Package 'MiDAS'

September 30, 2019

Title R package for immunogenomics data handling and association analysis

Version 0.0.0.9033

Description What the package does (one paragraph).

License What license is it under?

Encoding UTF-8

LazyData true

Imports stringi (>= 1.2.4), assertthat (>= 0.2.0), stats, utils, qdapTools (>= 1.3.3), broom (>= 0.5.1), dplyr (>= 0.8.0.1), purrr (>= 0.3.0), MASS (>= 7.3-51.1), rlang (>= 0.3.1), uniqtag (>= 1.0), kableExtra (>= 1.1.0), knitr (>= 1.21), magrittr (>= 1.5), tidyr (>= 0.8.2), formattable (>= 0.2.0.1), Hmisc (>= 4.2-0), methods

Suggests testthat (>= 2.0.1), seqinr (>= 3.4-5), survival (>= 2.43-3), vcfR (>= 1.8.0), ensembldb (>= 2.6.8), EnsDb.Hsapiens.v86 (>= 2.99.0), broom.mixed (>= 0.2.4)

RoxygenNote 6.1.1

R topics documented:

`aaVariationToCounts` *Transform amino acid variations data frame to counts table*

Description

`aaVariationToCounts` converts amino acid variations data frame into counts table.

Usage

```
aaVariationToCounts(aa_variation, inheritance_model = c("dominant",
  "recessive", "additive"))
```

Arguments

`aa_variation`	Data frame holding amino acid variation data as returned by hlaToAAVariation.
`inheritance_model`	String specifying inheritance model to use. Available choices are `"dominant"`, `"recessive"`, `"additive"`. In `"dominant"` model homozygotes and heterozygotes are coded as 1. In `"recessive"` model homozygotes are coded as 1 and all other as `0`. In `"additive"` model homozygotes are coded as 2 and heterozygotes as 1.

Value

Data frame containing counts of amino acid at specific positions according to inheritance specified model.

See Also

`hlaToAAVariation`

Examples

```
file <- system.file("extdata", "HLAHD_output_example.txt", package = "MiDAS")
hla_calls <- readHlaCalls(file)
aa_variation <- hlaToAAVariation(hla_calls)
aaVariationToCounts(aa_variation, inheritance_model = "additive")
```

`analyzeAssociations` *Association analysis*

Description

`analyzeAssociations` perform association analysis on single variable level using statistical model of choice.

Usage

```
analyzeAssociations(object, variables, correction = "bonferroni",
  n_correction = NULL, exponentiate = FALSE)
```

Arguments

`object`	An existing fit from a model function such as lm, glm and many others.
`variables`	Character vector specifying variables to use in association tests.
`correction`	String specifying multiple testing correction method. See details for further information.
`n_correction`	Integer specifying number of comparisons to consider during multiple testing correction calculations, must be at least equal to number of comparisons being made; only set this (to non-default) when you know what you are doing!
`exponentiate`	Logical flag indicating whether or not to exponentiate the coefficient estimates. Internally this is passed to `tidy`. This is typical for logistic and multinomial regressions, but a bad idea if there is no log or logit link. Defaults to FALSE.

Details

`correction` specifies p-value adjustment method to use, common choice is Benjamini & Hochberg (1995) (`"BH"`). Internally this is passed to p.adjust.

Value

Tibble containing combined results for all alleles in `hla_calls`.

See Also

p.adjust, tidy

Other MiDAS statistical functions: analyzeConditionalAssociations, prepareMiDAS, runMiDAS

Examples

```
library("survival")
hla_calls_file <- system.file("extdata", "HLAHD_output_example.txt", package = "MiDAS")
hla_calls <- readHlaCalls(hla_calls_file)
pheno_file <- system.file("extdata", "pheno_example.txt", package = "MiDAS")
pheno <- read.table(pheno_file, header = TRUE)
covar_file <- system.file("extdata", "covar_example.txt", package = "MiDAS")
covar <- read.table(covar_file, header = TRUE)
midas_data <- prepareMiDAS(hla_calls = hla_calls,
                               pheno = pheno,
                               covar = covar,
                               analysis_type = "hla_allele",
                               inheritance_model = "additive"
)

# Cox proportional hazards regression model
## define base model with response and covariates
object <- coxph(Surv(OS, OS_DIED) ~ AGE + SEX, data = midas_data)

## test for alleles associations
analyzeAssociations(object = object,
                    variables = c("B*14:02", "DRB1*11:01")
)
```

---

analyzeConditionalAssociations

Stepwise conditional association analysis

Description

analyzeConditionalAssociations perform stepwise conditional testing adding the previous top-associated variable as covariate, until there is no more significant variables based on a self-defined threshold.

