer substitution matrix normalized by row with rownames and colnames consisting of "A", "T", "G", and "C".
`places`	decimal places to round distances to.

Details

The targeting model is transformed into a distance matrix by:

1. Converting the likelihood of being mutated $p = mutability * substitution$ to distance $d = -log10(p)$.
2. Dividing this distance by the mean of the distances.
3. Converting all infinite, no change (e.g., A->A), and NA distances to zero.

The 1-mer substitution matrix is transformed into a distance matrix by:

1. Symmetrize the 1-mer substitution matrix.
2. Converting the rates to distance $d = -log10(p)$.
3. Dividing this distance by the mean of the distances.
4. Converting all infinite, no change (e.g., A -> A), and NA distances to zero.

Value

For input of TargetingModel, a matrix of distances for each 5-mer motif with rows names defining the center nucleotide and column names defining the 5-mer nucleotide sequence. For input of 1-mer substitution matrix, a 4x4 symmetric distance matrix.

See Also

See TargetingModel for this class of objects and createTargetingModel for building one.

Examples

```
# Calculate targeting distance of HH_S5F
dist <- calcTargetingDistance(HH_S5F)

# Calculate targeting distance of HH_S1F
dist <- calcTargetingDistance(HH_S1F)
```

`calculateMutability` *Calculate total mutability*

Description

`calculateMutability` calculates the total (summed) mutability for a set of sequences based on a 5-mer nucleotide mutability model.

Usage

```
calculateMutability(sequences, model = HH_S5F, progress = FALSE)
```

Arguments

`sequences`	character vector of sequences.
`model`	TargetingModel object with mutation likelihood information.
`progress`	if `TRUE` print a progress bar.

Value

Numeric vector with a total mutability score for each sequence.

Examples

```
# Subset example data to one isotype and sample as a demo
data(ExampleDb, package="alakazam")
db <- subset(ExampleDb, c_call == "IGHA" & sample_id == "-1h")

# Calculate mutability of germline sequences using \link{HH_S5F} model
mutability <- calculateMutability(sequences=db[["germline_alignment_d_mask"]], model=HH_S5F)
```

collapseClones — *Constructs effective clonal sequences for all clones*

Description

`collapseClones` creates effective input and germline sequences for each clonal group and appends columns containing the consensus sequences to the input `data.frame`.

Usage

```
collapseClones(
  db,
  cloneColumn = "clone_id",
  sequenceColumn = "sequence_alignment",
  germlineColumn = "germline_alignment_d_mask",
  muFreqColumn = NULL,
  regionDefinition = NULL,
 method = c("mostCommon", "thresholdedFreq", "catchAll", "mostMutated", "leastMutated"),
  minimumFrequency = NULL,
  includeAmbiguous = FALSE,
  breakTiesStochastic = FALSE,
  breakTiesByColumns = NULL,
  expandedDb = FALSE,
  nproc = 1,
  juncLengthColumn = "junction_length",
  fields = NULL
)
```

Arguments

`db`	`data.frame` containing sequence data. Required.
`cloneColumn`	`character` name of the column containing clonal identifiers. Required.
`sequenceColumn`	`character` name of the column containing input sequences. Required. The length of each input sequence should match that of its corresponding germline sequence.
`germlineColumn`	`character` name of the column containing germline sequences. Required. The length of each germline sequence should match that of its corresponding input sequence.
`muFreqColumn`	`character` name of the column containing mutation frequency. Optional. Applicable to the `"mostMutated"` and `"leastMutated"` methods. If not supplied, mutation frequency is computed by calling `observedMutations`. Default is `NULL`. See Cautions for note on usage.
`regionDefinition`	RegionDefinition object defining the regions and boundaries of the Ig sequences. Optional. Default is `NULL`.

`method`	method for calculating input consensus sequence. Required. One of `"thresholdedFreq"`, `"mostCommon"`, `"catchAll"`, `"mostMutated"`, or `"leastMutated"`. See "Methods" for details.
`minimumFrequency`	frequency threshold for calculating input consensus sequence. Applicable to and required for the `"thresholdedFreq"` method. A canonical choice is 0.6. Default is `NULL`.
`includeAmbiguous`	whether to use ambiguous characters to represent positions at which there are multiple characters with frequencies that are at least `minimumFrequency` or that are maximal (i.e. ties). Applicable to and required for the `"thresholdedFreq"` and `"mostCommon"` methods. Default is `FALSE`. See "Choosing ambiguous characters" for rules on choosing ambiguous characters.
`breakTiesStochastic`	In case of ties, whether to randomly pick a sequence from sequences that fulfill the criteria as consensus. Applicable to and required for all methods except for `"catchAll"`. Default is `FALSE`. See "Methods" for details.
`breakTiesByColumns`	A list of the form `list(c(col_1, col_2, ...), c(fun_1, fun_2, ...))`, where `col_i` is a `character` name of a column in `db`, and `fun_i` is a function to be applied on that column. Currently, only `max` and `min` are supported. Note that the two `c()`'s in `list()` are essential (i.e. if there is only 1 column, the list should be of the form `list(c(col_1), c(func_1))`. Applicable to and optional for the `"mostMutated"` and `"leastMutated"` methods. If supplied, `fun_i`'s are applied on `col_i`'s to help break ties. Default is `NULL`. See "Methods" for details.
`expandedDb`	`logical` indicating whether or not to return the expanded `db`, containing all the sequences (as opposed to returning just one sequence per clone).
`nproc`	Number of cores to distribute the operation over. If the `cluster` has already been set earlier, then pass the `cluster`. This will ensure that it is not reset.
`juncLengthColumn`	`character` name of the column containing the junction length. Needed when `regionDefinition` includes CDR3 and FWR4.
`fields`	additional fields used for grouping. Use sample_id, to avoid combining sequences with the same clone_id that belong to different sample_id.

Value

A modified `db` with the following additional columns:

- `clonal_sequence`: effective sequence for the clone.
- `clonal_germline`: germline sequence for the clone.
- `clonal_sequence_mufreq`: mutation frequency of `clonal_sequence`; only added for the `"mostMutated"` and `"leastMutated"` methods.

`clonal_sequence` is generated with the method of choice indicated by `method`, and `clonal_germline` is generated with the `"mostCommon"` method, along with, where applicable, user-defined parameters such as `minimumFrequency`, `includeAmbiguous`, `breakTiesStochastic`, and `breakTiesByColumns`.

Consensus lengths

For each clone, `clonal_sequence` and `clonal_germline` have the same length.

- For the `"thresholdedFreq"`, `"mostCommon"`, and `"catchAll"` methods:

  The length of the consensus sequences is determined by the longest possible consensus sequence (based on `inputSeq` and `germlineSeq`) and `regionDefinition@seqLength` (if supplied), whichever is shorter.

  Given a set of sequences of potentially varying lengths, the longest possible length of their consensus sequence is taken to be the longest length along which there is information contained at every nucleotide position across majority of the sequences. Majority is defined to be greater than `floor(n/2)`, where `n` is the number of sequences. If the longest possible consensus length is 0, there will be a warning and an empty string (`""`) will be returned.

  If a length limit is defined by supplying a `regionDefinition` via `regionDefinition@seqLength`, the consensus length will be further restricted to the shorter of the longest possible length and `regionDefinition@seqLength`.

- For the `"mostMutated"` and `"leastMutated"` methods:

  The length of the consensus sequences depends on that of the most/least mutated input sequence, and, if supplied, the length limit defined by `regionDefinition@seqLength`, whichever is shorter. If the germline consensus computed using the `"mostCommon"` method is longer than the most/least mutated input sequence, the germline consensus is trimmed to be of the same length as the input consensus.

Methods

The descriptions below use "sequences" as a generalization of input sequences and germline sequences.

- `method="thresholdedFreq"`

  A threshold must be supplied to the argument `minimumFrequency`. At each position along the length of the consensus sequence, the frequency of each nucleotide/character across sequences is tabulated. The nucleotide/character whose frequency is at least (i.e. `>=`) `minimumFrequency` becomes the consensus; if there is none, the consensus nucleotide will be `"N"`.

  When there are ties (frequencies of multiple nucleotides/characters are at least `minimumFrequency`), this method can be deterministic or stochastic, depending on additional parameters.

  - With `includeAmbiguous=TRUE`, ties are resolved deterministically by representing ties using ambiguous characters. See "Choosing ambiguous characters" for how ambiguous characters are chosen.
  - With `breakTiesStochastic=TRUE`, ties are resolved stochastically by randomly picking a character among the ties.
  - When both `TRUE`, `includeAmbiguous` takes precedence over `breakTiesStochastic`.
  - When both `FALSE`, the first character from the ties is taken to be the consensus following the order of `"A"`, `"T"`, `"G"`, `"C"`, `"N"`, `"."`, and `"-"`.

  Below are some examples looking at a single position based on 5 sequences with `minimumFrequency=0.6`, `includeAmbiguous=FALSE`, and `breakTiesStochastic=FALSE`:

  - If the sequences have `"A"`, `"A"`, `"A"`, `"T"`, `"C"`, the consensus will be `"A"`, because `"A"` has frequency 0.6, which is at least `minimumFrequency`.

  - If the sequences have `"A"`, `"A"`, `"T"`, `"T"`, `"C"`, the consensus will be `"N"`, because none of `"A"`, `"T"`, or `"C"` has frequency that is at least `minimumFrequency`.

- `method="mostCommon"`

  The most frequent nucleotide/character across sequences at each position along the length of the consensus sequence makes up the consensus.

  When there are ties (multiple nucleotides/characters with equally maximal frequencies), this method can be deterministic or stochastic, depending on additional parameters. The same rules for breaking ties for `method="thresholdedFreq"` apply.

  Below are some examples looking at a single position based on 5 sequences with `includeAmbiguous=FALSE`, and `breakTiesStochastic=FALSE`:

  - If the sequences have `"A"`, `"A"`, `"T"`, `"A"`, `"C"`, the consensus will be `"A"`.
  - If the sequences have `"T"`, `"T"`, `"C"`, `"C"`, `"G"`, the consensus will be `"T"`, because `"T"` is before `"C"` in the order of `"A"`, `"T"`, `"G"`, `"C"`, `"N"`, `"."`, and `"-"`.

- `method="catchAll"`

  This method returns a consensus sequence capturing most of the information contained in the sequences. Ambiguous characters are used where applicable. See "Choosing ambiguous characters" for how ambiguous characters are chosen. This method is deterministic and does not involve breaking ties.

  Below are some examples for `method="catchAll"` looking at a single position based on 5 sequences:

  - If the sequences have `"N"`, `"N"`, `"N"`, `"N"`, `"N"`, the consensus will be `"N"`.
  - If the sequences have `"N"`, `"A"`, `"A"`, `"A"`, `"A"`, the consensus will be `"A"`.
  - If the sequences have `"N"`, `"A"`, `"G"`, `"A"`, `"A"`, the consensus will be `"R"`.
  - If the sequences have `"-"`, `"-"`, `"."`, `"."`, `"."`, the consensus will be `"-"`.
  - If the sequences have `"-"`, `"-"`, `"-"`, `"-"`, `"-"`, the consensus will be `"-"`.
  - If the sequences have `"."`, `"."`, `"."`, `"."`, `"."`, the consensus will be `"."`.

- `method="mostMutated"` and `method="leastMutated"`

  These methods return the most/least mutated sequence as the consensus sequence.

  When there are ties (multiple sequences have the maximal/minimal mutation frequency), this method can be deterministic or stochastic, depending on additional parameters.

  - With `breakTiesStochastic=TRUE`, ties are resolved stochastically by randomly picking a sequence out of sequences with the maximal/minimal mutation frequency.
  - When `breakTiesByColumns` is supplied, ties are resolved deterministically. Column by column, a function is applied on the column and sequences with column value matching the functional value are retained, until ties are resolved or columns run out. In the latter case, the first remaining sequence is taken as the consensus.
  - When `breakTiesStochastic=TRUE` and `breakTiesByColumns` is also supplied, `breakTiesStochastic` takes precedence over `breakTiesByColumns`.
  - When `breakTiesStochastic=FALSE` and `breakTiesByColumns` is not supplied (i.e. `NULL`), the sequence that appears first among the ties is taken as the consensus.

Choosing ambiguous characters

Ambiguous characters may be present in the returned consensuses when using the `"catchAll"` method and when using the `"thresholdedFreq"` or `"mostCommon"` methods with `includeAmbiguous=TRUE`.

The rules on choosing ambiguous characters are as follows:

- If a position contains only `"N"` across sequences, the consensus at that position is `"N"`.
- If a position contains one or more of `"A"`, `"T"`, `"G"`, or `"C"`, the consensus will be an IUPAC character representing all of the characters present, regardless of whether `"N"`, `"-"`, or `"."` is present.
- If a position contains only `"-"` and `"."` across sequences, the consensus at that position is taken to be `"-"`.
- If a position contains only one of `"-"` or `"."` across sequences, the consensus at that position is taken to be the character present.

Cautions

- Note that this function does not perform multiple sequence alignment. As a prerequisite, it is assumed that the sequences in `sequenceColumn` and `germlineColumn` have been aligned somehow. In the case of immunoglobulin repertoire analysis, this usually means that the sequences are IMGT-gapped.
- When using the `"mostMutated"` and `"leastMutated"` methods, if you supply both `muFreqColumn` and `regionDefinition`, it is your responsibility to ensure that the mutation frequency in `muFreqColumn` was calculated with sequence lengths restricted to the **same** `regionDefinition` you are supplying. Otherwise, the "most/least mutated" sequence you obtain might not be the most/least mutated given the `regionDefinition` supplied, because your mutation frequency was based on a `regionDefinition` different from the one supplied.
- If you intend to run `collapseClones` before building a 5-mer targeting model, you **must** choose parameters such that your collapsed clonal consensuses do **not** include ambiguous characters. This is because the targeting model functions do NOT support ambiguous characters in their inputs.

See Also

See IMGT_SCHEMES for a set of predefined RegionDefinition objects.

Examples

```
# Subset example data
data(ExampleDb, package="alakazam")
db <- subset(ExampleDb, c_call %in% c("IGHA", "IGHG") & sample_id == "+7d" &
                        clone_id %in% c("3100", "3141", "3184"))

# thresholdedFreq method, resolving ties deterministically without using ambiguous characters
clones <- collapseClones(db, cloneColumn="clone_id", sequenceColumn="sequence_alignment",
                         germlineColumn="germline_alignment_d_mask",
                         method="thresholdedFreq", minimumFrequency=0.6,
                         includeAmbiguous=FALSE, breakTiesStochastic=FALSE)

# mostCommon method, resolving ties deterministically using ambiguous characters
clones <- collapseClones(db, cloneColumn="clone_id", sequenceColumn="sequence_alignment",
                         germlineColumn="germline_alignment_d_mask",
                         method="mostCommon",
                         includeAmbiguous=TRUE, breakTiesStochastic=FALSE)
```

```
# Make a copy of db that has a mutation frequency column
db2 <- observedMutations(db, frequency=TRUE, combine=TRUE)

# mostMutated method, resolving ties stochastically
clones <- collapseClones(db2, cloneColumn="clone_id", sequenceColumn="sequence_alignment",
                         germlineColumn="germline_alignment_d_mask",
                         method="mostMutated", muFreqColumn="mu_freq",
                         breakTiesStochastic=TRUE, breakTiesByColumns=NULL)

# mostMutated method, resolving ties deterministically using additional columns
clones <- collapseClones(db2, cloneColumn="clone_id", sequenceColumn="sequence_alignment",
                         germlineColumn="germline_alignment_d_mask",
                         method="mostMutated", muFreqColumn="mu_freq",
                         breakTiesStochastic=FALSE,
                         breakTiesByColumns=list(c("duplicate_count"), c(max)))

# Build consensus for V segment only
# Capture all nucleotide variations using ambiguous characters
clones <- collapseClones(db, cloneColumn="clone_id", sequenceColumn="sequence_alignment",
                         germlineColumn="germline_alignment_d_mask",
                         method="catchAll", regionDefinition=IMGT_V)

# Return the same number of rows as the input
clones <- collapseClones(db, cloneColumn="clone_id", sequenceColumn="sequence_alignment",
                         germlineColumn="germline_alignment_d_mask",
                         method="mostCommon", expandedDb=TRUE)
```

---

consensusSequence *Construct a consensus sequence*

Description

Construct a consensus sequence

Usage

```
consensusSequence(
  sequences,
  db = NULL,
 method = c("mostCommon", "thresholdedFreq", "catchAll", "mostMutated", "leastMutated"),
  minFreq = NULL,
  muFreqColumn = NULL,
  lenLimit = NULL,
  includeAmbiguous = FALSE,
  breakTiesStochastic = FALSE,
  breakTiesByColumns = NULL
)
```

Arguments

`sequences`	character vector of sequences.
`db`	`data.frame` containing sequence data for a single clone. Applicable to and required for the `"mostMutated"` and `"leastMutated"` methods. Default is `NULL`.
`method`	method to calculate consensus sequence. One of `"thresholdedFreq"`, `"mostCommon"`, `"catchAll"`, `"mostMutated"`, or `"leastMutated"`. See "Methods" under collapseClones for details.
`minFreq`	frequency threshold for calculating input consensus sequence. Applicable to and required for the `"thresholdedFreq"` method. A canonical choice is 0.6. Default is `NULL`.
`muFreqColumn`	`character` name of the column in db containing mutation frequency. Applicable to and required for the `"mostMutated"` and `"leastMutated"` methods. Default is `NULL`.
`lenLimit`	limit on consensus length. if `NULL` then no length limit is set.
`includeAmbiguous`	whether to use ambiguous characters to represent positions at which there are multiple characters with frequencies that are at least `minimumFrequency` or that are maximal (i.e. ties). Applicable to and required for the `"thresholdedFreq"` and `"mostCommon"` methods. Default is `FALSE`. See "Choosing ambiguous characters" under collapseClones for rules on choosing ambiguous characters. Note: this argument refers to the use of ambiguous nucleotides in the output consensus sequence. Ambiguous nucleotides in the input sequences are allowed for methods catchAll, mostMutated and leastMutated.
`breakTiesStochastic`	In case of ties, whether to randomly pick a sequence from sequences that fulfill the criteria as consensus. Applicable to and required for all methods except for `"catchAll"`. Default is `FALSE`. See "Methods" under collapseClones for details.
`breakTiesByColumns`	A list of the form `list(c(col_1, col_2, ...), c(fun_1, fun_2, ...))`, where `col_i` is a `character` name of a column in `db`, and `fun_i` is a function to be applied on that column. Currently, only `max` and `min` are supported. Note that the two `c()`'s in `list()` are essential (i.e. if there is only 1 column, the list should be of the form `list(c(col_1), c(func_1))`. Applicable to and optional for the `"mostMutated"` and `"leastMutated"` methods. If supplied, `fun_i`'s are applied on `col_i`'s to help break ties. Default is `NULL`. See "Methods" under collapseClones for details.

Details

See collapseClones for detailed documentation on methods and additional parameters.

Value

A list containing `cons`, which is a character string that is the consensus sequence for `sequences`; and `muFreq`, which is the maximal/minimal mutation frequency of the consensus sequence for the `"mostMutated"` and `"leastMutated"` methods, or `NULL` for all other methods.

Examples

```
# Subset example data
data(ExampleDb, package="alakazam")
db <- subset(ExampleDb, c_call %in% c("IGHA", "IGHG") & sample_id == "+7d")
clone <- subset(db, clone_id == "3192")

# First compute mutation frequency for most/leastMutated methods
clone <- observedMutations(clone, frequency=TRUE, combine=TRUE)

# Manually create a tie
clone <- rbind(clone, clone[which.max(clone$mu_freq), ])

# ThresholdedFreq method.
# Resolve ties deterministically without using ambiguous characters
cons1 <- consensusSequence(clone$sequence_alignment,
                           method="thresholdedFreq", minFreq=0.3,
                           includeAmbiguous=FALSE,
                           breakTiesStochastic=FALSE)
cons1$cons
```

`convertNumbering` *convertNumbering: IMGT-Kabat number conversion*

Description

Converts numbering systems like Kabat or IMGT using these conventions: http://www.imgt.org/IMGTScientificChart/Numbe
Kabat_part1.html with Gaps (unoccupied positions) shown by "G" and Asterisks (*) shown by "S": arbitrary mappings (multiple possible "to" values) represented with "NA"

Usage

```
convertNumbering(locus, from, to, calls)
```

Arguments

`locus`	string indicating heavy ("IGH") or light chains ("IGK" or "IGL)
`from`	string indicating numbering system to convert to ("IMGT" or "KABAT")
`to`	string indicating original numbering system ("IMGT" or "KABAT")
`calls`	vector of strings representing original numbering

Value

A vector of string indicating the corresponding numbering

Examples

```
convertNumbering("IGH", "IMGT", "KABAT", c("51", "23", "110"))
convertNumbering("IGH", "KABAT", "IMGT", c("51", "23", "G"))
```

createBaseline *Creates a Baseline object*

Description

`createBaseline` creates and initialize a `Baseline` object.

Usage

```
createBaseline(
  description = "",
  db = data.frame(),
  regionDefinition = createRegionDefinition(),
  testStatistic = "",
  regions = NULL,
  numbOfSeqs = matrix(),
  binomK = matrix(),
  binomN = matrix(),
  binomP = matrix(),
  pdfs = list(),
  stats = data.frame()
)
```

Arguments

`description`	`character` providing general information regarding the sequences, selection analysis and/or object.
`db`	`data.frame` containing annotation information about the sequences and selection results.
`regionDefinition`	RegionDefinition object defining the regions and boundaries of the Ig sequences.
`testStatistic`	`character` indicating the statistical framework used to test for selection. For example, `"local"` or `"focused"` or `"imbalanced"`.
`regions`	`character` vector defining the regions the BASELINe analysis was carried out on. For `"cdr"` and `"fwr"` or `"cdr1"`, `"cdr2"`, `"cdr3"`, etc. If `NULL` then regions will be determined automatically from `regionDefinition`.
`numbOfSeqs`	`matrix` of dimensions `r` x `c` containing the number of sequences or PDFs in each region, where: `r` = number of rows = number of groups or sequences. `c` = number of columns = number of regions.
`binomK`	`matrix` of dimensions `r` x `c` containing the number of successes in the binomial trials in each region, where: `r` = number of rows = number of groups or sequences. `c` = number of columns = number of regions.

binomN	`matrix` of dimensions `r x c` containing the total number of trials in the binomial in each region, where: `r` = number of rows = number of groups or sequences. `c` = number of columns = number of regions.
binomP	`matrix` of dimensions `r x c` containing the probability of success in one binomial trial in each region, where: `r` = number of rows = number of groups or sequences. `c` = number of columns = number of regions.
pdfs	`list` of matrices containing PDFs with one item for each defined region (e.g. `cdr` and `fwr`). Matrices have dimensions `r x c` dimensions, where: `r` = number of rows = number of sequences or groups. `c` = number of columns = length of the PDF (default 4001).
stats	`data.frame` of BASELINe statistics, including: mean selection strength (mean Sigma), 95% confidence intervals, and p-values with positive signs for the presence of positive selection and/or p-values with negative signs for the presence of negative selection.

Details

Create and initialize a `Baseline` object.

The `testStatistic` indicates the statistical framework used to test for selection. For example,

- `local` = CDR_R / (CDR_R + CDR_S).
- `focused` = CDR_R / (CDR_R + CDR_S + FWR_S).
- `immbalance` = CDR_R + CDR_s / (CDR_R + CDR_S + FWR_S + FWR_R)

For `focused` the `regionDefinition` must only contain two regions. If more than two regions are defined, then the `local` test statistic will be used. For further information on the frame of these tests see Uduman et al. (2011).

Value

A `Baseline` object.

References

1. Hershberg U, et al. Improved methods for detecting selection by mutation analysis of Ig V region sequences. Int Immunol. 2008 20(5):683-94.
2. Uduman M, et al. Detecting selection in immunoglobulin sequences. Nucleic Acids Res. 2011 39(Web Server issue):W499-504.
3. Yaari G, et al. Models of somatic hypermutation targeting and substitution based on synonymous mutations from high-throughput immunoglobulin sequencing data. Front Immunol. 2013 4(November):358.

See Also

See Baseline for the return object.

Examples

```
# Creates an empty Baseline object
createBaseline()
```

`createMutabilityMatrix` *Builds a mutability model*

Description

`createMutabilityMatrix` builds a 5-mer nucleotide mutability model by counting the number of mutations occurring in the center position for all 5-mer motifs.

Usage

```
createMutabilityMatrix(
  db,
  substitutionModel,
  model = c("s", "rs"),
  sequenceColumn = "sequence_alignment",
  germlineColumn = "germline_alignment_d_mask",
  vCallColumn = "v_call",
  multipleMutation = c("independent", "ignore"),
  minNumSeqMutations = 500,
  numSeqMutationsOnly = FALSE
)
```

Arguments

`db`	data.frame containing sequence data.
`substitutionModel`	matrix of 5-mer substitution rates built by createSubstitutionMatrix. Note, this model will only impact mutability scores when `model="s"` (using only silent mutations).
`model`	type of model to create. The default model, "s", builds a model by counting only silent mutations. `model="s"` should be used for data that includes functional sequences. Setting `model="rs"` creates a model by counting both replacement and silent mutations and may be used on fully non-functional sequence data sets.
`sequenceColumn`	name of the column containing IMGT-gapped sample sequences.
`germlineColumn`	name of the column containing IMGT-gapped germline sequences.
`vCallColumn`	name of the column containing the V-segment allele call.
`multipleMutation`	string specifying how to handle multiple mutations occurring within the same 5-mer. If `"independent"` then multiple mutations within the same 5-mer are counted independently. If `"ignore"` then 5-mers with multiple mutations are excluded from the total mutation tally.

`minNumSeqMutations`
: minimum number of mutations in sequences containing each 5-mer to compute the mutability rates. If the number is smaller than this threshold, the mutability for the 5-mer will be inferred. Default is 500. Not required if `numSeqMutationsOnly=TRUE`.

`numSeqMutationsOnly`
: when `TRUE`, return only a vector counting the number of observed mutations in sequences containing each 5-mer. This option can be used for parameter tuning for `minNumSeqMutations` during preliminary analysis using minNumSeqMutationsTune. Default is `FALSE`.

Details

Caution: The targeting model functions do NOT support ambiguous characters in their inputs. You MUST make sure that your input and germline sequences do NOT contain ambiguous characters (especially if they are clonal consensuses returned from `collapseClones`).

Value

When `numSeqMutationsOnly` is `FALSE`, a `MutabilityModel` containing a named numeric vector of 1024 normalized mutability rates for each 5-mer motif with names defining the 5-mer nucleotide sequence.

When `numSeqMutationsOnly` is `TRUE`, a named numeric vector of length 1024 counting the number of observed mutations in sequences containing each 5-mer.

References

1. Yaari G, et al. Models of somatic hypermutation targeting and substitution based on synonymous mutations from high-throughput immunoglobulin sequencing data. Front Immunol. 2013 4(November):358.

See Also

MutabilityModel, extendMutabilityMatrix, createSubstitutionMatrix, createTargetingMatrix, createTargetingModel, minNumSeqMutationsTune

Examples

```
# Subset example data to 50 sequences of one isotype and sample as a demo
data(ExampleDb, package="alakazam")
db <- subset(ExampleDb, c_call == "IGHA" & sample_id == "-1h")[1:50,]

# Create model using only silent mutations
sub_model <- createSubstitutionMatrix(db, sequenceColumn="sequence_alignment",
                                      germlineColumn="germline_alignment_d_mask",
                                      vCallColumn="v_call",model="s")
mut_model <- createMutabilityMatrix(db, sub_model, model="s",
                                    sequenceColumn="sequence_alignment",
                                    germlineColumn="germline_alignment_d_mask",
                                    vCallColumn="v_call",
                                    minNumSeqMutations=200,
                                    numSeqMutationsOnly=FALSE)

# View top 5 mutability estimates
head(sort(mut_model, decreasing=TRUE), 5)

# View the number of S mutations used for estimating mutabilities
mut_model@numMutS

# Count the number of mutations in sequences containing each 5-mer
mut_count <- createMutabilityMatrix(db, sub_model, model="s",
                                    sequenceColumn="sequence_alignment",
                                    germlineColumn="germline_alignment_d_mask",
                                    vCallColumn="v_call",
                                    numSeqMutationsOnly=TRUE)
```

---

createMutationDefinition
Creates a MutationDefinition

Description

`createMutationDefinition` creates a `MutationDefinition`.

Usage

```
createMutationDefinition(name, classes, description = "", citation = "")
```

Arguments

`name`	name of the mutation definition.
`classes`	named character vectors with single-letter amino acid codes as names and amino acid classes as values, with `NA` assigned to set of characters `c("X", "*", "-", ".")`. Replacement (R) is be defined as a change in amino acid class and silent (S) as no change in class.
`description`	description of the mutation definition and its source data.
`citation`	publication source.

Value

A `MutationDefinition` object.

See Also

See MutationDefinition for the return object.

Examples

```
# Define hydropathy classes
suppressPackageStartupMessages(library(alakazam))
hydropathy <- list(hydrophobic=c("A", "I", "L", "M", "F", "W", "V"),
                   hydrophilic=c("R", "N", "D", "C", "Q", "E", "K"),
                   neutral=c("G", "H", "P", "S", "T", "Y"))
chars <- unlist(hydropathy, use.names=FALSE)
classes <- setNames(translateStrings(chars, hydropathy), chars)

# Create hydropathy mutation definition
md <- createMutationDefinition("Hydropathy", classes)
```

---

createRegionDefinition

Creates a RegionDefinition

Description

`createRegionDefinition` creates a `RegionDefinition`.

Usage

```
createRegionDefinition(
  name = "",
  boundaries = factor(),
  description = "",
  citation = ""
)
```

Arguments

`name`	name of the region definition.
`boundaries`	`factor` defining the region boundaries of the sequence. The levels and values of `boundaries` determine the number of regions (e.g. CDR and FWR).
`description`	description of the region definition and its source data.
`citation`	publication source.

Value

A `RegionDefinition` object.

See Also

See RegionDefinition for the return object.

Examples

```
# Creates an empty RegionDefinition object
createRegionDefinition()
```

`createSubstitutionMatrix` *Builds a substitution model*

Description

`createSubstitutionMatrix` builds a 5-mer nucleotide substitution model by counting the number of substitution mutations occurring in the center position for all 5-mer motifs.

Usage

```
createSubstitutionMatrix(
  db,
  model = c("s", "rs"),
  sequenceColumn = "sequence_alignment",
  germlineColumn = "germline_alignment_d_mask",
  vCallColumn = "v_call",
  multipleMutation = c("independent", "ignore"),
  returnModel = c("5mer", "1mer", "1mer_raw"),
  minNumMutations = 50,
  numMutationsOnly = FALSE
)
```

Arguments

`db`	data.frame containing sequence data.
`model`	type of model to create. The default model, "s", builds a model by counting only silent mutations. `model="s"` should be used for data that includes functional sequences. Setting `model="rs"` creates a model by counting both replacement and silent mutations and may be used on fully non-functional sequence data sets.
`sequenceColumn`	name of the column containing IMGT-gapped sample sequences.
`germlineColumn`	name of the column containing IMGT-gapped germline sequences.
`vCallColumn`	name of the column containing the V-segment allele call.
`multipleMutation`	string specifying how to handle multiple mutations occurring within the same 5-mer. If `"independent"` then multiple mutations within the same 5-mer are counted independently. If `"ignore"` then 5-mers with multiple mutations are excluded from the total mutation tally.

`returnModel`	string specifying what type of model to return; one of `c("5mer", "1mer", "1mer_raw")`. If `"5mer"` (the default) then a 5-mer nucleotide context model is returned. If `"1mer"` or `"1mer_raw"` then a single nucleotide substitution matrix (no context) is returned; where `"1mer_raw"` is the unnormalized version of the `"1mer"` model. Note, neither 1-mer model may be used as input to createMutabilityMatrix.
`minNumMutations`	minimum number of mutations required to compute the 5-mer substitution rates. If the number of mutations for a 5-mer is below this threshold, its substitution rates will be estimated from neighboring 5-mers. Default is 50. Not required if `numMutationsOnly=TRUE`.
`numMutationsOnly`	when `TRUE`, return counting information on the number of mutations for each 5-mer, instead of building a substitution matrix. This option can be used for parameter tuning for `minNumMutations` during preliminary analysis. Default is `FALSE`. Only applies when `returnModel` is set to `"5mer"`. The `data.frame` returned when this argument is `TRUE` can serve as the input for minNumMutationsTune.

Details

Caution: The targeting model functions do NOT support ambiguous characters in their inputs. You MUST make sure that your input and germline sequences do NOT contain ambiguous characters (especially if they are clonal consensuses returned from `collapseClones`).

Value

For `returnModel = "5mer"`:

When `numMutationsOnly` is `FALSE`, a 4x1024 matrix of column normalized substitution rates for each 5-mer motif with row names defining the center nucleotide, one of `c("A", "C", "G", "T")`, and column names defining the 5-mer nucleotide sequence.

When `numMutationsOnly` is `TRUE`, a 1024x4 data frame with each row providing information on counting the number of mutations for a 5-mer. Columns are named `fivemer.total`, `fivemer.every`, `inner3.total`, and `inner3.every`, corresponding to, respectively, the total number of mutations when counted as a 5-mer, whether there is mutation to every other base when counted as a 5-mer, the total number of mutations when counted as an inner 3-mer, and whether there is mutation to every other base when counted as an inner 3-mer.

For `returnModel = "1mer"` or `"1mer_raw"`: a 4x4 normalized or un-normalized 1-mer substitution matrix respectively.

References

1. Yaari G, et al. Models of somatic hypermutation targeting and substitution based on synonymous mutations from high-throughput immunoglobulin sequencing data. Front Immunol. 2013 4(November):358.

See Also

extendSubstitutionMatrix, createMutabilityMatrix, createTargetingMatrix, createTargetingModel, minNumMutationsTune.

Examples

```
# Subset example data to one isotype and sample as a demo
data(ExampleDb, package="alakazam")
db <- subset(ExampleDb, c_call == "IGHA" & sample_id == "-1h")[1:25,]

# Count the number of mutations per 5-mer
subCount <- createSubstitutionMatrix(db, sequenceColumn="sequence_alignment",
                                     germlineColumn="germline_alignment_d_mask",
                                     vCallColumn="v_call",
                                     model="s", multipleMutation="independent",
                                     returnModel="5mer", numMutationsOnly=TRUE)

# Create model using only silent mutations
sub <- createSubstitutionMatrix(db, sequenceColumn="sequence_alignment",
                                germlineColumn="germline_alignment_d_mask",
                                vCallColumn="v_call",
                                model="s", multipleMutation="independent",
                                returnModel="5mer", numMutationsOnly=FALSE,
                                minNumMutations=20)
```

`createTargetingMatrix` *Calculates a targeting rate matrix*

Description

`createTargetingMatrix` calculates the targeting model matrix as the combined probability of mutability and substitution.

Usage

```
createTargetingMatrix(substitutionModel, mutabilityModel)
```

Arguments

`substitutionModel`	matrix of 5-mers substitution rates built by createSubstitutionMatrix or extendSubstitutionMatrix.
`mutabilityModel`	vector of 5-mers mutability rates built by createMutabilityMatrix or extendMutabilityMatrix.

Details

Targeting rates are calculated by multiplying the normalized mutability rate by the normalized substitution rates for each individual 5-mer.

Value

A `TargetingMatrix` with the same dimensions as the input `substitutionModel` containing normalized targeting probabilities for each 5-mer motif with row names defining the center nucleotide and column names defining the 5-mer nucleotide sequence.

If the input `mutabilityModel` is of class `MutabilityModel`, then the output `TargetingMatrix` will carry over the input `numMutS` and `numMutR` slots.

References

1. Yaari G, et al. Models of somatic hypermutation targeting and substitution based on synonymous mutations from high-throughput immunoglobulin sequencing data. Front Immunol. 2013 4(November):358.

See Also

createSubstitutionMatrix, extendSubstitutionMatrix, createMutabilityMatrix, extendMutabilityMatrix, TargetingMatrix, createTargetingModel

Examples

```
# Subset example data to 50 sequences, of one isotype and sample as a demo
data(ExampleDb, package="alakazam")
db <- subset(ExampleDb, c_call == "IGHA" & sample_id == "-1h")[1:50,]

# Create 4x1024 models using only silent mutations
sub_model <- createSubstitutionMatrix(db, model="s", sequenceColumn="sequence_alignment",
                                      germlineColumn="germline_alignment_d_mask",
                                      vCallColumn="v_call")
mut_model <- createMutabilityMatrix(db, sub_model, model="s",
                                    sequenceColumn="sequence_alignment",
                                    germlineColumn="germline_alignment_d_mask",
                                    vCallColumn="v_call")

# Extend substitution and mutability to including Ns (5x3125 model)
sub_model <- extendSubstitutionMatrix(sub_model)
mut_model <- extendMutabilityMatrix(mut_model)

# Create targeting model from substitution and mutability
tar_model <- createTargetingMatrix(sub_model, mut_model)
```

`createTargetingModel` *Creates a TargetingModel*

Description

`createTargetingModel` creates a 5-mer `TargetingModel`.