Usage

```
analyzeConditionalAssociations(object, variables,
  correction = "bonferroni", n_correction = NULL, th, keep = FALSE,
  rss_th = 1e-07, exponentiate = FALSE)
```

Arguments

object	An existing fit from a model function such as lm, glm and many others.
variables	Character vector specifying variables to use in association tests.
correction	String specifying multiple testing correction method. See details for further information.

`n_correction`	Integer specifying number of comparisons to consider during multiple testing correction calculations, must be at least equal to number of comparisons being made; only set this (to non-default) when you know what you are doing!
`th`	Number specifying p-value threshold for a variable to be considered significant.
`keep`	Logical flag indicating if the output should be a list of results resulting from each selection step. Default is to return only the final result.
`rss_th`	Number specifying residual sum of squares threshold at which function should stop adding additional variables. As the residual sum of squares approaches `0` the perfect fit is obtained making further attempts at variables selection nonsense, thus function is stopped. This behavior can be controlled using `rss_th`.
`exponentiate`	Logical flag indicating whether or not to exponentiate the coefficient estimates. Internally this is passed to `tidy`. This is typical for logistic and multinomial regressions, but a bad idea if there is no log or logit link. Defaults to FALSE.

Details

Selection criteria is the p-value from the test on coefficients values.

Value

Tibble with stepwise conditional testing results.

See Also

`p.adjust`, `tidy`

Other MiDAS statistical functions: `analyzeAssociations`, `prepareMiDAS`, `runMiDAS`

Examples

```
library("survival")
hla_calls_file <- system.file("extdata", "HLAHD_output_example.txt", package = "MiDAS")
hla_calls <- readHlaCalls(hla_calls_file)
pheno_file <- system.file("extdata", "pheno_example.txt", package = "MiDAS")
pheno <- read.table(pheno_file, header = TRUE, stringsAsFactors = FALSE)
covar_file <- system.file("extdata", "covar_example.txt", package = "MiDAS")
covar <- read.table(covar_file, header = TRUE, stringsAsFactors = FALSE)
midas_data <- prepareMiDAS(hla_calls = hla_calls,
                             pheno = pheno,
                             covar = covar,
                             analysis_type = "hla_allele",
                             inheritance_model = "additive"
)

## define base model with covariates only
object <- coxph(Surv(OS, OS_DIED) ~ AGE + SEX, data = midas_data)
analyzeConditionalAssociations(object,
                            variables = c("B*14:02", "DRB1*11:01"),
                            th = 0.05,
                            rss_th = 1e-07
)
```

backquote *Backquote string*

Description

backquote places backticks around string.

Usage

```
backquote(x)
```

Arguments

x	Character vector.

Details

backquote is useful when using HLA allele numbers in formulas, where '*' and ':' characters have special meanings.

Value

Character vector with its elements backticked.

Examples

```
backquote("A*01:01")
```

characterMatches *Check if character matches one of possible values*

Description

characterMatches checks if all elements of a character vector matches values in choices.

Usage

```
characterMatches(x, choice)
```

Arguments

x	Character vector to test.
choice	Character vector with possible values for x.

Value

Logical indicating if x's elements matches any of the values in choice.

See Also

Other assert functions: `checkAdditionalData`, `checkHlaCallsFormat`, `checkKirCountsFormat`, `checkStatisticalModel`, `colnamesMatches`, `hasTidyMethod`, `isCharacterOrNULL`, `isClassOrNULL`, `isCountOrNULL`, `isCountsOrZeros`, `isFlagOrNULL`, `isNumberOrNULL`, `isStringOrNULL`, `isTRUEorFALSE`, `stringMatches`

`checkAdditionalData` *Assert additional data*

Description

`checkAdditionalData` asserts if phenotype or covariate data frame has proper format.

Usage

```
checkAdditionalData(data_frame, hla_calls, accept.null = FALSE)
```

Arguments

`data_frame`	Data frame containing phenotype or covariate data corresponding to accompanying hla calls data frame.
`hla_calls`	Data frame containing HLA allele calls, as return by `readHlaCalls` function.
`accept.null`	Logical indicating if NULL data_frame should be accepted.