Usage

```
createTargetingModel(
  db,
  model = c("s", "rs"),
  sequenceColumn = "sequence_alignment",
  germlineColumn = "germline_alignment_d_mask",
  vCallColumn = "v_call",
  multipleMutation = c("independent", "ignore"),
  minNumMutations = 50,
  minNumSeqMutations = 500,
  modelName = "",
  modelDescription = "",
  modelSpecies = "",
  modelCitation = "",
  modelDate = NULL
)
```

Arguments

`db`	data.frame containing sequence data.
`model`	type of model to create. The default model, "s", builds a model by counting only silent mutations. `model="s"` should be used for data that includes functional sequences. Setting `model="rs"` creates a model by counting both replacement and silent mutations and may be used on fully non-functional sequence data sets.
`sequenceColumn`	name of the column containing IMGT-gapped sample sequences.
`germlineColumn`	name of the column containing IMGT-gapped germline sequences.
`vCallColumn`	name of the column containing the V-segment allele calls.
`multipleMutation`	string specifying how to handle multiple mutations occurring within the same 5-mer. If `"independent"` then multiple mutations within the same 5-mer are counted independently. If `"ignore"` then 5-mers with multiple mutations are excluded from the total mutation tally.
`minNumMutations`	minimum number of mutations required to compute the 5-mer substitution rates. If the number of mutations for a 5-mer is below this threshold, its substitution rates will be estimated from neighboring 5-mers. Default is 50.

`minNumSeqMutations`	minimum number of mutations in sequences containing each 5-mer to compute the mutability rates. If the number is smaller than this threshold, the mutability for the 5-mer will be inferred. Default is 500.
`modelName`	name of the model.
`modelDescription`	description of the model and its source data.
`modelSpecies`	genus and species of the source sequencing data.
`modelCitation`	publication source.
`modelDate`	date the model was built. If `NULL` the current date will be used.

Details

Caution: The targeting model functions do NOT support ambiguous characters in their inputs. You MUST make sure that your input and germline sequences do NOT contain ambiguous characters (especially if they are clonal consensuses returned from `collapseClones`).

Value

A TargetingModel object.

References

1. Yaari G, et al. Models of somatic hypermutation targeting and substitution based on synonymous mutations from high-throughput immunoglobulin sequencing data. Front Immunol. 2013 4(November):358.

See Also

See TargetingModel for the return object. See plotMutability plotting a mutability model. See createSubstitutionMatrix, extendSubstitutionMatrix, createMutabilityMatrix, extendMutabilityMatrix and createTargetingMatrix for component steps in building a model.

Examples

```
# Subset example data to one isotype and sample as a demo
data(ExampleDb, package="alakazam")
db <- subset(ExampleDb, c_call == "IGHA" & sample_id == "-1h")[1:80,]

# Create model using only silent mutations and ignore multiple mutations
model <- createTargetingModel(db, model="s", sequenceColumn="sequence_alignment",
                              germlineColumn="germline_alignment_d_mask",
                              vCallColumn="v_call", multipleMutation="ignore")

# View top 5 mutability estimates
head(sort(model@mutability, decreasing=TRUE), 5)

# View number of silent mutations used for estimating mutability
model@numMutS
```

`DensityThreshold-class` *Output of the* `dens` *method of findThreshold*

Description

`DensityThreshold` contains output from the `dens` method findThreshold.

Usage

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

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

Arguments

`x`	DensityThreshold object
`y`	ignored.
`...`	arguments to pass to plotDensityThreshold.

Slots

`x` input distance vector with NA or infinite values removed.

`bandwidth` bandwidth value fit during density estimation.

`xdens` x-axis (distance value) vector for smoothed density estimate.

`ydens` y-axis (density) vector for smoothed density estimate.

`threshold` distance threshold that separates two modes of the input distribution.

See Also

findThreshold

`distToNearest` *Distance to nearest neighbor*

Description

Calculate the non-zero distance from each sequence to its nearest neighbor within partitions based on shared V gene, J gene, and junction length.

Usage

```
distToNearest(
  db,
  sequenceColumn = "junction",
  vCallColumn = "v_call",
  jCallColumn = "j_call",
 model = c("ham", "aa", "hh_s1f", "hh_s5f", "mk_rs1nf", "mk_rs5nf", "m1n_compat",
    "hs1f_compat"),
  normalize = c("len", "none"),
  symmetry = c("avg", "min"),
  first = TRUE,
  VJthenLen = TRUE,
  nproc = 1,
  fields = NULL,
  cross = NULL,
  mst = FALSE,
  subsample = NULL,
  progress = FALSE,
  cellIdColumn = NULL,
  locusColumn = "locus",
  locusValues = c("IGH"),
  onlyHeavy = TRUE,
  keepVJLgroup = TRUE
)
```

Arguments

`db`	data.frame containing sequence data.
`sequenceColumn`	name of the column containing the junction for grouping and for calculating nearest neighbor distances. Note that while both heavy/long and light/short chain junctions may be used for V-J-length grouping, only the heavy/long chain (IGH, TRB, TRD) junction is used to calculate distances.
`vCallColumn`	name of the column containing the V-segment allele calls.
`jCallColumn`	name of the column containing the J-segment allele calls.
`model`	underlying SHM model, which must be one of `c("ham", "aa", "hh_s1f", "hh_s5f", "mk_rs1nf", "hs1f_compat", "m1n_compat")`. See Details for further information.
`normalize`	method of normalization. The default is `"len"`, which divides the distance by the length of the sequence group. If `"none"` then no normalization if performed.
`symmetry`	if model is hs5f, distance between seq1 and seq2 is either the average (avg) of seq1->seq2 and seq2->seq1 or the minimum (min).
`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.
`VJthenLen`	logical value specifying whether to perform partitioning as a 2-stage process. If `TRUE`, partitions are made first based on V and J gene, and then further split

	based on junction lengths corresponding to `sequenceColumn`. If `FALSE`, perform partition as a 1-stage process during which V gene, J gene, and junction length are used to create partitions simultaneously. Defaults to `TRUE`.
`nproc`	number of cores to distribute the function over.
`fields`	additional fields to use for grouping.
`cross`	character vector of column names to use for grouping to calculate distances across groups. Meaning the columns that define self versus others.
`mst`	if `TRUE`, return comma-separated branch lengths from minimum spanning tree.
`subsample`	number of sequences to subsample for speeding up pairwise-distance-matrix calculation. Subsampling is performed without replacement in each V-J-length group of heavy chain sequences. If `subsample` is larger than the unique number of heavy chain sequences in each VJL group, then the subsampling process is ignored for that group. For each heavy chain sequence in `db`, the reported `dist_nearest` is the distance to the closest heavy chain sequence in the subsampled set for the V-J-length group. If `NULL` no subsampling is performed.
`progress`	if `TRUE` print a progress bar.
`cellIdColumn`	name of the character column containing cell identifiers or barcodes. If specified, grouping will be performed in single-cell mode with the behavior governed by the `locusColumn` and `onlyHeavy` arguments. If set to `NULL` then the bulk sequencing data is assumed.
`locusColumn`	name of the column containing locus information. Valid loci values are "IGH", "IGI", "IGK", "IGL", "TRA", "TRB", "TRD", and "TRG".
`locusValues`	Loci values to focus the analysis on.
`onlyHeavy`	This is deprecated. Only IGH (BCR) or TRB/TRD (TCR) sequences will be used for grouping. Only applicable to single-cell data. Ignored if `cellIdColumn=NULL`. See groupGenes for further details.
`keepVJLgroup`	logical value specifying whether to keep in the output the the column column indicating grouping based on V-J-length combinations. Only applicable for 1-stage partitioning (i.e. `VJthenLen=FALSE`). Also see groupGenes.

Details

The distance to nearest neighbor can be used to estimate a threshold for assigning Ig sequences to clonal groups. A histogram of the resulting vector is often bimodal, with the ideal threshold being a value that separates the two modes.

Refer to the details section for a more thorough description of the implementation.

There are two modes of operation for `distToNearest`: single-cell (all sequences are single-cell data), non-single-cell (all sequences are bulk sequencing data). Mixed data, where both single-cell and non-single-cell sequences are present in the data, is considered a case under the single-single cell mode .

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

Under single-cell mode, only heavy/long chain (IGH, TRB, TRD) sequences will be used for calculating nearest neighbor distances regardless of `locusValue` values in the `locusColumn` field (if present). Under non-single-cell mode, all input sequences with `locusValue` value(s) in the `locusColumn` field will be used for calculating nearest neighbor distances.

Values in the `locusColumn` 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.

For single-cell mode, the input format is the same as that for groupGenes. Namely, each row represents a sequence/chain. Sequences/chains from the same cell are linked by a cell ID in the `cellIdColumn` field. Grouping will be done by using IGH (BCR) or TRB/TRD (TCR) sequences only. The argument that allowed to include light chains, `onlyHeavy`, is deprecated.

Note, `distToNearest` required that each cell (each unique value in `cellIdColumn`) correspond to only a single `IGH` (BCR) or `TRB/TRD` (TCR) sequence.

The following distance measures are accepted by the `model` parameter.

- `"ham"`: Single nucleotide Hamming distance matrix from getDNAMatrix with gaps assigned zero distance.
- `"aa"`: Single amino acid Hamming distance matrix from getAAMatrix.
- `"hh_s1f"`: Human single nucleotide distance matrix derived from HH_S1F with calcTargetingDistance.
- `"hh_s5f"`: Human 5-mer nucleotide context distance matrix derived from HH_S5F with calcTargetingDistance.
- `"mk_rs1nf"`: Mouse single nucleotide distance matrix derived from MK_RS1NF with calcTargetingDistance.
- `"mk_rs5nf"`: Mouse 5-mer nucleotide context distance matrix derived from MK_RS5NF with calcTargetingDistance.
- `"hs1f_compat"`: Backwards compatible human single nucleotide distance matrix used in SHazaM v0.1.4 and Change-O v0.3.3.
- `"m1n_compat"`: Backwards compatibility mouse single nucleotide distance matrix used in SHazaM v0.1.4 and Change-O v0.3.3.

Note on `NA`s: if, for a given combination of V gene, J gene, and junction length, there is only 1 heavy chain sequence (as defined by `sequenceColumn`), `NA` is returned instead of a distance (since it has no heavy/long chain neighbor). If for a given combination there are multiple heavy/long chain sequences but only 1 unique one, (in which case every heavy/long chain sequence in this group is the de facto nearest neighbor to each other, thus giving rise to distances of 0), `NA`s are returned instead of zero-distances.

Note on `subsample`: Subsampling is performed independently in each V-J-length group for heavy/long chain sequences. If `subsample` is larger than number of heavy/long chain sequences in the group, it is ignored. In other words, subsampling is performed only on groups in which the number of heavy/long chain sequences is equal to or greater than `subsample`. `dist_nearest` has values calculated using all heavy chain sequences in the group for groups with fewer than `subsample` heavy/long chain sequences, and values calculated using a subset of heavy/long chain sequences for the larger groups. To select a value of `subsample`, it can be useful to explore the group sizes in `db` (and the number of heavy/long chain sequences in those groups).

Value

Returns a modified db data.frame with nearest neighbor distances between heavy chain sequences in the `dist_nearest` column if `cross=NULL`. If `cross` was specified, distances will be added as the `cross_dist_nearest` column.

Note that distances between light/short (IGK, IGL, TRA, TRG) chain sequences are not calculated, even if light/short chains were used for V-J-length grouping via `onlyHeavy=FALSE`. Light/short chain sequences, if any, will have `NA` in the `dist_nearest` output column.

Note that the output `vCallColumn` and `jCallColumn` columns will be converted to type `character` if they were type `factor` in the input db.

References

1. Smith DS, et al. Di- and trinucleotide target preferences of somatic mutagenesis in normal and autoreactive B cells. J Immunol. 1996 156:2642-52.
2. Glanville J, Kuo TC, von Budingen H-C, et al. Naive antibody gene-segment frequencies are heritable and unaltered by chronic lymphocyte ablation. Proc Natl Acad Sci USA. 2011 108(50):20066-71.
3. Yaari G, et al. Models of somatic hypermutation targeting and substitution based on synonymous mutations from high-throughput immunoglobulin sequencing data. Front Immunol. 2013 4:358.

See Also

See calcTargetingDistance for generating nucleotide distance matrices from a TargetingModel object. See HH_S5F, HH_S1F, MK_RS1NF, getDNAMatrix, and getAAMatrix for individual model details. getLocus to get locus values based on allele calls.

Examples

```
# Subset example data to one sample as a demo
data(ExampleDb, package="alakazam")
db <- subset(ExampleDb, sample_id == "-1h")

# Use genotyped V assignments, Hamming distance, and normalize by junction length
# First partition based on V and J assignments, then by junction length
# Take into consideration ambiguous V and J annotations
dist <- distToNearest(db, sequenceColumn="junction",
                      vCallColumn="v_call_genotyped", jCallColumn="j_call",
                      model="ham", first=FALSE, VJthenLen=TRUE, normalize="len")

# Plot histogram of non-NA distances
p1 <- ggplot(data=subset(dist, !is.na(dist_nearest))) +
      theme_bw() +
      ggtitle("Distance to nearest: Hamming") +
      xlab("distance") +
      geom_histogram(aes(x=dist_nearest), binwidth=0.025,
                     fill="steelblue", color="white")
plot(p1)
```

editBaseline *Edit the Baseline object*

Description

editBaseline edits a field in a Baseline object.

Usage

```
editBaseline(baseline, field, value)
```

Arguments

baseline	Baseline object to be edited.
field	name of the field in the Baseline object to be edited.
value	value to set the field.

Value

A Baseline object with the field of choice updated.

See Also

See Baseline for the input and return object.

Examples

```
# Subset example data as a demo
data(ExampleDb, package="alakazam")
db <- subset(ExampleDb, c_call == "IGHG" & sample_id == "+7d")
set.seed(112)
db <- dplyr::slice_sample(db, n=100)

# Make Baseline object
baseline <- calcBaseline(db,
                         sequenceColumn="sequence_alignment",
                         germlineColumn="germline_alignment_d_mask",
                         testStatistic="focused",
                         regionDefinition=IMGT_V,
                         targetingModel=HH_S5F,
                         nproc=1)

# Edit the field "description"
baseline <- editBaseline(baseline, field="description",
                         value="+7d IGHG")
```

`expectedMutations` *Calculate expected mutation frequencies*

Description

`expectedMutations` calculates the expected mutation frequencies for each sequence in the input `data.frame`.

Usage

```
expectedMutations(
  db,
  sequenceColumn = "sequence_alignment",
  germlineColumn = "germline_alignment",
  targetingModel = HH_S5F,
  regionDefinition = NULL,
  mutationDefinition = NULL,
  nproc = 1,
  cloneColumn = "clone_id",
  juncLengthColumn = "junction_length"
)
```

Arguments

`db`	`data.frame` containing sequence data.
`sequenceColumn`	`character` name of the column containing input sequences.
`germlineColumn`	`character` name of the column containing the germline or reference sequence.
`targetingModel`	TargetingModel object. Default is HH_S5F.
`regionDefinition`	RegionDefinition object defining the regions and boundaries of the Ig sequences. To use regions definitions, sequences in `sequenceColumn` and `germlineColumn` must be aligned, following the IMGT schema.
`mutationDefinition`	MutationDefinition object defining replacement and silent mutation criteria. If `NULL` then replacement and silent are determined by exact amino acid identity.
`nproc`	`numeric` number of cores to distribute the operation over. If the cluster has already been set the call function with `nproc` = 0 to not reset or reinitialize. Default is `nproc` = 1.
`cloneColumn`	clone id column name in db
`juncLengthColumn`	junction length column name in db

Details

Only the part of the sequences defined in `regionDefinition` are analyzed. For example, when using the IMGT_V definition, mutations in positions beyond 312 will be ignored.

Value

A modified db `data.frame` with expected mutation frequencies for each region defined in `regionDefinition`.

The columns names are dynamically created based on the regions in `regionDefinition`. For example, when using the IMGT_V definition, which defines positions for CDR and FWR, the following columns are added:

- `mu_expected_cdr_r`: number of replacement mutations in CDR1 and CDR2 of the V-segment.
- `mu_expected_cdr_s`: number of silent mutations in CDR1 and CDR2 of the V-segment.
- `mu_expected_fwr_r`: number of replacement mutations in FWR1, FWR2 and FWR3 of the V-segment.
- `mu_expected_fwr_s`: number of silent mutations in FWR1, FWR2 and FWR3 of the V-segment.

See Also

calcExpectedMutations is called by this function to calculate the expected mutation frequencies. See observedMutations for getting observed mutation counts. See IMGT_SCHEMES for a set of predefined RegionDefinition objects.

Examples

```
# Subset example data
data(ExampleDb, package="alakazam")
db <- subset(ExampleDb, c_call %in% c("IGHA", "IGHG") & sample_id == "+7d")
set.seed(112)
db <- dplyr::slice_sample(db, n=100)
# Calculate expected mutations over V region
db_exp <- expectedMutations(db,
                            sequenceColumn="sequence_alignment",
                            germlineColumn="germline_alignment_d_mask",
                            regionDefinition=IMGT_V,
                            nproc=1)

# Calculate hydropathy expected mutations over V region
db_exp <- expectedMutations(db,
                           sequenceColumn="sequence_alignment",
                           germlineColumn="germline_alignment_d_mask",
                           regionDefinition=IMGT_V,
                           mutationDefinition=HYDROPATHY_MUTATIONS,
                           nproc=1)
```

`extendMutabilityMatrix` *Extends a mutability model to include Ns.*

Description

`extendMutabilityMatrix` extends a 5-mer nucleotide mutability model with 5-mers that include Ns by averaging over all corresponding 5-mers without Ns.

Usage

```
extendMutabilityMatrix(mutabilityModel)
```

Arguments

`mutabilityModel`	vector of 5-mer mutability rates built by createMutabilityMatrix.

Value

A `MutabilityModel` containing a 3125 vector of normalized mutability rates for each 5-mer motif with names defining the 5-mer nucleotide sequence. Note that "normalized" means that the mutability rates for the 1024 5-mers that contain no "N" at any position sums up to 1 (as opposed to the entire vector summing up to 1).