Value

Logical indicating if `data_frame` is properly formatted. Otherwise raise error.

See Also

Other assert functions: `characterMatches`, `checkHlaCallsFormat`, `checkKirCountsFormat`, `checkStatisticalMode` `colnamesMatches`, `hasTidyMethod`, `isCharacterOrNULL`, `isClassOrNULL`, `isCountOrNULL`, `isCountsOrZeros`, `isFlagOrNULL`, `isNumberOrNULL`, `isStringOrNULL`, `isTRUEorFALSE`, `stringMatches`

Examples

```
hla_file <- system.file("extdata", "HLAHD_output_example.txt", package = "MiDAS")
pheno_file <- system.file("extdata", "pheno_example.txt", package = "MiDAS")
pheno <- read.table(pheno_file, header = TRUE)
hla_calls <- readHlaCalls(hla_file)
checkAdditionalData(pheno, hla_calls)
```

`checkAlleleFormat` *Check allele format*

Description

`checkAlleleFormat` test if the input character follows HLA nomenclature specifications.

Usage

```
checkAlleleFormat(allele)
```

Arguments

`allele`	Character vector containing HLA allele numbers.

Details

Correct HLA number should consist of HLA gene name followed by "*" and sets of digits separated with ":". Maximum number of sets of digits is 4 which is termed 8-digit resolution. Optionally HLA numbers can be supplemented with additional suffix indicating its expression status. See `http://hla.alleles.org/nomenclature/naming.html` for more details.

HLA alleles with identical sequences across exons encoding the peptide binding domains might be designated with G group allele numbers. Those numbers have additional G or GG suffix. See `http://hla.alleles.org/alleles/g_groups.html` for more details.

Value

Logical vector specifying if `allele` follows HLA alleles naming conventions.

Examples

```
allele <- c("A*01:01", "A*01:02")
checkAlleleFormat(allele)
```

`checkHlaCallsFormat` *Assert hla calls data frame format*

Description

`checkHlaCallsFormat` asserts if hla calls data frame have proper format.

Usage

```
checkHlaCallsFormat(hla_calls)
```

Arguments

`hla_calls`	Data frame containing HLA allele calls, as return by `readHlaCalls` function.

Value

Logical indicating if `hla_calls` follows hla calls data frame format. Otherwise raise error.

See Also

Other assert functions: `characterMatches`, `checkAdditionalData`, `checkKirCountsFormat`, `checkStatisticalMode` `colnamesMatches`, `hasTidyMethod`, `isCharacterOrNULL`, `isClassOrNULL`, `isCountOrNULL`, `isCountsOrZeros`, `isFlagOrNULL`, `isNumberOrNULL`, `isStringOrNULL`, `isTRUEorFALSE`, `stringMatches`

Examples

```
file <- system.file("extdata", "HLAHD_output_example.txt", package = "MiDAS")
hla_calls <- readHlaCalls(file)
checkHlaCallsFormat(hla_calls)
```

`checkKirCountsFormat` *Assert KIR counts data frame format*

Description

`checkKirCountsFormat` asserts if KIR counts data frame have proper format.

Usage

```
checkKirCountsFormat(kir_counts, accept.null = FALSE)
```

Arguments

`kir_counts`	Data frame containing KIR gene counts, as returned by `readKirCalls` function.
`accept.null`	Logical indicating if NULL `kir_counts` should be accepted.

Value

Logical indicating if `kir_counts` follow KIR counts data frame format. Otherwise raise error.

See Also

`readKirCalls`, `getHlaKirInteractions`, `kirHaplotypeToCounts`, `prepareMiDAS`.

Other assert functions: `characterMatches`, `checkAdditionalData`, `checkHlaCallsFormat`, `checkStatisticalModel` `colnamesMatches`, `hasTidyMethod`, `isCharacterOrNULL`, `isClassOrNULL`, `isCountOrNULL`, `isCountsOrZeros`, `isFlagOrNULL`, `isNumberOrNULL`, `isStringOrNULL`, `isTRUEorFALSE`, `stringMatches`

Examples

```
file <- system.file("extdata", "KIP_output_example.txt", package = "MiDAS")
kir_counts <- readKirCalls(file)
checkKirCountsFormat(kir_counts)
```

checkStatisticalModel *Assert statistical model*

Description

checkStatisticalModel asserts if object is an existing fit from a model function such as lm, glm and many others.

Usage

```
checkStatisticalModel(object)
```

Arguments

object	An existing fit from a model function such as lm, glm and many others.

Value

Logical indicating if object is an existing fit from a model function such as lm, glm and many others. Otherwise raise error.

See Also

Other assert functions: characterMatches, checkAdditionalData, checkHlaCallsFormat, checkKirCountsFormat, colnamesMatches, hasTidyMethod, isCharacterOrNULL, isClassOrNULL, isCountOrNULL, isCountsOrZeros, isFlagOrNULL, isNumberOrNULL, isStringOrNULL, isTRUEorFALSE, stringMatches

Examples

```
object <- lm(dist ~ speed, data = cars)
checkStatisticalModel(object)
```

colnamesMatches *Check column names*

Description

colnamesMatches check if data frame's columns are named as specified

Usage

```
colnamesMatches(x, cols)
```

Arguments

x	Data frame to test.
cols	Ordered character vector to test against x's colnames.

Value

Logical indicating if `x`'s colnames equals `choice`.

See Also

Other assert functions: `characterMatches`, `checkAdditionalData`, `checkHlaCallsFormat`, `checkKirCountsFormat`, `checkStatisticalModel`, `hasTidyMethod`, `isCharacterOrNULL`, `isClassOrNULL`, `isCountOrNULL`, `isCountsOrZeros`, `isFlagOrNULL`, `isNumberOrNULL`, `isStringOrNULL`, `isTRUEorFALSE`, `stringMatches`

`convertAlleleToVariable` *Converts allele numbers to additional variables*

Description

`convertAlleleToVariable` convert input HLA allele numbers to additional variables based on the supplied match table (dictionary).

Usage

```
convertAlleleToVariable(allele, dictionary)
```

Arguments

`allele`	Character vector containing HLA allele numbers.
`dictionary`	Path to the file containing HLA allele numbers matchings or data frame providing this information. See details for further explanations.

Details

`dictionary` file should be a tsv format with header and two columns. First column should hold allele numbers and second corresponding additional variables. Optionally a data frame formatted in the same manner can be passed instead.

Value

Vector containing HLA allele numbers converted to additional variables according to matching file.

Type of the returned vector depends on the type of the additional variable. For example for HLA alleles supertypes character vector is returned while for expression values numeric vector will be returned.

Examples

```
dictionary <- system.file("extdata", "Match_allele_HLA_supertype.txt", package = "MiDAS")
convertAlleleToVariable(c("A*01:01", "A*02:01"), dictionary = dictionary)
```

countsToHlaCalls *Convert HLA counts table to HLA calls*

Description

countsToHlaCalls convert counts table to HLA calls data frame, this is useful when working with data from UK Biobank.

Usage

```
countsToHlaCalls(counts)
```

Arguments

counts	Data frame with HLA alleles counts, as returned by hlaCallsToCounts function. First column should contain samples IDs, following columns should be named with valid HLA alleles numbers.

Details

Note that proper HLA calls reconstruction from counts table is only possible under additive inheritance model. This mode of operation is the only one implemented so the function will always treat counts table as coming from `hlaCallsToCounts(hla_calls, inheritance_model = 'additive')`.

Value

Data frame containing HLA allele calls.

Examples

```
file <- system.file("extdata", "HLAHD_output_example.txt", package = "MiDAS")
hla_calls <- readHlaCalls(file)
hla_counts <- hlaCallsToCounts(hla_calls, inheritance_model = "additive")
countsToHlaCalls(hla_counts)
```

countsToVariables *Convert counts data frame according to match table*

Description

countsToVariables convert counts data frame to variables based on match table (dictionary).

Usage

```
countsToVariables(counts, dictionary, na.value = NA, nacols.rm = TRUE)
```

Arguments

`counts`	Data frame with counts, such as returned by `hlaCallsToCounts` function. First column should contain samples IDs, following columns should contain counts (natural numbers including zero).
`dictionary`	Path to the file containing variables matchings or data frame providing this information. See details for further explanations.
`na.value`	Vector of length one speciyfing value for variables for which no matching is found in `counts`. Default behaviour is to mark such instances with `NA`.
`nacols.rm`	logical indicating if result columns that contain only `NA` should be removed.