If the input `mutabilityModel` is of class `MutabilityModel`, then the output `MutabilityModel` will carry over the input `numMutS` and `numMutR` slots.

See Also

createMutabilityMatrix, extendSubstitutionMatrix, MutabilityModel

Examples

```
# Subset example data to one isotype and sample as a demo
data(ExampleDb, package="alakazam")
db <- subset(ExampleDb, c_call == "IGHA" & sample_id == "-1h")
set.seed(112)
db <- dplyr::slice_sample(db, n=75)

# Create model using only silent mutations and ignore multiple mutations
sub_model <- createSubstitutionMatrix(db, model="s", sequenceColumn="sequence_alignment",
                                      germlineColumn="germline_alignment_d_mask",
                                      vCallColumn="v_call")
mut_model <- createMutabilityMatrix(db, sub_model, model="s",
                                    sequenceColumn="sequence_alignment",
                                    germlineColumn="germline_alignment_d_mask",
                                    vCallColumn="v_call")
ext_model <- extendMutabilityMatrix(mut_model)
```

extendSubstitutionMatrix

Extends a substitution model to include Ns.

Description

`extendSubstitutionMatrix` extends a 5-mer nucleotide substitution model with 5-mers that include Ns by averaging over all corresponding 5-mers without Ns.

Usage

```
extendSubstitutionMatrix(substitutionModel)
```

Arguments

`substitutionModel`	matrix of 5-mers substitution counts built by createSubstitutionMatrix.

Value

A 5x3125 matrix of normalized substitution rate for each 5-mer motif with rows names defining the center nucleotide, one of `c("A", "C", "G", "T", "N")`, and column names defining the 5-mer nucleotide sequence.

See Also

createSubstitutionMatrix, extendMutabilityMatrix

Examples

```
# Subset example data to one isotype and sample as a demo
data(ExampleDb, package="alakazam")
db <- subset(ExampleDb, c_call == "IGHA" & sample_id == "-1h")

# Create model using only silent mutations
sub_model <- createSubstitutionMatrix(db, sequenceColumn="sequence_alignment",
                                      germlineColumn="germline_alignment_d_mask",
                                      vCallColumn="v_call",model="s")
ext_model <- extendSubstitutionMatrix(sub_model)
```

findThreshold *Find distance threshold*

Description

findThreshold automatically determines an optimal threshold for clonal assignment of Ig sequences using a vector of nearest neighbor distances. It provides two alternative methods using either a Gamma/Gaussian Mixture Model fit (method="gmm") or kernel density fit (method="density").

Usage

```
findThreshold(
  distances,
  method = c("density", "gmm"),
  edge = 0.9,
  cross = NULL,
  subsample = NULL,
  model = c("gamma-gamma", "gamma-norm", "norm-gamma", "norm-norm"),
  cutoff = c("optimal", "intersect", "user"),
  sen = NULL,
  spc = NULL,
  progress = FALSE
)
```

Arguments

distances	numeric vector containing nearest neighbor distances.
method	string defining the method to use for determining the optimal threshold. One of "gmm" or "density". See Details for methodological descriptions.
edge	upper range as a fraction of the data density to rule initialization of Gaussian fit parameters. Default value is 90 Applies only when method="density". .
cross	supplementary nearest neighbor distance vector output from distToNearest for initialization of the Gaussian fit parameters. Applies only when method="gmm".
subsample	maximum number of distances to subsample to before threshold detection.
model	allows the user to choose among four possible combinations of fitting curves: "norm-norm", "norm-gamma", "gamma-norm", and "gamma-gamma". Applies only when method="gmm".
cutoff	method to use for threshold selection: the optimal threshold "opt", the intersection point of the two fitted curves "intersect", or a value defined by user for one of the sensitivity or specificity "user". Applies only when method="gmm".
sen	sensitivity required. Applies only when method="gmm" and cutoff="user".
spc	specificity required. Applies only when method="gmm" and cutoff="user".
progress	if TRUE print a progress bar.

Details

- "gmm": Performs a maximum-likelihood fitting procedure, for learning the parameters of two mixture univariate, either Gamma or Gaussian, distributions which fit the bimodal distribution entries. Retrieving the fit parameters, it then calculates the optimum threshold method="optimal", where the average of the sensitivity plus specificity reaches its maximum. In addition, the findThreshold function is also able to calculate the intersection point (method="intersect") of the two fitted curves and allows the user to invoke its value as the cut-off point, instead of optimal point.
- "density": Fits a binned approximation to the ordinary kernel density estimate to the nearest neighbor distances after determining the optimal bandwidth for the density estimate via least-squares cross-validation of the 4th derivative of the kernel density estimator. The optimal threshold is set as the minimum value in the valley in the density estimate between the two modes of the distribution.

Value

- "gmm" method: Returns a GmmThreshold object including the threshold and the function fit parameters, i.e. mixing weight, mean, and standard deviation of a Normal distribution, or mixing weight, shape and scale of a Gamma distribution.
- "density" method: Returns a DensityThreshold object including the optimum threshold and the density fit parameters.

Note

Visually inspecting the resulting distribution fits is strongly recommended when using either fitting method. Empirical observations imply that the bimodality of the distance-to-nearest distribution is detectable for a minimum of 1,000 distances. Larger numbers of distances will improve the fitting procedure, although this can come at the expense of higher computational demands.

See Also

See distToNearest for generating the nearest neighbor distance vectors. See plotGmmThreshold and plotDensityThreshold for plotting output.

Examples

```
# Subset example data to 50 sequences, one sample and isotype as a demo
data(ExampleDb, package="alakazam")
db <- subset(ExampleDb, sample_id == "-1h" & c_call=="IGHG")[1:50,]

# Use nucleotide Hamming distance and normalize by junction length
db <- distToNearest(db, sequenceColumn="junction", vCallColumn="v_call",
                    jCallColumn="j_call", model="ham", normalize="len", nproc=1)

# Find threshold using the "gmm" method with user defined specificity
output <- findThreshold(db$dist_nearest, method="gmm", model="gamma-gamma",
                        cutoff="user", spc=0.99)
plot(output, binwidth=0.02, title=paste0(output@model, "   loglk=", output@loglk))
print(output)
```

`GmmThreshold-class` *Output of the* `gmm` *method of findThreshold*

Description

`GmmThreshold` contains output from the `gmm` method findThreshold. It includes parameters of two Gaussian fits and threshold cut.

Usage

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

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

Arguments

`x`	GmmThreshold object
`y`	ignored.
`...`	arguments to pass to plotGmmThreshold.

Slots

`x` input distance vector with NA or infinite values removed.

`model` first-second fit functions.

`cutoff` type of threshold cut.

`a1` mixing weight of the first curve.

`b1` second parameter of the first curve. Either the mean of a Normal distribution or shape of a Gamma distribution.

`c1` third parameter of the first curve. Either the standard deviation of a Normal distribution or scale of a Gamma distribution.

`a2` mixing weight of the second curve.

`b2` second parameter of the second curve. Either the mean of a Normal distribution or shape of a Gamma distribution.

`c2` third parameter of the second curve. Either the standard deviation of a Normal distribution or scale of a Gamma distribution.

`loglk` log-likelihood of the fit.

`threshold` threshold.

`sensitivity` sensitivity.

`specificity` specificity.

`pvalue` p-value from Hartigans' dip statistic (HDS) test. Values less than 0.05 indicate significant bimodality.

See Also

findThreshold

`groupBaseline` *Group BASELINe PDFs*

Description

`groupBaseline` convolves groups of BASELINe posterior probability density functions (PDFs) to get combined PDFs for each group.

Usage

```
groupBaseline(baseline, groupBy, nproc = 1)
```

Arguments

`baseline`	`Baseline` object containing the `db` and the BASELINe posterior probability density functions (PDF) for each of the sequences, as returned by calcBaseline.
`groupBy`	The columns in the `db` slot of the `Baseline` object by which to group the sequence PDFs.
`nproc`	number of cores to distribute the operation over. If `nproc` = 0 then the `cluster` has already been set and will not be reset.

Details

While the selection strengths predicted by BASELINe perform well on average, the estimates for individual sequences can be highly variable, especially when the number of mutations is small.

To overcome this, PDFs from sequences grouped by biological or experimental relevance, are convolved to from a single PDF for the selection strength. For example, sequences from each sample may be combined together, allowing you to compare selection across samples. This is accomplished through a fast numerical convolution technique.

Value

A Baseline object, containing the modified `db` and the BASELINe posterior probability density functions (PDF) for each of the groups.

References

1. Yaari G, et al. Quantifying selection in high-throughput immunoglobulin sequencing data sets. Nucleic Acids Res. 2012 40(17):e134. (Corrections at http://selection.med.yale.edu/baseline/correction/)

See Also

To generate the Baseline object see calcBaseline. To calculate BASELINe statistics, such as the mean selection strength and the 95% confidence interval, see summarizeBaseline.

Examples

```
## Not run:
# Subset example data from alakazam as a demo
data(ExampleDb, package="alakazam")
db <- subset(ExampleDb, c_call %in% c("IGHM", "IGHG"))
set.seed(112)
db <- dplyr::slice_sample(db, n=200)

# Collapse clones
db <- collapseClones(db, cloneColumn="clone_id",
                     sequenceColumn="sequence_alignment",
                     germlineColumn="germline_alignment_d_mask",
                     method="thresholdedFreq", minimumFrequency=0.6,
                     includeAmbiguous=FALSE, breakTiesStochastic=FALSE)

# Calculate BASELINe
baseline <- calcBaseline(db,
                         sequenceColumn="clonal_sequence",
                         germlineColumn="clonal_germline",
                         testStatistic="focused",
                         regionDefinition=IMGT_V,
                         targetingModel=HH_S5F,
                         nproc=1)

# Group PDFs by sample
grouped1 <- groupBaseline(baseline, groupBy="sample_id")
sample_colors <- c("-1h"="steelblue", "+7d"="firebrick")
plotBaselineDensity(grouped1, idColumn="sample_id", colorValues=sample_colors,
                    sigmaLimits=c(-1, 1))

# Group PDFs by both sample (between variable) and isotype (within variable)
grouped2 <- groupBaseline(baseline, groupBy=c("sample_id", "c_call"))
isotype_colors <- c("IGHM"="darkorchid", "IGHD"="firebrick",
                    "IGHG"="seagreen", "IGHA"="steelblue")
plotBaselineDensity(grouped2, idColumn="sample_id", groupColumn="c_call",
                    colorElement="group", colorValues=isotype_colors,
                    sigmaLimits=c(-1, 1))
# Collapse previous isotype (within variable) grouped PDFs into sample PDFs
grouped3 <- groupBaseline(grouped2, groupBy="sample_id")
sample_colors <- c("-1h"="steelblue", "+7d"="firebrick")
plotBaselineDensity(grouped3, idColumn="sample_id", colorValues=sample_colors,
                    sigmaLimits=c(-1, 1))

## End(Not run)
```

`HH_S1F` *Human heavy chain, silent, 1-mer, functional substitution model.*

Description

1-mer substitution model of somatic hypermutation based on analysis of silent mutations in functional heavy chain Ig sequences from Homo sapiens.

Usage

```
HH_S1F
```

Format

A 4x4 matrix of nucleotide substitution rates. The rates are normalized, therefore each row sums up to 1.

Note

`HH_S1F` replaces `HS1FDistance` in versions of SHazaM prior to 0.1.5.

References

1. Yaari G, et al. Models of somatic hypermutation targeting and substitution based on synonymous mutations from high-throughput immunoglobulin sequencing data. Front Immunol. 2013 4(November):358.

See Also

See HKL_S1F for the human light chain 1-mer substitution model and MK_RS1NF for the mouse light chain 1-mer substitution model.

`HH_S5F` *Human heavy chain, silent, 5-mer, functional targeting model.*

Description

5-mer model of somatic hypermutation targeting based on analysis of silent mutations in functional heavy chain Ig sequences from Homo sapiens.

Usage

```
HH_S5F
```

Format

A TargetingModel object.

References

1. Yaari G, et al. Models of somatic hypermutation targeting and substitution based on synonymous mutations from high-throughput immunoglobulin sequencing data. Front Immunol. 2013 4(November):358.

See Also

See HH_S1F for the 1-mer substitution matrix from the same publication; HKL_S5F for the human light chain 5-mer targeting model; MK_RS5NF for the mouse 5-mer targeting model; and U5N for the uniform 5-mer null targeting model.

`HKL_S1F` *Human kappa and lambda chain, silent, 1-mer, functional substitution model.*

Description

1-mer substitution model of somatic hypermutation based on analysis of silent mutations in functional kappa and lambda light chain Ig sequences from Homo sapiens.

Usage

```
HKL_S1F
```

Format

A 4x4 matrix of nucleotide substitution rates. The rates are normalized, therefore each row sums up to 1.

Note

Reported in Table III in Cui et al, 2016.

References

1. Cui A, Di Niro R, Vander Heiden J, Briggs A, Adams K, Gilbert T, O'Connor K, Vigneault F, Shlomchik M and Kleinstein S (2016). A Model of Somatic Hypermutation Targeting in Mice Based on High-Throughput Ig Sequencing Data. The Journal of Immunology, 197(9), 3566-3574.

See Also

See HH_S1F for the human heavy chain 1-mer substitution model and MK_RS1NF for the mouse light chain 1-mer substitution model.

HKL_S5F *Human kappa and lambda light chain, silent, 5-mer, functional targeting model.*

Description

5-mer model of somatic hypermutation targeting based on analysis of silent mutations in functional kappa and lambda light chain Ig sequences from Homo sapiens.

Usage

```
HKL_S5F
```

Format

A TargetingModel object.

References

1. Cui A, Di Niro R, Vander Heiden J, Briggs A, Adams K, Gilbert T, O'Connor K, Vigneault F, Shlomchik M and Kleinstein S (2016). A Model of Somatic Hypermutation Targeting in Mice Based on High-Throughput Ig Sequencing Data. The Journal of Immunology, 197(9), 3566-3574.

See Also

See HH_S5F for the human heavy chain 5-mer targeting model; MK_RS5NF for the mouse kappa light chain 5-mer targeting model; and U5N for the uniform 5-mer null targeting model.

IMGT_SCHEMES *IMGT unique numbering schemes*

Description

Sequence region definitions according to the IMGT unique numbering scheme.

Format

A RegionDefinition object defining:

- `IMGT_V`: The IMGT numbered V segment up to position nucleotide 312. This definition combines the CDR1 and CDR2 into a single CDR region, and FWR1, FWR2 and FWR3 into a single FWR region. CDR3 and FWR4 are excluded as they are downstream of nucleotide 312.
- `IMGT_V_BY_CODONS`: The IMGT numbered V segment up to position nucleotide 312. This definition treats each codon, from codon 1 to codon 104, as a distinct region.

- `IMGT_V_BY_REGIONS`: The IMGT numbered V segment up to position nucleotide 312. This defines separate regions for each of CDR1, CDR2, FWR1, FWR2 and FWR3. CDR3 and FWR4 are excluded as they are downstream of nucleotide 312.
- `IMGT_V_BY_SEGMENTS`: The IMGT numbered V segment up to position nucleotide 312. This definition has no subdivisions and treats the entire V segment as a single region.
- `IMGT_VDJ`: IMGT numbered regions for CDR1-3 and FWR1-4 with combined CDR and FWR definitions spanning CDR1-3 and FWR1-4, respectively. Note, unless the definition object has been updated using setRegionBoundaries this schema will have a value of `0` for the `seqLength` slot and the `boundaries` slot will be empty. This is because these slots depend on the junction length which is unknown in the template scheme. After setRegionBoundaries has been run, these slots will be populated with the appropriate values for the specified sequence and junction length.
- `IMGT_VDJ_BY_REGIONS`: The IMGT numbered regions for FWR1-4 and CDR1-3 with separate region boundaries for each of CDR1, CDR2, CDR3, FWR1, FWR2, FWR3 and FWR4. Note, unless the definition object has been updated using setRegionBoundaries this schema will have a value of `0` for the `seqLength` slot and the `boundaries` slot will be empty. This is because these slots depend on the junction length which is unknown in the template scheme. After setRegionBoundaries has been run, these slots will be populated with the appropriate values for the specified sequence and junction length.

References

1. Lefranc MP, et al. IMGT unique numbering for immunoglobulin and T cell receptor variable domains and Ig superfamily V-like domains. Developmental and comparative immunology. 2003 27:55-77.

makeAverage1merMut — *Make a 1-mer mutability model by averaging over a 5-mer mutability model*

Description

`makeAverage1merMut` averages mutability rates in a 5-mer mutability model to derive a 1-mer mutability model.

Usage

```
makeAverage1merMut(mut5mer)
```

Arguments

`mut5mer`	a named vector of length 1024 such as that returned by `createMutabilityMatrix` and that returned by `makeDegenerate5merMut` with `extended=FALSE`. Names should correspond to 5-mers made up of "A", "T", "G", and "C" (case-insensitive). `NA` values are allowed.

Details

For example, the mutability rate of "A" in the resultant 1-mer model is derived by averaging the mutability rates of all the 5-mers that have an "A" as their central 1-mer, followed by normalization.

Value

A named vector of length 4 containing normalized mutability rates.

See Also

See makeDegenerate5merMut for making a degenerate 5-mer mutability model based on a 1-mer mutability model.

Examples

```
# Make a degenerate 5-mer model (length of 1024) based on a 1-mer model
example1merMut <- c(A=0.2, T=0.1, C=0.4, G=0.3)
degenerate5merMut <- makeDegenerate5merMut(mut1mer = example1merMut)

# Now make a 1-mer model by averaging over the degenerate 5-mer model
# Expected to get back example1merMut
makeAverage1merMut(mut5mer = degenerate5merMut)
```

---

makeAverage1merSub *Make a 1-mer substitution model by averaging over a 5-mer substitution model*

Description

makeAverage1merSub averages substitution rates in a 5-mer substitution model to derive a 1-mer substitution model.

Usage

```
makeAverage1merSub(sub5mer)
```

Arguments

sub5mer	a 4x1024 matrix such as that returned by createSubstitutionMatrix and that returned by makeDegenerate5merSub with extended=FALSE. Column names should correspond to 5-mers containing the central 1-mer to mutate from. Row names should correspond to nucleotides to mutate into. Nucleotides should include "A", "T", "G", and "C" (case-insensitive).

Details

For example, the substitution rate from "A" to "T" in the resultant 1-mer model is derived by averaging the substitution rates into a "T" of all the 5-mers that have an "A" as their central 1-mer.