Details

`dictionary` file should be a tsv format with header and two columns. First column should be named `"Name"` and hold variable name, second should be named `"Expression"` and hold expression used to identify variable (eg. `"KIR2DL3 & ! KIR2DL2"` will match all samples with `KIR2DL3` and without `KIR2DL2`). Optionally a data frame formatted in the same manner can be passed instead.

Dictionaries shipped with the package:

`hla_kir_interactions`	HLA - KIR interactions based on Pende et al., 2019.
`kir_haplotypes`	KIR genes to KIR haplotypes dictionary.

Value

Data frame of indicators for new variables, with `1` signaling presence of variable and `0` absence.

Examples

```
file <- system.file("extdata", "KIP_output_example.txt", package = "MiDAS")
kir_counts <- readKirCalls(file)
countsToVariables(kir_counts, "kir_haplotypes")
```

---

formatAssociationsResults

Pretty format association analysis results

Description

`formatAssociationsResults` formats results table to specified format. It uses `formatResults` with pre specified arguments to return pretty formatted table depending on the type of analysis and model type. This function is intended only to be used internally by `runMiDAS`.

Usage

```
formatAssociationsResults(results, type = "hla_allele",
  response_variable = "R", logistic = FALSE, pvalue_cutoff = NULL,
  format = getOption("knitr.table.format"))
```

Arguments

results	Tibble as returned by analyzeAssociations.
type	String specifying type of analysis from which results were produced. Possible values includes 'hla_allele', 'aa_level', 'expression_level', 'allele_g_group', 'allele_supertype', 'allele_group', 'kir_genes', 'hla_kir_interactions', 'custom'.
response_variable	String giving the name of response variable, it is used to produce binary phenotype column names.
logistic	Logical indicating if statistical model is logistic. If set to TRUE, estimate will be renamed to odds ratio.
pvalue_cutoff	Number specifying p-value cutoff for results to be included in output. If NULL cutoff of 0.05 on p.adjusted value is used instead.
format	String with possible values "latex" and "html".

Value

A character vector with pretty formatted results table.

See Also

formatResults, runMiDAS

---

formatResults *Helper function for pretty formating statistical analysis results*

Description

formatResults format statistical analysis results table to html or latex format.

Usage

```
formatResults(results, filter_by = "p.value <= 0.05",
  arrange_by = "p.value", select_cols = c("term", "estimate",
  "std.error", "p.value", "p.adjusted"), format = c("html", "latex"),
  header = NULL)
```

Arguments

results	Tibble as returned by analyzeAssociations.
filter_by	Character vector specifying conditional expression used to filter results, this is equivalent to ... argument passed to filter except it has to be a character vector.
arrange_by	Character vector specifying variable names to use for sorting. Equivalent to ... argument passed to arrange.
select_cols	Character vector specifying variable names that should be included in the output table. Can be also used to rename selected variables, see examples.
format	String with possible values "latex" and "html".
header	String specifying header for result table. If NULL no header is added.

Value

Character vector of formatted table source code.

See Also

runMiDAS, analyzeAssociations, analyzeConditionalAssociations.

Examples

```
hla_calls <- readHlaCalls(system.file("extdata", "HLAHD_output_example.txt", package = "MiDAS"))
hla_counts <- hlaCallsToCounts(hla_calls, inheritance_model = "additive")
midas_data <- read.table(
  system.file("extdata", "pheno_example.txt", package = "MiDAS"),
  header = TRUE)
midas_data <- dplyr::left_join(x = midas_data, y = hla_counts, by = "ID")
object <- lm(OS ~ 1, data = midas_data)
res <- analyzeAssociations(object, variables = colnames(midas_data)[-1])
formatResults(res,
              filter_by = c("p.value <= 0.05", "estimate > 0"),
              arrange_by = c("p.value * estimate"),
              select_cols = c("allele" = "term", "p.value"),
              format = "html",
              header = "HLA allelic associations")
```

---

getAAFrequencies *Calculate amino acid's frequencies*

Description

getAAFrequencies calculates amino acid's frequencies in amino acid variations data frame.

Usage

```
getAAFrequencies(aa_variation)
```

Arguments

aa_variation	Data frame holding amino acid variation data as returned by hlaToAAVariation.

Details

Amino acid's frequencies are counted in reference to sample taking both gene copies into consideration. 'n / (2 * j)' where 'n' is the number of amino acid occurrences and 'j' is the sample size.