Value

A 4x4 matrix with row names representing nucleotides to mutate from and column names representing nucleotides to mutate into. Rates are normalized by row.

See Also

See makeDegenerate5merSub for making a degenerate 5-mer substitution model based on a 1-mer substitution model.

Examples

```
# Make a degenerate 5-mer model (4x1024) based on HKL_S1F (4x4)
degenerate5merSub <- makeDegenerate5merSub(sub1mer = HKL_S1F)

# Now make a 1-mer model by averaging over the degenerate 5-mer model
# Expected to get back HKL_S1F
makeAverage1merSub(sub5mer = degenerate5merSub)
```

---

makeDegenerate5merMut *Make a degenerate 5-mer mutability model based on a 1-mer mutability model*

Description

`makeDegenerate5merMut` populates mutability rates from a 1-mer mutability model into 5-mers with corresponding central 1-mers.

Usage

```
makeDegenerate5merMut(mut1mer, extended = FALSE)
```

Arguments

`mut1mer`	a named vector of length 4 containing (normalized) mutability rates. Names should correspond to nucleotides, which should include "A", "T", "G", and "C" (case-insensitive).
`extended`	whether to return the unextended (`extended=FALSE`) or extended (`extended=TRUE`) 5-mer mutability model. Default is `FALSE`.

Details

As a concrete example, consider a 1-mer mutability model in which mutability rates of "A", "T", "G", and "C" are, respectively, 0.14, 0.23, 0.31, and 0.32. In the resultant degenerate 5-mer mutability model, all the 5-mers that have an "A" as their central 1-mer would have mutability rate of 0.14/256, where 256 is the number of such 5-mers.

When `extended=TRUE`, `extendMutabilityMatrix` is called to extend the mutability vector of length 1024 into a vector of length 3125.

Value

For extended=FALSE, a vector of length 1024. The vector returned is normalized. For extended=TRUE, a vector of length 3125.

See Also

See makeAverage1merMut for making a 1-mer mutability model by taking the average of a 5-mer mutability model. See extendMutabilityMatrix for extending the mutability vector.

Examples

```
# Make a degenerate 5-mer model (length of 1024) based on a 1-mer model
example1merMut <- c(A=0.2, T=0.1, C=0.4, G=0.3)
degenerate5merMut <- makeDegenerate5merMut(mut1mer = example1merMut)

# Look at a few 5-mers
degenerate5merMut[c("AAAAT", "AACAT", "AAGAT", "AATAT")]

# Normalized
sum(degenerate5merMut)
```

---

makeDegenerate5merSub *Make a degenerate 5-mer substitution model based on a 1-mer substitution model*

Description

makeDegenerate5merSub populates substitution rates from a 1-mer substitution model into 5-mers with corresponding central 1-mers.

Usage

```
makeDegenerate5merSub(sub1mer, extended = FALSE)
```

Arguments

sub1mer	a 4x4 matrix containing (normalized) substitution rates. Row names should correspond to nucleotides to mutate from. Column names should correspond to nucleotides to mutate into. Nucleotides should include "A", "T", "G", and "C" (case-insensitive).
extended	whether to return the unextended (extended=FALSE) or extended (extended=TRUE) 5-mer substitution model. Default is FALSE.

Details

As a concrete example, consider a 1-mer substitution model in which substitution rates from "A" to "T", "G", and "C" are, respectively, 0.1, 0.6, and 0.3. In the resultant degenerate 5-mer substitution model, all the 5-mers (columns) that have an "A" as their central 1-mer would have substitution rates (rows) of 0.1, 0.6, and 0.3 to "T", "G", and "C" respectively.

When `extended=TRUE`, `extendSubstitutionMatrix` is called to extend the 4x1024 substitution matrix.

Value

For `extended=FALSE`, a 4x1024 matrix. For `extended=TRUE`, a 5x3125 matrix.

See Also

See makeAverage1merSub for making a 1-mer substitution model by taking the average of a 5-mer substitution model. See extendSubstitutionMatrix for extending the substitution matrix.

Examples

```
# Make a degenerate 5-mer model (4x1024) based on HKL_S1F (4x4)
# Note: not to be confused with HKL_S5F@substitution, which is non-degenerate
degenerate5merSub <- makeDegenerate5merSub(sub1mer = HKL_S1F)

# Look at a few 5-mers
degenerate5merSub[, c("AAAAT", "AACAT", "AAGAT", "AATAT")]
```

`makeGraphDf` *Build a data.frame from a ChangeoClone and an igraph object containing a clonal lineage*

Description

`makeGraphDf` creates a data.frame from a ChangeoClone and an igraph `graph` object containing a B cell lineage tree and associated sequence data. The data.frame contains the original fields and additions such as each sequence's parent in the lineage tree, the lineage germline, and additional rows for inferred sequences.

Usage

```
makeGraphDf(
  curCloneGraph,
  curCloneObj,
  objSeqId = "sequence_id",
  objSeq = "sequence"
)
```

Arguments

`curCloneGraph`	an igraph `graph` object for the lineage tree generated by buildPhylipLineage. Note that the field containing the nucleotide sequence in the object must be named `sequence`.
`curCloneObj`	ChangeoClone object used to generate the lineage.
`objSeqId`	name of the sequence identifier field in `curCloneObj`.
`objSeq`	name of the nucleotide sequence field in `curCloneObj`.

Value

A `data.frame` with sequence and lineage information, including the the parent nucleotide sequence in the lineage tree(`parent_sequence`), an internal parent identifier (`parent`), and additional rows for germline sequence and inferred intermediate sequences.

Values in the `sequence_id` field are renamed to numeric values, prefixed with the clonal grouping identifier and labeled as either `"Inferred"` or `"Germline"` if they are not an observed sequence. For example, for a lineage with `clone_id = 34` the new identifiers would be of the form: `"34_Germline"`, `"34_Inferred1"`, `"34_1"`, `"34_2"`, etc.

Note that the original sequence identifier is preserved in the `orig_sequence_id` field and the original parent sequence identifier is retained in `orig_parent`.

See Also

See observedMutations to calculate mutation frequencies using `parent_sequence` as the reference germline. See ChangeoClone, buildPhylipLineage, and graph for details on the input objects.

Examples

```
# Load and subset example data
data(ExampleDb, package = "alakazam")
data(ExampleTrees, package = "alakazam")
graph <- ExampleTrees[[17]]
db <- subset(ExampleDb, clone_id == graph$clone)
clone <- alakazam::makeChangeoClone(db)

# Extend data with lineage information
df <- makeGraphDf(graph, clone)
```

`minNumMutationsTune` *Parameter tuning for minNumMutations*

Description

`minNumMutationsTune` helps with picking a threshold value for `minNumMutations` in createSubstitutionMatrix by tabulating the number of 5-mers for which substitution rates would be computed directly or inferred at various threshold values.

Usage

```
minNumMutationsTune(subCount, minNumMutationsRange)
```

Arguments

`subCount`	`data.frame` returned by createSubstitutionMatrix with `numMutationsOnly=TRUE`.
`minNumMutationsRange`	a number or a vector indicating the value or range of values of `minNumMutations` to try.

Details

At a given threshold value of `minNumMutations`, for a given 5-mer, if the total number of mutations is greater than the threshold and there are mutations to every other base, substitution rates are computed directly for the 5-mer using its mutations. Otherwise, mutations from 5-mers with the same inner 3-mer as the 5-mer of interest are aggregated. If the number of such mutations is greater than the threshold and there are mutations to every other base, these mutations are used for inferring the substitution rates for the 5-mer of interest; if not, mutations from all 5-mers with the same center nucleotide are aggregated and used for inferring the substitution rates for the 5-mer of interest (i.e. the 1-mer model).

Value

A 3xn `matrix`, where n is the number of trial values of `minNumMutations` supplied in `minNumMutationsRange`. Each column corresponds to a value in `minNumMutationsRange`. The rows correspond to the number of 5-mers for which substitution rates would be computed directly using the 5-mer itself (`"5mer"`), using its inner 3-mer (`"3mer"`), and using the central 1-mer (`"1mer"`), respectively.

References

1. Yaari G, et al. Models of somatic hypermutation targeting and substitution based on synonymous mutations from high-throughput immunoglobulin sequencing data. Front Immunol. 2013 4(November):358.

See Also

See argument `numMutationsOnly` in createSubstitutionMatrix for generating the required input `data.frame subCount`. See argument `minNumMutations` in createSubstitutionMatrix for what it does.

Examples

```
# Subset example data to one isotype and sample as a demo
data(ExampleDb, package="alakazam")
db <- subset(ExampleDb, c_call == "IGHA" & sample_id == "-1h")

# Count the number of mutations per 5-mer
subCount <- createSubstitutionMatrix(db, sequenceColumn="sequence_alignment",
                                     germlineColumn="germline_alignment_d_mask",
                                     vCallColumn="v_call",
                                          model="s", multipleMutation="independent",
                                          returnModel="5mer", numMutationsOnly=TRUE)

# Tune minNumMutations
minNumMutationsTune(subCount, seq(from=10, to=80, by=10))
```

---

minNumSeqMutationsTune

Parameter tuning for minNumSeqMutations

Description

`minNumSeqMutationsTune` helps with picking a threshold value for `minNumSeqMutations` in createMutabilityMatrix by tabulating the number of 5-mers for which mutability would be computed directly or inferred at various threshold values.

Usage

```
minNumSeqMutationsTune(mutCount, minNumSeqMutationsRange)
```

Arguments

`mutCount`	a `vector` of length 1024 returned by createMutabilityMatrix with `numSeqMutationsOnly=TRUE`.
`minNumSeqMutationsRange`	a number or a vector indicating the value or the range of values of `minNumSeqMutations` to try.

Details

At a given threshold value of `minNumSeqMutations`, for a given 5-mer, if the total number of mutations is greater than the threshold, mutability is computed directly. Otherwise, mutability is inferred.

Value

A 2xn `matrix`, where n is the number of trial values of `minNumSeqMutations` supplied in `minNumSeqMutationsRange`. Each column corresponds to a value in `minNumSeqMutationsRange`. The rows correspond to the number of 5-mers for which mutability would be computed directly (`"measured"`) and inferred (`"inferred"`), respectively.

References

1. Yaari G, et al. Models of somatic hypermutation targeting and substitution based on synonymous mutations from high-throughput immunoglobulin sequencing data. Front Immunol. 2013 4(November):358.

See Also

See argument `numSeqMutationsOnly` in createMutabilityMatrix for generating the required input vector `mutCount`. See argument `minNumSeqMutations` in createMutabilityMatrix for what it does.

Examples

```
# Subset example data to one isotype and sample as a demo
data(ExampleDb, package="alakazam")
db <- subset(ExampleDb, c_call == "IGHA" & sample_id == "-1h")
set.seed(112)
db <- dplyr::slice_sample(db, n=75)
# Create model using only silent mutations
sub <- createSubstitutionMatrix(db, sequenceColumn="sequence_alignment",
                                germlineColumn="germline_alignment_d_mask",
                                vCallColumn="v_call",
                                model="s", multipleMutation="independent",
                                returnModel="5mer", numMutationsOnly=FALSE,
                                minNumMutations=20)

# Count the number of mutations in sequences containing each 5-mer
mutCount <- createMutabilityMatrix(db, substitutionModel = sub,
                                   sequenceColumn="sequence_alignment",
                                   germlineColumn="germline_alignment_d_mask",
                                   vCallColumn="v_call",
                                   model="s", multipleMutation="independent",
                                   numSeqMutationsOnly=TRUE)

# Tune minNumSeqMutations
minNumSeqMutationsTune(mutCount, seq(from=100, to=300, by=50))
```

`MK_RS1NF` *Mouse kappa chain, replacement and silent, 1-mer, non-functional substitution model.*

Description

1-mer substitution model of somatic hypermutation based on analysis of replacement and silent mutations in non-functional kappa light chain Ig sequences from NP-immunized Mus musculus.

Usage

```
MK_RS1NF
```

Format

A 4x4 matrix of nucleotide substitution rates. The rates are normalized, therefore each row sums up to 1.

Note

`MK_RS1NF` replaces `M1NDistance` from versions of SHazaM prior to 0.1.5.

References

1. Cui A, Di Niro R, Vander Heiden J, Briggs A, Adams K, Gilbert T, O'Connor K, Vigneault F, Shlomchik M and Kleinstein S (2016). A Model of Somatic Hypermutation Targeting in Mice Based on High-Throughput Ig Sequencing Data. The Journal of Immunology, 197(9), 3566-3574.

See Also

See HH_S1F for the human heavy chain 1-mer substitution model and HKL_S1F for the human light chain 1-mer substitution model.

`MK_RS5NF` *Mouse kappa light chain, replacement and silent, 5-mer, non-functional targeting model.*

Description

5-mer model of somatic hypermutation targeting based on analysis of replacement and silent mutations in non-functional kappa light chain Ig sequences from NP-immunized Mus musculus.

Usage

```
MK_RS5NF
```

Format

TargetingModel object.

References

1. Cui A, Di Niro R, Vander Heiden J, Briggs A, Adams K, Gilbert T, O'Connor K, Vigneault F, Shlomchik M and Kleinstein S (2016). A Model of Somatic Hypermutation Targeting in Mice Based on High-Throughput Ig Sequencing Data. The Journal of Immunology, 197(9), 3566-3574.

See Also

See MK_RS1NF for the 1-mer substitution matrix from the same publication; HH_S5F for the human heavy chain silent 5-mer functional targeting model; HKL_S5F for the human light chain silent 5-mer functional targeting model; and U5N for the uniform 5-mer null targeting model.

`MutabilityModel-class` *S4 class defining a mutability model*

Description

`MutabilityModel` defines a data structure for the 5-mer motif-based SHM targeting mutability model.

Usage

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

## S4 method for signature 'MutabilityModel'
as.data.frame(x)
```

Arguments

`x`	`MutabilityModel` object.

Slots

`.Data` numeric vector containing 5-mer mutability estimates

`source` character vector annotating whether the mutability was inferred or directly measured.

`numMutS` a number indicating the number of silent mutations used for estimating mutability

`numMutR` a number indicating the number of replacement mutations used for estimating mutability

`MutationDefinition-class` *S4 class defining replacement and silent mutation definitions*

Description

`MutationDefinition` defines a common data structure for defining the whether a mutation is annotated as a replacement or silent mutation.

Slots

`name` name of the MutationDefinition.

`description` description of the model and its source.

`classes` named character vectors with single-letter amino acid codes as names and amino acid classes as values, with `NA` assigned to set of characters `c("X", "*", "-", ".")`. Replacement (R) is be defined as a change in amino acid class and silent (S) as no change in class.

`codonTable` matrix of codons (columns) and substitutions (rows).

`citation` publication source.

See Also

See MUTATION_SCHEMES for a set of predefined MutationDefinition objects.

MUTATION_SCHEMES *Amino acid mutation definitions*

Description

Definitions of replacement (R) and silent (S) mutations for different amino acid physicochemical classes.

Format

A MutationDefinition object defining:

- CHARGE_MUTATIONS: Amino acid mutations are defined by changes in side chain charge class.
- HYDROPATHY_MUTATIONS: Amino acid mutations are defined by changes in side chain hydrophobicity class.
- POLARITY_MUTATIONS: Amino acid mutations are defined by changes in side chain polarity class.
- VOLUME_MUTATIONS: Amino acid mutations are defined by changes in side chain volume class.

References

1. https://www.imgt.org/IMGTeducation/Aide-memoire/_UK/aminoacids/IMGTclasses.html

---

observedMutations *Calculate observed numbers of mutations*

Description

observedMutations calculates the observed number of mutations for each sequence in the input data.frame.

Usage

```
observedMutations(
  db,
  sequenceColumn = "sequence_alignment",
  germlineColumn = "germline_alignment_d_mask",
  regionDefinition = NULL,
  mutationDefinition = NULL,
  ambiguousMode = c("eitherOr", "and"),
  frequency = FALSE,
  combine = FALSE,
  nproc = 1,
  cloneColumn = "clone_id",
  juncLengthColumn = "junction_length"
)
```

Arguments

`db`	`data.frame` containing sequence data.
`sequenceColumn`	`character` name of the column containing input sequences. IUPAC ambiguous characters for DNA are supported.
`germlineColumn`	`character` name of the column containing the germline or reference sequence. IUPAC ambiguous characters for DNA are supported.
`regionDefinition`	RegionDefinition object defining the regions and boundaries of the Ig sequences. If NULL, mutations are counted for entire sequence. To use regions definitions, sequences in `sequenceColumn` and `germlineColumn` must be aligned, following the IMGT schema.
`mutationDefinition`	MutationDefinition object defining replacement and silent mutation criteria. If `NULL` then replacement and silent are determined by exact amino acid identity.
`ambiguousMode`	whether to consider ambiguous characters as `"either or"` or `"and"` when determining and counting the type(s) of mutations. Applicable only if `sequenceColumn` and/or `germlineColumn` contain(s) ambiguous characters. One of `c("eitherOr", "and")`. Default is `"eitherOr"`.
`frequency`	`logical` indicating whether or not to calculate mutation frequencies. Default is `FALSE`.
`combine`	`logical` indicating whether for each sequence should the mutation counts for the different regions (CDR, FWR) and mutation types be combined and return one value of count/frequency per sequence instead of multiple values. Default is `FALSE`.
`nproc`	number of cores to distribute the operation over. If the cluster has already been set the call function with `nproc` = 0 to not reset or reinitialize. Default is `nproc` = 1.
`cloneColumn`	clone id column name in `db`
`juncLengthColumn`	junction length column name in `db`

Details

Mutation counts are determined by comparing a reference sequence to the input sequences in the column specified by `sequenceColumn`. See calcObservedMutations for more technical details, **including criteria for which sequence differences are included in the mutation counts and which are not**.

The mutations are binned as either replacement (R) or silent (S) across the different regions of the sequences as defined by `regionDefinition`. Typically, this would be the framework (FWR) and complementarity determining (CDR) regions of IMGT-gapped nucleotide sequences. Mutation counts are appended to the input db as additional columns.

If db includes lineage information, such as the parent_sequence column created by makeGraphDf, the reference sequence can be set to use that field as reference sequence using the germlineColumn argument.

Value

A modified db data.frame with observed mutation counts for each sequence listed. The columns names are dynamically created based on the regions in the regionDefinition. For example, when using the IMGT_V definition, which defines positions for CDR and FWR, the following columns are added:

- mu_count_cdr_r: number of replacement mutations in CDR1 and CDR2 of the V-segment.
- mu_count_cdr_s: number of silent mutations in CDR1 and CDR2 of the V-segment.
- mu_count_fwr_r: number of replacement mutations in FWR1, FWR2 and FWR3 of the V-segment.
- mu_count_fwr_s: number of silent mutations in FWR1, FWR2 and FWR3 of the V-segment.

If frequency=TRUE, R and S mutation frequencies are calculated over the number of non-N positions in the specified regions.

- mu_freq_cdr_r: frequency of replacement mutations in CDR1 and CDR2 of the V-segment.
- mu_freq_cdr_s: frequency of silent mutations in CDR1 and CDR2 of the V-segment.
- mu_freq_fwr_r: frequency of replacement mutations in FWR1, FWR2 and FWR3 of the V-segment.
- mu_freq_fwr_s: frequency of silent mutations in FWR1, FWR2 and FWR3 of the V-segment.

If frequency=TRUE and combine=TRUE, the mutations and non-N positions are aggregated and a single mu_freq value is returned

- mu_freq: frequency of replacement and silent mutations in the specified region

See Also

calcObservedMutations is called by this function to get the number of mutations in each sequence grouped by the RegionDefinition. See IMGT_SCHEMES for a set of predefined RegionDefinition objects. See expectedMutations for calculating expected mutation frequencies. See makeGraphDf for creating the field parent_sequence.

Examples

```
# Subset example data
data(ExampleDb, package="alakazam")
db <- ExampleDb[1:10, ]

# Calculate mutation frequency over the entire sequence
db_obs <- observedMutations(db, sequenceColumn="sequence_alignment",
                            germlineColumn="germline_alignment_d_mask",
                            frequency=TRUE,
                            nproc=1)

# Count of V-region mutations split by FWR and CDR
# With mutations only considered replacement if charge changes
db_obs <- observedMutations(db, sequenceColumn="sequence_alignment",
                            germlineColumn="germline_alignment_d_mask",
                            regionDefinition=IMGT_V,
                            mutationDefinition=CHARGE_MUTATIONS,
                            nproc=1)

# Count of VDJ-region mutations, split by FWR and CDR
db_obs <- observedMutations(db, sequenceColumn="sequence_alignment",
                            germlineColumn="germline_alignment_d_mask",
                            regionDefinition=IMGT_VDJ,
                            nproc=1)

# Extend data with lineage information
data(ExampleTrees, package="alakazam")
graph <- ExampleTrees[[17]]
clone <- alakazam::makeChangeoClone(subset(ExampleDb, clone_id == graph$clone))
gdf <- makeGraphDf(graph, clone)

# Count of mutations between observed sequence and immediate ancestor
db_obs <- observedMutations(gdf, sequenceColumn="sequence",
                            germlineColumn="parent_sequence",
                            regionDefinition=IMGT_VDJ,
                            nproc=1)
```

`plotBaselineDensity` *Plots BASELINe probability density functions*

Description

`plotBaselineDensity` plots the probability density functions resulting from selection analysis using the BASELINe method.

Usage

```
plotBaselineDensity(
  baseline,
  idColumn,
  groupColumn = NULL,
  colorElement = c("id", "group"),
  colorValues = NULL,
  title = NULL,
  subsetRegions = NULL,
  sigmaLimits = c(-5, 5),
  facetBy = c("region", "group"),
  style = c("density"),
  sizeElement = c("none", "id", "group"),
  size = 1,
  silent = FALSE,
  ...
)
```

Arguments

`baseline`	`Baseline` object containing selection probability density functions.
`idColumn`	name of the column in the `db` slot of `baseline` containing primary identifiers.
`groupColumn`	name of the column in the `db` slot of `baseline` containing secondary grouping identifiers. If `NULL`, organize the plot only on values in `idColumn`.
`colorElement`	one of `c("id", "group")` specifying whether the `idColumn` or `groupColumn` will be used for color coding. The other entry, if present, will be coded by line style.
`colorValues`	named vector of colors for entries in `colorElement`, with names defining unique values in the `colorElement` 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.
`title`	string defining the plot title.
`subsetRegions`	character vector defining a subset of regions to plot, corresponding to the regions for which the `baseline` data was calculated. If `NULL` all regions in `baseline` are plotted.
`sigmaLimits`	numeric vector containing two values defining the `c(lower, upper)` bounds of the selection scores to plot.
`facetBy`	one of `c("region", "group")` specifying which category to facet the plot by, either values in `groupColumn` ("group") or regions defined in the `regions` slot of the `baseline` object ("region"). If this is set to "group", then the region will behave as the `groupColumn` for purposes of the `colorElement` argument.
`style`	type of plot to draw. One of: • `"density"`: plots a set of curves for each probability density function in `baseline`, with colors determined by values in the `colorElement` column. Faceting is determined by the `facetBy` argument.
`sizeElement`	one of `c("none", "id", "group")` specifying whether the lines in the plot should be all of the same size (`none`) or have their sizes depend on the values in `id` or `code`.
`size`	numeric scaling factor for lines, points and text in the plot.
`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

Takes as input a Baseline object returned from groupBaseline.

Examples

```
## Not run:
# Subset example data as a demo
data(ExampleDb, package="alakazam")
db <- subset(ExampleDb, c_call %in% c("IGHM", "IGHG"))
set.seed(112)
db <- dplyr::slice_sample(db, n=100)

# Collapse clones
db <- collapseClones(db, cloneColumn="clone_id",
                     sequenceColumn="sequence_alignment",
                     germlineColumn="germline_alignment_d_mask",
                     method="thresholdedFreq", minimumFrequency=0.6,
                     includeAmbiguous=FALSE, breakTiesStochastic=FALSE)

# Calculate BASELINe
baseline <- calcBaseline(db,
                         sequenceColumn="clonal_sequence",
                         germlineColumn="clonal_germline",
                         testStatistic="focused",
                         regionDefinition=IMGT_V,
                         targetingModel=HH_S5F,
                         nproc=1)

# Grouping the PDFs by the sample and isotype annotations
grouped <- groupBaseline(baseline, groupBy=c("sample_id", "c_call"))

# Plot density faceted by region with custom isotype colors
isotype_colors <- c("IGHM"="darkorchid", "IGHD"="firebrick",
                    "IGHG"="seagreen", "IGHA"="steelblue")
plotBaselineDensity(grouped, "sample_id", "c_call", colorValues=isotype_colors,
                    colorElement="group", sigmaLimits=c(-1, 1))

# Facet by isotype instead of region
sample_colors <- c("-1h"="steelblue", "+7d"="firebrick")
plotBaselineDensity(grouped, "sample_id", "c_call", facetBy="group",
                    colorValues=sample_colors, sigmaLimits=c(-1, 1))

## End(Not run)
```

`plotBaselineSummary` *Plots BASELINe summary statistics*

Description

`plotBaselineSummary` plots a summary of the results of selection analysis using the BASELINe method.

Usage

```
plotBaselineSummary(
  baseline,
  idColumn,
  groupColumn = NULL,
  groupColors = NULL,
  subsetRegions = NULL,
  facetBy = c("region", "group"),
  title = NULL,
  style = c("summary"),
  size = 1,
  silent = FALSE,
  ...
)
```

Arguments

`baseline`	either a data.frame returned from summarizeBaseline or a `Baseline` object returned from groupBaseline containing selection probability density functions and summary statistics.
`idColumn`	name of the column in `baseline` containing primary identifiers. If the input is a `Baseline` object, then this will be a column in the `stats` slot of `baseline`.
`groupColumn`	name of the column in `baseline` containing secondary grouping identifiers. If the input is a `Baseline` object, then this will be a column in the `stats` slot of `baseline`.
`groupColors`	named vector of colors for entries in `groupColumn`, with names defining unique values in the `groupColumn` and values being colors. Also controls the order in which groups appear on the plot. If `NULL` alphabetical ordering and a default color palette will be used. Has no effect if `facetBy="group"`.
`subsetRegions`	character vector defining a subset of regions to plot, corresponding to the regions for which the `baseline` data was calculated. If `NULL` all regions in `baseline` are plotted.
`facetBy`	one of c("group", "region") specifying which category to facet the plot by, either values in `groupColumn` ("group") or regions defined in `baseline` ("region"). The data that is not used for faceting will be color coded.
`title`	string defining the plot title.
`style`	type of plot to draw. One of: • `"summary"`: plots the mean and confidence interval for the selection scores of each value in `idColumn`. Faceting and coloring are determine by values in `groupColumn` and regions defined in `baseline`, depending upon the `facetBy` argument.

`size`	numeric scaling factor for lines, points and text in the plot.
`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

Takes as input either a Baseline object returned by groupBaseline or a data.frame returned from summarizeBaseline.

Examples

```
# Subset example data as a demo
data(ExampleDb, package="alakazam")
db <- subset(ExampleDb, c_call %in% c("IGHM", "IGHG"))
set.seed(112)
db <- dplyr::slice_sample(db, n=25)

# Collapse clones
db <- collapseClones(db, cloneColumn="clone_id",
                     sequenceColumn="sequence_alignment",
                     germlineColumn="germline_alignment_d_mask",
                     method="thresholdedFreq", minimumFrequency=0.6,
                     includeAmbiguous=FALSE, breakTiesStochastic=FALSE)

# Calculate BASELINe
baseline <- calcBaseline(db,
                         sequenceColumn="clonal_sequence",
                         germlineColumn="clonal_germline",
                         testStatistic="focused",
                         regionDefinition=IMGT_V,
                         targetingModel=HH_S5F,
                         nproc=1)

# Grouping the PDFs by sample and isotype annotations
grouped <- groupBaseline(baseline, groupBy=c("sample_id", "c_call"))

# Plot mean and confidence interval by region with custom group colors
isotype_colors <- c("IGHM"="darkorchid", "IGHD"="firebrick",
                    "IGHG"="seagreen", "IGHA"="steelblue")
plotBaselineSummary(grouped, "sample_id", "c_call",
                    groupColors=isotype_colors, facetBy="region")
```

plotDensityThreshold *Plot findThreshold results for the density method*

Description

plotDensityThreshold plots the results from "density" method of findThreshold, including the smoothed density estimate, input nearest neighbor distance histogram, and threshold selected.

Usage

```
plotDensityThreshold(
  data,
  cross = NULL,
  xmin = NULL,
  xmax = NULL,
  breaks = NULL,
  binwidth = NULL,
  title = NULL,
  size = 1,
  silent = FALSE,
  ...
)
```

Arguments

data	DensityThreshold object output by the "density" method of findThreshold.
cross	numeric vector of distances from distToNearest to draw as a histogram below the data histogram for comparison purposes.
xmin	minimum limit for plotting the x-axis. If NULL the limit will be set automatically.
xmax	maximum limit for plotting the x-axis. If NULL the limit will be set automatically.
breaks	number of breaks to show on the x-axis. If NULL the breaks will be set automatically.
binwidth	binwidth for the histogram. If NULL the binwidth will be set automatically to the bandwidth parameter determined by findThreshold.
title	string defining the plot title.
size	numeric value for the plot line sizes.
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 DensityThreshold for the the input object definition and findThreshold for generating the input object. See distToNearest calculating nearest neighbor distances.

Examples

```
# Subset example data to one sample as a demo
data(ExampleDb, package="alakazam")
db <- subset(ExampleDb, sample_id == "-1h")

# Use nucleotide Hamming distance and normalize by junction length
db <- distToNearest(db, sequenceColumn="junction", vCallColumn="v_call_genotyped",
                    jCallColumn="j_call", model="ham", normalize="len", nproc=1)

# To find the threshold cut, call findThreshold function for "gmm" method.
output <- findThreshold(db$dist_nearest, method="density")
print(output)

# Plot
plotDensityThreshold(output)
```

---

plotGmmThreshold   *Plot findThreshold results for the gmm method*

Description

plotGmmThreshold plots the results from "gmm" method of findThreshold, including the Gaussian distributions, input nearest neighbor distance histogram, and threshold selected.

Usage

```
plotGmmThreshold(
  data,
  cross = NULL,
  xmin = NULL,
  xmax = NULL,
  breaks = NULL,
  binwidth = NULL,
  title = NULL,
  size = 1,
  silent = FALSE,
  ...
)
```

Arguments

`data`	GmmThreshold object output by the `"gmm"` method of findThreshold.
`cross`	numeric vector of distances from distToNearest to draw as a histogram below the `data` histogram for comparison purposes.
`xmin`	minimum limit for plotting the x-axis. If `NULL` the limit will be set automatically.
`xmax`	maximum limit for plotting the x-axis. If `NULL` the limit will be set automatically.
`breaks`	number of breaks to show on the x-axis. If `NULL` the breaks will be set automatically.
`binwidth`	binwidth for the histogram. If `NULL` the binwidth will be set automatically.
`title`	string defining the plot title.
`size`	numeric value for lines in the plot.
`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 GmmThreshold for the the input object definition and findThreshold for generating the input object. See distToNearest calculating nearest neighbor distances.

Examples

```
# Subset example data to one sample as a demo
data(ExampleDb, package="alakazam")
db <- subset(ExampleDb, sample_id == "-1h")

# Use nucleotide Hamming distance and normalize by junction length
db <- distToNearest(db, sequenceColumn="junction", vCallColumn="v_call_genotyped",
                    jCallColumn="j_call", model="ham", normalize="len", nproc=1)

# To find the threshold cut, call findThreshold function for "gmm" method.
output <- findThreshold(db$dist_nearest, method="gmm", model="norm-norm", cutoff="opt")
print(output)

# Plot results
plotGmmThreshold(output, binwidth=0.02)
```

plotMutability *Plot mutability probabilities*

Description

`plotMutability` plots the mutability rates of a `TargetingModel`.

Usage

```
plotMutability(
  model,
  nucleotides = c("A", "C", "G", "T"),
  mark = NULL,
  style = c("hedgehog", "bar"),
  size = 1,
  silent = FALSE,
  ...
)
```

Arguments

`model`	TargetingModel object or vector containing normalized mutability rates.
`nucleotides`	vector of center nucleotide characters to plot.
`mark`	vector of 5-mer motifs to highlight in the plot. If `NULL` only highlight classical hot and cold spot motifs.
`style`	type of plot to draw. One of: • `"hedgehog"`: circular plot showing higher mutability scores further from the circle. The 5-mer is denoted by the values of the inner circle. The 5-mer is read from the most interior position of the 5-mer (5') to most exterior position (3'), with the center nucleotide in the center ring. Note, the order in which the 5-mers are plotted is different for nucleotides `c("A", "C")` and `c("G", "T")`. • `"bar"`: bar plot of mutability similar to the hedgehog style with the most 5' positions of each 5-mer at the base of the plot.
`size`	numeric scaling factor for lines and text in the plot.
`silent`	if `TRUE` do not draw the plot and just return the ggplot2 objects; if `FALSE` draw the plot.
`...`	additional arguments to pass to ggplot2::theme.

Value

A named list of ggplot objects defining the plots, with names defined by the center nucleotide for the plot object.

See Also

Takes as input a TargetingModel object. See createTargetingModel for model building.

Examples

```
# Plot one nucleotide in circular style
plotMutability(HH_S5F, "C")

# Plot two nucleotides in barchart style
plotMutability(HH_S5F, c("G", "T"), style="bar")
```

---

plotSlideWindowTune *Visualize parameter tuning for sliding window approach*

Description

Visualize results from slideWindowTune

Usage

```
plotSlideWindowTune(
  tuneList,
  plotFiltered = c("filtered", "remaining", "per_mutation"),
  percentage = FALSE,
  jitter.x = FALSE,
  jitter.x.amt = 0.1,
  jitter.y = FALSE,
  jitter.y.amt = 0.1,
  pchs = 1:length(tuneList),
  ltys = 1:length(tuneList),
  cols = 1,
  plotLegend = TRUE,
  legendPos = "topright",
  legendHoriz = FALSE,
  legendCex = 1,
  title = NULL,
  returnRaw = FALSE
)
```

Arguments

`tuneList`	a list of logical matrices returned by slideWindowTune.
`plotFiltered`	whether to plot the number of filtered ('filtered'), or remaining ('remaining') sequences for each mutation threshold. Use 'per_mutation' to plot the number of sequences at each mutation value. Default is `'filtered'`.

`percentage`	whether to plot on the y-axis the percentage of filtered sequences (as opposed to the absolute number). Default is `FALSE`.
`jitter.x`	whether to jitter x-axis values. Default is `FALSE`.
`jitter.x.amt`	amount of jittering to be applied on x-axis values if `jitter.x=TRUE`. Default is 0.1.
`jitter.y`	whether to jitter y-axis values. Default is `FALSE`.
`jitter.y.amt`	amount of jittering to be applied on y-axis values if `jitter.y=TRUE`. Default is 0.1.
`pchs`	point types to pass on to plot. Default is `1:length(tuneList)`.
`ltys`	line types to pass on to plot. Default is `1:length(tuneList)`.
`cols`	colors to pass on to plot.
`plotLegend`	whether to plot legend. Default is `TRUE`.
`legendPos`	position of legend to pass on to legend. Can be either a numeric vector specifying x-y coordinates, or one of `"topright"`, `"center"`, etc. Default is `"topright"`.
`legendHoriz`	whether to make legend horizontal. Default is `FALSE`.
`legendCex`	numeric values by which legend should be magnified relative to 1.
`title`	plot main title. Default is NULL (no title)
`returnRaw`	Return a data.frame with sequence counts (TRUE) or a plot. Default is `FALSE`.

Details

For each `windowSize`, if `plotFiltered='filtered'`, the x-axis represents a mutation threshold range, and the y-axis the number of sequences that have at least that number of mutations. If `plotFiltered='remaining'`, the y-axis represents the number of sequences that have less mutations than the mutation threshold range. For the same window size, a sequence can be included in the counts for different mutation thresholds. For example, sequence "CCACCAAAA" with germline "AAAAAAAAA" has 4 mutations. This sequence has at least 2 mutations and at least 3 mutations, in a window of size 4. the sequence will be included in the sequence count for mutation thresholds 2 and 3. If `plotFiltered='per_mutation'`, the sequences are counted only once for each window size, at their largest mutation threshold. The above example sequence would be included in the sequence count for mutation threshold 3.

When plotting, a user-defined `amount` of jittering can be applied on values plotted on either axis or both axes via adjusting `jitter.x`, `jitter.y`, `jitter.x.amt` and `jitter.y.amt`. This may be help with visually distinguishing lines for different window sizes in case they are very close or identical to each other. If plotting percentages (`percentage=TRUE`) and using jittering on the y-axis values (`jitter.y=TRUE`), it is strongly recommended that `jitter.y.amt` be set very small (e.g. 0.01).

`NA` for a combination of `mutThresh` and `windowSize` where `mutThresh` is greater than `windowSize` will not be plotted.

See Also

See slideWindowTune for how to get `tuneList`. See jitter for use of `amount` of jittering.

Examples