Value

Data frame containing the amino acid's positions and their corresponding frequencies.

See Also

hlaToAAVariation

Examples

```
file <- system.file("extdata", "HLAHD_output_example.txt", package = "MiDAS")
hla_calls <- readHlaCalls(file)
aa_variation <- hlaToAAVariation(hla_calls)
getAAFrequencies(aa_variation)
```

`getAlleleResolution` *Infers HLA allele resolution*

Description

`getAlleleResolution` returns the resolution of input HLA allele numbers.

Usage

```
getAlleleResolution(allele)
```

Arguments

`allele`	Character vector containing HLA allele numbers.

Details

HLA allele resolution can take the following values: 2, 4, 6, 8.See `http://hla.alleles.org/nomenclature/naming.html` for more details.

Value

Integer vector specifying alleles resolutions.

`NA` values are accepted and returned as `NA`.

Examples

```
allele <- c("A*01:01", "A*01:02")
getAlleleResolution(allele)
```

`getCountsFrequencies` *Calculate variables frequencies*

Description

`getCountsFrequencies` calculate variables frequencies based on counts table, such as produced by `hlaCallsToCounts`.

Usage

```
getCountsFrequencies(counts_table)
```

Arguments

`counts_table`	Data frame containing variables counts, such as produced by `hlaCallsToCounts`.

Details

Variables frequencies are counted in reference to sample size, depending on the inheritance model under which the counts table has been generated one might need to take under consideration both gene copies. Here sample size is assumed to be depended on both gene copies if any count is greater than 1 (‘n / (2 * j)‘ where ‘n‘ is the number of term occurrences and ‘j‘ is the sample size). If this is not the case the sample size is taken as is (‘n / j‘).

Value

Data frame containing variables, its corresponding total counts and frequencies.

See Also

`hlaCallsToCounts`

Examples

```
file <- system.file("extdata", "HLAHD_output_example.txt", package = "MiDAS")
hla_calls <- readHlaCalls(file)
hla_counts <- hlaCallsToCounts(hla_calls, inheritance_model = "additive")
getCountsFrequencies(hla_counts)
```

`getHlaFrequencies` *Calculate alleles frequencies*

Description

`getHlaFrequencies` calculates alleles frequencies in HLA calls data frame.

Usage

```
getHlaFrequencies(hla_calls)
```

Arguments

`hla_calls`	Data frame containing HLA allele calls, as return by `readHlaCalls` function.

Details

Allele frequencies are counted in reference to sample taking both gene copies into consideration. ‘n / (2 * j)‘ where ‘n‘ is the number of allele occurrences and ‘j‘ is the sample size.

Value

Data frame containing alleles and thier corresponding frequencies.

Examples

```
file <- system.file("extdata", "HLAHD_output_example.txt", package = "MiDAS")
hla_calls <- readHlaCalls(file)
getHlaFrequencies(hla_calls)
```

---

getHlaKirInteractions *Get HLA - KIR interactions*

Description

getHlaKirInteractions calculates binary presence-absence matrix of HLA - KIR interactions.

Usage

```
getHlaKirInteractions(hla_calls, kir_counts,
  interactions_dict = system.file("extdata",
  "Match_counts_hla_kir_interactions.txt", package = "MiDAS"))
```

Arguments

hla_calls	Data frame containing HLA allele calls, as return by readHlaCalls function.
kir_counts	Data frame containing KIR genes counts, as return by readKirCalls.
interactions_dict	Path to the file containing HLA - KIR interactions matchings. See details for further details.

Details

In order to be able to compare input data with interactions_dict hla_calls are first converted to variables such as G groups, using matching files shipped with the packages. Moreover hla_calls are also reduced to all possible resolutions.

interactions_dict file should be a tsv format with header and two columns. First column should be named "Name" and hold interactions names, second should be named "Expression" and hold expression used to identify interaction (eg. "C2 & KIR2DL1" will match all samples with C2 and KIR2DL1). The package is shipped with interactions file created based on Pende, et al. 2019.

Value

Data frame with binary presence-absence indicators for HLA - KIR interactions.