```
# Use an entry in the example data for input and germline sequence
data(ExampleDb, package="alakazam")

# Try out thresholds of 2-4 mutations in window sizes of 3-5 nucleotides
# on a subset of ExampleDb
tuneList <- slideWindowTune(db = ExampleDb[1:10, ],
                           mutThreshRange = 2:4, windowSizeRange = 3:5,
                           verbose = FALSE)

# Visualize
# Plot numbers of sequences filtered without jittering y-axis values
plotSlideWindowTune(tuneList, pchs=1:3, ltys=1:3, cols=1:3,
                    plotFiltered='filtered', jitter.y=FALSE)

# Notice that some of the lines overlap
# Jittering could help
plotSlideWindowTune(tuneList, pchs=1:3, ltys=1:3, cols=1:3,
                    plotFiltered='filtered', jitter.y=TRUE)

# Plot numbers of sequences remaining instead of filtered
plotSlideWindowTune(tuneList, pchs=1:3, ltys=1:3, cols=1:3,
                    plotFiltered='remaining', jitter.y=TRUE,
                    legendPos="bottomright")

# Plot percentages of sequences filtered with a tiny amount of jittering
plotSlideWindowTune(tuneList, pchs=1:3, ltys=1:3, cols=1:3,
                    plotFiltered='filtered', percentage=TRUE,
                    jitter.y=TRUE, jitter.y.amt=0.01)
```

---

plotTune *Visualize parameter tuning for minNumMutations and minNumSeqMutations*

Description

Visualize results from minNumMutationsTune and minNumSeqMutationsTune

Usage

```
plotTune(
  tuneMtx,
  thresh,
  criterion = c("5mer", "3mer", "1mer", "3mer+1mer", "measured", "inferred"),
  pchs = 1,
  ltys = 2,
  cols = 1,
  plotLegend = TRUE,
  legendPos = "topright",
  legendHoriz = FALSE,
  legendCex = 1
)
```

Arguments

`tuneMtx`	a `matrix` or a `list` of matrices produced by either minNumMutationsTune or minNumSeqMutationsTune. In the case of a list, it is assumed that each matrix corresponds to a sample and that all matrices in the list were produced using the same set of trial values of `minNumMutations` or `minNumSeqMutations`.
`thresh`	a number or a vector of indicating the value or the range of values of `minNumMutations` or `minNumSeqMutations` to plot. Should correspond to the columns of `tuneMtx`.
`criterion`	one of `"5mer"`, `"3mer"`, `"1mer"`, or `"3mer+1mer"` (for `tuneMtx` produced by minNumMutationsTune), or either `"measured"` or `"inferred"` (for `tuneMtx` produced by minNumSeqMutationsTune).
`pchs`	point types to pass on to plot.
`ltys`	line types to pass on to plot.
`cols`	colors to pass on to plot.
`plotLegend`	whether to plot legend. Default is `TRUE`. Only applicable if `tuneMtx` is a named list with names of the matrices corresponding to the names of the samples.
`legendPos`	position of legend to pass on to legend. Can be either a numeric vector specifying x-y coordinates, or one of `"topright"`, `"center"`, etc. Default is `"topright"`.
`legendHoriz`	whether to make legend horizontal. Default is `FALSE`.
`legendCex`	numeric values by which legend should be magnified relative to 1.

Details

For `tuneMtx` produced by minNumMutationsTune, for each sample, depending on `criterion`, the numbers of 5-mers for which substitution rates are directly computed (`"5mer"`), inferred based on inner 3-mers (`"3mer"`), inferred based on central 1-mers (`"1mer"`), or inferred based on inner 3-mers and central 1-mers (`"3mer+1mer"`) are plotted on the y-axis against values of `minNumMutations` on the x-axis.

For `tuneMtx` produced by minNumSeqMutationsTune, for each sample, depending on `criterion`, the numbers of 5-mers for which mutability rates are directly measured (`"measured"`) or inferred (`"inferred"`) are plotted on the y-axis against values of `minNumSeqMutations` on the x-axis.

Note that legends will be plotted only if `tuneMtx` is a supplied as a named `list` of matrices, ideally with names of each `matrix` corresponding to those of the samples based on which the matrices were produced, even if `plotLegend=TRUE`.

See Also

See minNumMutationsTune and minNumSeqMutationsTune for generating `tuneMtx`.

Examples

```
# Subset example data to one isotype and 200 sequences
data(ExampleDb, package="alakazam")
db <- subset(ExampleDb, c_call == "IGHA")
set.seed(112)
db <- dplyr::slice_sample(db, n=50)

tuneMtx = list()
for (i in 1:length(unique(db$sample_id))) {
    # Get data corresponding to current sample
    curDb = db[db[["sample_id"]] == unique(db[["sample_id"]])[i], ]

    # Count the number of mutations per 5-mer
    subCount = createSubstitutionMatrix(db=curDb, model="s",
                                        sequenceColumn="sequence_alignment",
                                        germlineColumn="germline_alignment_d_mask",
                                        vCallColumn="v_call",
                                        multipleMutation="independent",
                                        returnModel="5mer", numMutationsOnly=TRUE)

    # Tune over minNumMutations = 5..50
    subTune = minNumMutationsTune(subCount, seq(from=5, to=50, by=5))

    tuneMtx = c(tuneMtx, list(subTune))
}

# Name tuneMtx after sample names
names(tuneMtx) = unique(db[["sample_id"]])

# plot with legend for both samples for a subset of minNumMutations values
plotTune(tuneMtx, thresh=c(5, 15, 25, 40), criterion="3mer",
         pchs=16:17, ltys=1:2, cols=2:3,
         plotLegend=TRUE, legendPos=c(25, 30))

# plot for only 1 sample for all the minNumMutations values (no legend)
plotTune(tuneMtx[[1]], thresh=seq(from=5, to=50, by=5), criterion="3mer")
```

RegionDefinition-class *S4 class defining a region definition*

Description

RegionDefinition defines a common data structure for defining the region boundaries of an Ig sequence.

Slots

`name` name of the RegionDefinition.

`description` description of the model and its source.

`boundaries` `factor` defining the region boundaries of the sequence. The levels and values of `boundaries` determine the number of regions.

`seqLength` length of the sequence.

`regions` levels of the boundaries; e.g, `c("cdr", "fwr")`.

`labels` labels for the boundary and mutations combinations; e.g., `c("cdr_r", "cdr_s", "fwr_r", "fwr_s")`.

`citation` publication source.

See Also

See IMGT_SCHEMES for a set of predefined `RegionDefinition` objects.

`setRegionBoundaries` *Build a RegionDefinition object that includes CDR3 and FWR4.*

Description

`setRegionBoundaries` takes as input a junction length and an IMGT-numbered sequence and outputs a custom `RegionDefinition` object that includes the boundary definitions of CDR1-3 and FWR1-4 for that sequence. In contrast to the universal `RegionDefinition` object that end with FWR3, the returned definition is per-sequence due to variable junction lengths.

Usage

```
setRegionBoundaries(juncLength, sequenceImgt, regionDefinition = NULL)
```

Arguments

`juncLength`	junction length of the sequence.
`sequenceImgt`	IMGT-numbered sequence.
`regionDefinition`	`RegionDefinition` type to calculate the region definition for. Can be one of `IMGT_VDJ_BY_REGIONS` or `IMGT_VDJ`, which are template definitions that include CDR1-3 and FWR1-4. Only these two regions include all CDR1-3 and FWR1-4 regions. If this argument is set to `NULL`, then an empty `RegionDefinition` will be returned.

Value

A `RegionDefinition` object that includes CDR1-3 and FWR1-4 for the `sequenceImgt`, `juncLength`, and `regionDefinition` specified.

For `regionDefinition=IMGT_VDJ_BY_REGIONS`, the returned `RegionDefinition` includes:

- `fwr1`: Positions 1 to 78.
- `cdr1`: Positions 79 to 114.
- `fwr2`: Positions 115 to 165.
- `cdr2`: Positions 166 to 195.
- `fwr3`: Positions 196 to 312.
- `cdr3`: Positions 313 to (313 + juncLength - 6) since the junction sequence includes (on the left) the last codon from FWR3 and (on the right) the first codon from FWR4.
- `fwr4`: Positions (313 + juncLength - 6 + 1) to the end of the sequence.

For `regionDefinition=IMGT_VDJ`, the returned `RegionDefinition` includes:

- `fwr`: Positions belonging to a FWR.
- `cdr`: Positions belonging to a CDR.

In the case that the `regionDefinition` argument is not one of the extended regions (`IMGT_VDJ_BY_REGIONS` or `IMGT_VDJ`), the input `regionDefinition` is returned as is.

See Also

See RegionDefinition for the return object. See IMGT_SCHEMES for a set of predefined `RegionDefinition` objects.

Examples

```
# Load and subset example data
data(ExampleDb, package = "alakazam")
len <- ExampleDb$junction_length[1]
sequence <- ExampleDb$sequence_alignment[1]
region <- setRegionBoundaries(len, sequence, regionDefinition = IMGT_VDJ)
```

`shazam` *The shazam package*

Description

Dramatic improvements in high-throughput sequencing technologies now enable large-scale characterization of Ig repertoires, defined as the collection of transmembrane antigen-receptor proteins located on the surface of T and B lymphocytes. The `shazam` package provides tools for advanced analysis of somatic hypermutation (SHM) in immunoglobulin (Ig) sequences. The key functions in `shazam`, broken down topic, are described below.

Mutational profiling

shazam provides tools to quantify the extent and nature of SHM within full length V(D)J sequences as well as sub-regions (eg, FWR and CDR). Quantification of expected mutational loaded, under specific SHM targeting models, can also be performed along with model driven simulations of SHM.

- collapseClones: Build clonal consensus sequences.
- consensusSequence: Build a single consensus sequence.
- observedMutations: Compute observed mutation counts and frequencies.
- expectedMutations: Compute expected mutation frequencies.
- shmulateSeq: Simulate mutations in a single sequence.
- shmulateTree: Simulate mutations over a lineage tree.
- setRegionBoundaries: Extends a region definition to include CDR3 and FWR4.

SHM targeting models

Computational models and analyses of SHM have separated the process into two independent components:

1. A mutability model that defines where mutations occur.
2. A nucleotide substitution model that defines the resulting mutation.

Collectively these are what form the targeting model of SHM. shazam provides empirically derived targeting models for both humans and mice, along with tools to build these mutability and substitution models from data.

- createTargetingModel: Build a 5-mer targeting model.
- plotMutability: Plot 5-mer mutability rates.
- HH_S5F: Human 5-mer SHM targeting model.
- MK_RS5NF: Mouse 5-mer SHM targeting model.

Quantification of selection pressure

Bayesian Estimation of Antigen-driven Selection in Ig Sequences is a novel method for quantifying antigen-driven selection in high-throughput Ig sequence data. Targeting models created using shazam can be used to estimate the null distribution of expected mutation frequencies used by BASELINe, providing measures of selection pressure informed by known AID targeting biases.

- calcBaseline: Calculate the BASELINe probability density functions (PDFs).
- groupBaseline: Combine PDFs from sequences grouped by biological or experimental relevance.
- summarizeBaseline: Compute summary statistics from BASELINe PDFs.
- testBaseline: Perform significance testing for the difference between BASELINe PDFs.
- plotBaselineDensity: Plot the probability density functions resulting from selection analysis.
- plotBaselineSummary: Plot summary statistics resulting from selection analysis.

Mutational distance calculation

shazam provides tools to compute evolutionary distances between sequences or groups of sequences, which can leverage SHM targeting models. This information is particularly useful in understanding and defining clonal relationships.

- findThreshold: Identify clonal assignment threshold based on distances to nearest neighbors.
- distToNearest: Tune clonal assignment thresholds by calculating distances to nearest neighbors.
- calcTargetingDistance: Construct a nucleotide distance matrix from a 5-mer targeting model.

References

1. Hershberg U, et al. Improved methods for detecting selection by mutation analysis of Ig V region sequences. Int Immunol. 2008 20(5):683-94.
2. Uduman M, et al. Detecting selection in immunoglobulin sequences. Nucleic Acids Res. 2011 39(Web Server issue):W499-504. (Corrections at http://selection.med.yale.edu/baseline/correction/)
3. Yaari G, et al. Quantifying selection in high-throughput immunoglobulin sequencing data sets. Nucleic Acids Res. 2012 40(17):e134.
4. Yaari G, et al. Models of somatic hypermutation targeting and substitution based on synonymous mutations from high-throughput immunoglobulin sequencing data. Front Immunol. 2013 4:358.
5. Cui A, Di Niro R, Vander Heiden J, Briggs A, Adams K, Gilbert T, O'Connor K, Vigneault F, Shlomchik M and Kleinstein S (2016). A Model of Somatic Hypermutation Targeting in Mice Based on High-Throughput Ig Sequencing Data. The Journal of Immunology, 197(9), 3566-3574.

---

shmulateSeq *Simulate mutations in a single sequence*

Description

Generates random mutations in a sequence iteratively using a targeting model. Targeting probabilities at each position are updated after each iteration.

Usage

```
shmulateSeq(
  sequence,
  numMutations,
  targetingModel = HH_S5F,
  start = 1,
  end = nchar(sequence),
  frequency = FALSE
)
```

Arguments

`sequence`	sequence string in which mutations are to be introduced. Accepted alphabet: `{A, T, G, C, N, .}`. Note that `-` is not accepted.
`numMutations`	a whole number indicating the number of mutations to be introduced into `sequence`, if `frequency=FALSE`. A fraction between 0 and 1 indicating the mutation frequency if `frequency=TRUE`.
`targetingModel`	5-mer TargetingModel object to be used for computing probabilities of mutations at each position. Defaults to HH_S5F.
`start`	Initial position in `sequence` where mutations can be introduced. Default: 1
`end`	Last position in `sequence` where mutations can be introduced. Default: last position (sequence length).
`frequency`	If `TRUE`, treat `numMutations` as a frequency.

Details

If the input `sequence` has a non-triplet overhang at the end, it will be trimmed to the last codon. For example, `ATGCATGC` will be trimmed to `ATGCAT`.

Mutations are not introduced to positions in the input `sequence` that contain `.` or `N`.

With `frequency=TRUE`, the number of mutations is calculated according to the probability of mutation at each position. For example, if `numMutations=0.05` and the length of the input `sequence` is 100, then the number of mutations will be sampled from a binomial distribution with 100 trials and a probability of 0.05.

Value

A string defining the mutated sequence.

See Also

See shmulateTree for imposing mutations on a lineage tree. See HH_S5F and MK_RS5NF for predefined TargetingModel objects.

Examples

```
# Define example input sequence
sequence <- "NGATCTGACGACACGGCCGTGTATTACTGTGCGAGAGATA.TTTA"

# Simulate using the default human 5-mer targeting model
# Introduce 6 mutations
shmulateSeq(sequence, numMutations=6, frequency=FALSE)

# Introduction 5% mutations
shmulateSeq(sequence, numMutations=0.05, frequency=TRUE)
```

shmulateTree *Simulate mutations in a lineage tree*

Description

`shmulateTree` returns a set of simulated sequences generated from an input sequence and a lineage tree. The input sequence is used to replace the most recent common ancestor (MRCA) node of the `igraph` object defining the lineage tree. Sequences are then simulated with mutations corresponding to edge weights in the tree. Sequences will not be generated for groups of nodes that are specified to be excluded.

Usage

```
shmulateTree(
  sequence,
  graph,
  targetingModel = HH_S5F,
  field = NULL,
  exclude = NULL,
  junctionWeight = NULL,
  start = 1,
  end = nchar(sequence)
)
```

Arguments

`sequence`	string defining the MRCA sequence to seed mutations from.
`graph`	`igraph` object defining the seed lineage tree, with vertex annotations, whose edges are to be recreated.
`targetingModel`	5-mer TargetingModel object to be used for computing probabilities of mutations at each position. Defaults to HH_S5F.
`field`	annotation to use for both unweighted path length exclusion and consideration as the MRCA node. If `NULL` do not exclude any nodes.
`exclude`	vector of annotation values in `field` to exclude from potential MRCA set. If `NULL` do not exclude any nodes. Has no effect if `field=NULL`.
`junctionWeight`	fraction of the nucleotide sequence that is within the junction region. When specified this adds a proportional number of mutations to the immediate offspring nodes of the MRCA. Requires a value between 0 and 1. If `NULL` then edge weights are unmodified from the input `graph`.
`start`	Initial position in `sequence` where mutations can be introduced. Default: 1
`end`	Last position in `sequence` where mutations can be introduced. Default: last position (sequence length).

Value

A `data.frame` of simulated sequences with columns:

- `name`: name of the corresponding node in the input `graph`.
- `sequence`: mutated sequence.
- `distance`: Hamming distance of the mutated sequence from the seed `sequence`.

See Also

See shmulateSeq for imposing mutations on a single sequence. See HH_S5F and MK_RS5NF for predefined TargetingModel objects.

Examples

```
# Load example lineage and define example MRCA
data(ExampleTrees, package="alakazam")
graph <- ExampleTrees[[17]]
sequence <- "NGATCTGACGACACGGCCGTGTATTACTGTGCGAGAGATAGTTTA"

# Simulate using the default human 5-mer targeting model
shmulateTree(sequence, graph)

# Simulate using the mouse 5-mer targeting model
# Exclude nodes without a sample identifier
# Add 20% mutation rate to the immediate offsprings of the MRCA
shmulateTree(sequence, graph, targetingModel=MK_RS5NF,
             field="sample_id", exclude=NA, junctionWeight=0.2)
```

`slideWindowDb` *Sliding window approach towards filtering sequences in a* `data.frame`

Description

`slideWindowDb` determines whether each input sequence in a `data.frame` contains equal to or more than a given number of mutations in a given length of consecutive nucleotides (a "window") when compared to their respective germline sequence.

Usage

```
slideWindowDb(
  db,
  sequenceColumn = "sequence_alignment",
  germlineColumn = "germline_alignment_d_mask",
  mutThresh = 6,
  windowSize = 10,
  nproc = 1
)
```

Arguments

db	`data.frame` containing sequence data.
sequenceColumn	name of the column containing IMGT-gapped sample sequences.
germlineColumn	name of the column containing IMGT-gapped germline sequences.
mutThresh	threshold on the number of mutations in `windowSize` consecutive nucleotides. Must be between 1 and `windowSize` inclusive.
windowSize	length of consecutive nucleotides. Must be at least 2.
nproc	Number of cores to distribute the operation over. If the `cluster` has already been set earlier, then pass the `cluster`. This will ensure that it is not reset.

Value

a logical vector. The length of the vector matches the number of input sequences in `db`. Each entry in the vector indicates whether the corresponding input sequence should be filtered based on the given parameters.

See Also

See slideWindowSeq for applying the sliding window approach on a single sequence. See slideWindowTune for parameter tuning for `mutThresh` and `windowSize`.

Examples

```
# Use an entry in the example data for input and germline sequence
data(ExampleDb, package="alakazam")

# Apply the sliding window approach on a subset of ExampleDb
slideWindowDb(db=ExampleDb[1:10, ], sequenceColumn="sequence_alignment",
              germlineColumn="germline_alignment_d_mask",
              mutThresh=6, windowSize=10, nproc=1)
```

---

slideWindowSeq *Sliding window approach towards filtering a single sequence*

Description

`slideWindowSeq` determines whether an input sequence contains equal to or more than a given number of mutations in a given length of consecutive nucleotides (a "window") when compared to a germline sequence.

Usage

```
slideWindowSeq(inputSeq, germlineSeq, mutThresh, windowSize)
```

Arguments

inputSeq	input sequence.
germlineSeq	germline sequence.
mutThresh	threshold on the number of mutations in windowSize consecutive nucleotides. Must be between 1 and windowSize inclusive.
windowSize	length of consecutive nucleotides. Must be at least 2.

Value

TRUE if there are equal to or more than mutThresh number of mutations in any window of windowSize consecutive nucleotides (i.e. the sequence should be filtered); FALSE if otherwise.

See Also

calcObservedMutations is called by slideWindowSeq to identify observed mutations. See slideWindowDb for applying the sliding window approach on a data.frame. See slideWindowTune for parameter tuning for mutThresh and windowSize.

Examples

```
# Use an entry in the example data for input and germline sequence
data(ExampleDb, package="alakazam")
in_seq <- ExampleDb[["sequence_alignment"]][100]
germ_seq <-  ExampleDb[["germline_alignment_d_mask"]][100]

# Determine if in_seq has 6 or more mutations in 10 consecutive nucleotides
slideWindowSeq(inputSeq=in_seq, germlineSeq=germ_seq, mutThresh=6, windowSize=10)
slideWindowSeq(inputSeq="TCGTCGAAAA", germlineSeq="AAAAAAAAAA", mutThresh=6, windowSize=10)
```

---

slideWindowTune *Parameter tuning for sliding window approach*

Description

Apply slideWindowDb over a search grid made of combinations of mutThresh and windowSize to help with picking a pair of values for these parameters. Parameter tuning can be performed by choosing a combination that gives a reasonable number of filtered/remaining sequences.

Usage

```
slideWindowTune(
  db,
  sequenceColumn = "sequence_alignment",
  germlineColumn = "germline_alignment_d_mask",
  dbMutList = NULL,
  mutThreshRange,
  windowSizeRange,
  verbose = TRUE,
  nproc = 1
)
```

Arguments

db	`data.frame` containing sequence data.
sequenceColumn	name of the column containing IMGT-gapped sample sequences.
germlineColumn	name of the column containing IMGT-gapped germline sequences.
dbMutList	if supplied, this should be a list consisting of `data.frames` returned as `$pos` in the nested list produced by calcObservedMutations with `returnRaw=TRUE`; otherwise, calcObservedMutations is called on columns `sequenceColumn` and `germlineColumn` of `db`. Default is `NULL`.
mutThreshRange	range of threshold on the number of mutations in `windowSize` consecutive nucleotides to try. Must be between 1 and maximum `windowSizeRange` inclusive.
windowSizeRange	range of length of consecutive nucleotides to try. The lower end must be at least 2.
verbose	whether to print out messages indicating current progress. Default is `TRUE`.
nproc	Number of cores to distribute the operation over. If the `cluster` has already been set earlier, then pass the `cluster`. This will ensure that it is not reset.

Details

If, in a given combination of `mutThresh` and `windowSize`, `mutThresh` is greater than `windowSize`, `NA`s will be returned for that particular combination. A message indicating that the combination has been "skipped" will be printed if `verbose=TRUE`.

If calcObservedMutations was previously run on `db` and saved, supplying `$pos` from the saved result as `dbMutList` could save time by skipping a second call of calcObservedMutations. This could be helpful especially when `db` is large.

Value

a list of logical matrices. Each matrix corresponds to a `windowSize` in `windowSizeRange`. Each column in a matrix corresponds to a `mutThresh` in `mutThreshRange`. Each row corresponds to a sequence. `TRUE` values mean the sequences has at least the number of mutations specified in the column name, for that `windowSize`.

See Also

slideWindowDb is called on `db` for tuning. See slideWindowTunePlot for visualization. See calcObservedMutations for generating `dbMutList`.

Examples

```
# Load and subset example data
data(ExampleDb, package="alakazam")
db <- ExampleDb[1:5, ]

# Try out thresholds of 2-4 mutations in window sizes of 7-9 nucleotides.
# In this case, all combinations are legal.
slideWindowTune(db, mutThreshRange=2:4, windowSizeRange=7:9)

# Illegal combinations are skipped, returning NAs.
slideWindowTune(db, mutThreshRange=2:4, windowSizeRange=2:4,
                verbose=FALSE)

# Run calcObservedMutations separately
exDbMutList <- sapply(1:5, function(i) {
    calcObservedMutations(inputSeq=db[["sequence_alignment"]][i],
                          germlineSeq=db[["germline_alignment_d_mask"]][i],
                          returnRaw=TRUE)$pos })
slideWindowTune(db, dbMutList=exDbMutList,
                mutThreshRange=2:4, windowSizeRange=2:4)
```

---

slideWindowTunePlot *slideWindowTunePlot - plotSlideWindowTune backward compatibility*

Description

Wrapper function for plotSlideWindowTune

Usage

```
slideWindowTunePlot(
  tuneList,
  plotFiltered = c(TRUE, FALSE, NULL, "filtered", "remaining", "per_mutation"),
  percentage = FALSE,
  jitter.x = FALSE,
  jitter.x.amt = 0.1,
  jitter.y = FALSE,
  jitter.y.amt = 0.1,
  pchs = 1,
  ltys = 2,
  cols = 1,
  plotLegend = TRUE,
  legendPos = "topright",
  legendHoriz = FALSE,
  legendCex = 1,
  title = NULL,
  returnRaw = FALSE
)
```

Arguments

`tuneList`	a list of logical matrices returned by slideWindowTune.
`plotFiltered`	whether to plot the number of filtered (`TRUE` or `filtered`), or remaining (FALSE or remaining) sequences for each mutation threshold. Use `NULL` or `per_mutation` to plot the number of sequences at each mutation value. Default is `TRUE`.
`percentage`	whether to plot on the y-axis the percentage of filtered sequences (as opposed to the absolute number). Default is `FALSE`.
`jitter.x`	whether to jitter x-axis values. Default is `FALSE`.
`jitter.x.amt`	amount of jittering to be applied on x-axis values if `jitter.x=TRUE`. Default is 0.1.
`jitter.y`	whether to jitter y-axis values. Default is `FALSE`.
`jitter.y.amt`	amount of jittering to be applied on y-axis values if `jitter.y=TRUE`. Default is 0.1.
`pchs`	point types to pass on to plot.
`ltys`	line types to pass on to plot.
`cols`	colors to pass on to plot.
`plotLegend`	whether to plot legend. Default is `TRUE`.
`legendPos`	position of legend to pass on to legend. Can be either a numeric vector specifying x-y coordinates, or one of `"topright"`, `"center"`, etc. Default is `"topright"`.
`legendHoriz`	whether to make legend horizontal. Default is `FALSE`.
`legendCex`	numeric values by which legend should be magnified relative to 1.
`title`	plot main title. Default is NULL (no title)
`returnRaw`	Return a data.frame with sequence counts (TRUE) or a plot. Default is `FALSE`.

Details

For each `windowSize`, if `plotFiltered=TRUE`, the x-axis represents a mutation threshold range, and the y-axis the number of sequences that have at least that number of mutations. If `plotFiltered=TRUE`, the y-axis represents the number of sequences that have less mutations than the mutation threshold range. For the same window size, a sequence can be included in the counts for different mutation thresholds. For example, sequence "CCACCAAAA" with germline "AAAAAAAAA" has 4 mutations. This sequence has at least 2 mutations and at least 3 mutations, in a window of size 4. the sequence will be included in the sequence count for mutation thresholds 2 and 3. If `plotFiltered=TRUE`, the sequences are counted only once for each window size, at their largest mutation threshold. The above example sequence would be included in the sequence count for mutation threshold 3.

When plotting, a user-defined `amount` of jittering can be applied on values plotted on either axis or both axes via adjusting `jitter.x`, `jitter.y`, `jitter.x.amt` and `jitter.y.amt`. This may be help with visually distinguishing lines for different window sizes in case they are very close or identical to each other. If plotting percentages (`percentage=TRUE`) and using jittering on the y-axis values (`jitter.y=TRUE`), it is strongly recommended that `jitter.y.amt` be set very small (e.g. 0.01).

`NA` for a combination of `mutThresh` and `windowSize` where `mutThresh` is greater than `windowSize` will not be plotted.

See Also

See slideWindowTune for how to get `tuneList`. See jitter for use of `amount` of jittering.

Examples

```
# Use an entry in the example data for input and germline sequence
data(ExampleDb, package="alakazam")

# Try out thresholds of 2-4 mutations in window sizes of 3-5 nucleotides
# on a subset of ExampleDb
tuneList <- slideWindowTune(db = ExampleDb[1:10, ],
                           mutThreshRange = 2:4, windowSizeRange = 3:5,
                           verbose = FALSE)

# Visualize
# Plot numbers of sequences filtered without jittering y-axis values
slideWindowTunePlot(tuneList, pchs=1:3, ltys=1:3, cols=1:3,
                    plotFiltered=TRUE, jitter.y=FALSE)

# Notice that some of the lines overlap
# Jittering could help
slideWindowTunePlot(tuneList, pchs=1:3, ltys=1:3, cols=1:3,
                    plotFiltered=TRUE, jitter.y=TRUE)

# Plot numbers of sequences remaining instead of filtered
slideWindowTunePlot(tuneList, pchs=1:3, ltys=1:3, cols=1:3,
                    plotFiltered=FALSE, jitter.y=TRUE,
                    legendPos="bottomright")

# Plot percentages of sequences filtered with a tiny amount of jittering
slideWindowTunePlot(tuneList, pchs=1:3, ltys=1:3, cols=1:3,
                    plotFiltered=TRUE, percentage=TRUE,
                    jitter.y=TRUE, jitter.y.amt=0.01)
```

`summarizeBaseline` *Calculate BASELINe summary statistics*

Description

`summarizeBaseline` calculates BASELINe statistics such as the mean selection strength (mean Sigma), the 95% confidence intervals and p-values for the presence of selection.

Usage

```
summarizeBaseline(baseline, returnType = c("baseline", "df"), nproc = 1)
```

Arguments

baseline	`Baseline` object returned by calcBaseline containing annotations and BASELINe posterior probability density functions (PDFs) for each sequence.
returnType	One of `c("baseline", "df")` defining whether to return a `Baseline` object ("baseline") with an updated `stats` slot or a data.frame ("df") of summary statistics.
nproc	number of cores to distribute the operation over. If `nproc` = 0 then the `cluster` has already been set and will not be reset.

Details

The returned p-value can be either positive or negative. Its magnitude (without the sign) should be interpreted as per normal. Its sign indicates the direction of the selection detected. A positive p-value indicates positive selection, whereas a negative p-value indicates negative selection.

Value

Either a modified `Baseline` object or data.frame containing the mean BASELINe selection strength, its 95% confidence intervals, and a p-value for the presence of selection.

References

1. Uduman M, et al. Detecting selection in immunoglobulin sequences. Nucleic Acids Res. 2011 39(Web Server issue):W499-504.

See Also

See calcBaseline for generating `Baseline` objects and groupBaseline for convolving groups of BASELINe PDFs.

Examples

```
# Subset example data
data(ExampleDb, package="alakazam")
db <- subset(ExampleDb, c_call == "IGHG")
set.seed(112)
db <- dplyr::slice_sample(db, n=100)

# Collapse clones
db <- collapseClones(db, cloneColumn="clone_id",
                     sequenceColumn="sequence_alignment",
                     germlineColumn="germline_alignment_d_mask",
                     method="thresholdedFreq", minimumFrequency=0.6,
                     includeAmbiguous=FALSE, breakTiesStochastic=FALSE)

# Calculate BASELINe
baseline <- calcBaseline(db,
                         sequenceColumn="clonal_sequence",
                         germlineColumn="clonal_germline",
                         testStatistic="focused",
                         regionDefinition=IMGT_V,
                         targetingModel=HH_S5F,
                         nproc = 1)

# Grouping the PDFs by the sample annotation
grouped <- groupBaseline(baseline, groupBy="sample_id")

# Get a data.frame of the summary statistics
stats <- summarizeBaseline(grouped, returnType="df")
```

`TargetingMatrix-class` *S4 class defining a targeting matrix*

Description

`TargetingMatrix` defines a data structure for just the targeting matrix (as opposed to the entire `TargetingModel`)

Slots

`.Data` matrix.

`numMutS` number indicating the number of silent mutations used for estimating mutability.

`numMutR` number indicating the number of replacement mutations used for estimating mutability.

`TargetingModel-class` *S4 class defining a targeting model*

Description

`TargetingModel` defines a common data structure for mutability, substitution and targeting of immunoglobulin (Ig) sequencing data in a 5-mer microsequence context.

Usage

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

Arguments

`x`	`TargetingModel` object.
`y`	ignored.
`...`	arguments to pass to plotMutability.

Slots

`name` Name of the model.

`description` Description of the model and its source data.

`species` Genus and species of the source sequencing data.

`date` Date the model was built.

`citation` Publication source.

`substitution` Normalized rates of the center nucleotide of a given 5-mer mutating to a different nucleotide. The substitution model is stored as a 5x3125 matrix of rates. Rows define the mutated nucleotide at the center of each 5-mer, one of `c("A", "C", "G", "T", "N")`, and columns define the complete 5-mer of the unmutated nucleotide sequence.

`mutability` Normalized rates of a given 5-mer being mutated. The mutability model is stored as a numeric vector of length 3125 with mutability rates for each 5-mer. Note that "normalized" means that the mutability rates for the 1024 5-mers that contain no "N" at any position sums up to 1 (as opposed to the entire vector summing up to 1).

`targeting` Rate matrix of a given mutation occurring, defined as $mutability * substitution$. The targeting model is stored as a 5x3125 matrix. Rows define the mutated nucleotide at the center of each 5-mer, one of `c("A", "C", "G", "T", "N")`, and columns define the complete 5-mer of the unmutated nucleotide sequence.

`numMutS` number indicating the number of silent mutations used for estimating mutability.

`numMutR` number indicating the number of replacement mutations used for estimating mutability.

See Also

See createTargetingModel building models from sequencing data.

`testBaseline` *Two-sided test of BASELINe PDFs*

Description

`testBaseline` performs a two-sample significance test of BASELINe posterior probability density functions (PDFs).

Usage

```
testBaseline(baseline, groupBy)
```

Arguments

`baseline`	`Baseline` object containing the `db` and grouped BASELINe PDFs returned by groupBaseline.
`groupBy`	string defining the column in the `db` slot of the `Baseline` containing sequence or group identifiers.

Value

A data.frame with test results containing the following columns:

- `region`: sequence region, such as `cdr` and `fwr`.
- `test`: string defining the groups be compared. The string is formatted as the conclusion associated with the p-value in the form `GROUP1 != GROUP2`. Meaning, the p-value for rejection of the null hypothesis that GROUP1 and GROUP2 have equivalent distributions.
- `pvalue`: two-sided p-value for the comparison.
- `fdr`: FDR corrected `pvalue`.

References

1. Yaari G, et al. Quantifying selection in high-throughput immunoglobulin sequencing data sets. Nucleic Acids Res. 2012 40(17):e134. (Corrections at http://selection.med.yale.edu/baseline/correction/)

See Also

To generate the Baseline input object see groupBaseline.

Examples

```
# Subset example data as a demo
data(ExampleDb, package="alakazam")
db <- subset(ExampleDb, c_call %in% c("IGHM", "IGHG"))
set.seed(112)
db <- dplyr::slice_sample(db, n=200)

# Collapse clones
db <- collapseClones(db, cloneColumn="clone_id",
                     sequenceColumn="sequence_alignment",
                     germlineColumn="germline_alignment_d_mask",
                     method="thresholdedFreq", minimumFrequency=0.6,
                     includeAmbiguous=FALSE, breakTiesStochastic=FALSE)

# Calculate BASELINe
baseline <- calcBaseline(db,
                         sequenceColumn="clonal_sequence",
                         germlineColumn="clonal_germline",
                         testStatistic="focused",
                         regionDefinition=IMGT_V,
                         targetingModel=HH_S5F,
                         nproc=1)

# Group PDFs by the isotype
grouped <- groupBaseline(baseline, groupBy="c_call")

# Visualize isotype PDFs
plot(grouped, "c_call")

# Perform test on isotype PDFs
```

```
testBaseline(grouped, groupBy="c_call")
```

U5N — *Uniform 5-mer null targeting model.*

Description

A null 5-mer model of somatic hypermutation targeting where all substitution, mutability and targeting rates are uniformly distributed.

Usage

```
U5N
```

Format

A TargetingModel object.

See Also

See HH_S5F and HKL_S5F for the human 5-mer targeting models; and MK_RS5NF for the mouse 5-mer targeting model.

writeTargetingDistance — *Write targeting model distances to a file*

Description

`writeTargetingDistance` writes a 5-mer targeting distance matrix to a tab-delimited file.

Usage

```
writeTargetingDistance(model, file)
```

Arguments

`model`	TargetingModel object with mutation likelihood information.
`file`	name of file to write.

Details

The targeting distance write as a tab-delimited 5x3125 matrix. Rows define the mutated nucleotide at the center of each 5-mer, one of `c("A", "C", "G", "T", "N")`, and columns define the complete 5-mer of the unmutated nucleotide sequence. `NA` values in the distance matrix are replaced with distance 0.

See Also

Takes as input a TargetingModel object and calculates distances using calcTargetingDistance.

Examples

```
## Not run:
# Write HS5F targeting model to working directory as hs5f.tab
writeTargetingDistance(HH_S5F, "hh_s5f.tsv")

## End(Not run)
```

Index