Examples

```
hla_file <- system.file("extdata", "HLAHD_output_example.txt", package = "MiDAS")
hla_calls <- readHlaCalls(hla_file)
kir_file <- system.file("extdata", "KIP_output_example.txt", package = "MiDAS")
kir_counts <- readKirCalls(kir_file, counts = TRUE)
getHlaKirInteractions(hla_calls, kir_counts)
```

`getVariableAAPos` *Returns positions of variable amino acids in the alignment*

Description

`getVariableAAPos` finds variable amino acid positions in the alignment.

Usage

```
getVariableAAPos(alignment, varchar = "[A-Z]")
```

Arguments

`alignment`	Matrix containing amino acid level alignment.
`varchar`	Regex matching characters that should be considered when looking for variable amino acid positions. See details for further explanations.

Details

The variable amino acid positions in the alignment are those at which different amino acids can be found. As the alignments can also contain indels and unknown characters, the user choice might be to consider those positions also as variable. This can be achieved by passing appropriate regular expression in `varchar`. Eg. when `varchar = "[A-Z]"` occurence of deletion/insertion (".") will not be treated as variability. In order to detect this kind of variability `varchar = "[A-Z\\.]"` should be used.

Value

Integer vector specifying which alignment columns are variable.

Examples

```
file <- system.file("extdata", "A_prot.txt", package = "MiDAS")
alignment <- readHlaAlignments(file)
getVariableAAPos(alignment)
```

`hasTidyMethod` *Check if tidy method for class exist*

Description

`hasTidyMethod` check if there is tidy method available for given class.

Usage

```
hasTidyMethod(class)
```

Arguments

`class`	Object class.

Value

Logical indicating if there is tidy method for given class.

See Also

Other assert functions: `characterMatches`, `checkAdditionalData`, `checkHlaCallsFormat`, `checkKirCountsFormat`, `checkStatisticalModel`, `colnamesMatches`, `isCharacterOrNULL`, `isClassOrNULL`, `isCountOrNULL`, `isCountsOrZeros`, `isFlagOrNULL`, `isNumberOrNULL`, `isStringOrNULL`, `isTRUEorFALSE`, `stringMatches`

`hlaCallsToCounts` *Transform HLA calls to counts table*

Description

`hlaCallsToCounts` convert HLA calls data frame into counts table.

Usage

```
hlaCallsToCounts(hla_calls, inheritance_model = c("dominant",
  "recessive", "additive"), check_hla_format = TRUE)
```

Arguments

`hla_calls`	Data frame containing HLA allele calls, as return by `readHlaCalls` function.
`inheritance_model`	String specifying inheritance model to use. Available choices are `"dominant"`, `"recessive"`, `"additive"`. In `"dominant"` model homozygotes and heterozygotes are coded as `1`. In `"recessive"` model homozygotes are coded as `1` and all other as `0`. In `"additive"` model homozygotes are coded as `2` and heterozygotes as `1`.
`check_hla_format`	Logical indicating if `hla_calls` format should be checked. This is useful if one wants to use `hlaCallsToCounts` with input not adhering to HLA nomenclature standards. See examples.

Value

Data frame containing counts of HLA alleles according to specified inheritance model.

Examples

```
file <- system.file("extdata", "HLAHD_output_example.txt", package = "MiDAS")
hla_calls <- readHlaCalls(file)
hlaCallsToCounts(hla_calls, inheritance_model = "additive")

# usage with non-HLA alleles numbers input
hla_vars <- hlaToVariable(hla_calls, dictionary = "allele_HLA_supertype")
hlaCallsToCounts(hla_calls, inheritance_model = "additive", check_hla_format = FALSE)
```

`hlaToAAVariation` *Convert HLA allele numbers to amino acid variation matrix*

Description

`hlaToAAVariation` convert HLA allele numbers data frame to a matrix holding information on amino acid variation.

Usage

```
hlaToAAVariation(hla_calls, indels = TRUE, unkchar = FALSE,
  alnpath = system.file("extdata", package = "MiDAS"), as_df = TRUE)
```

Arguments

`hla_calls`	Data frame containing HLA allele calls, as return by `readHlaCalls` function.
`indels`	Logical indicating whether indels should be considered when checking variability.
`unkchar`	Logical indicating whether unknown characters in the alignment should be considered when checking variability.
`alnpath`	String providing optional path to directory containing HLA alignment files. See details for further explanations.
`as_df`	Logical indicating if data frame should be returned. Otherwise matrix is returned.

Details

Variable amino acid positions are found by comparing elements of the alignment column wise. Some of the values in alignment can be treated specially using `indels` and `unkchar` arguments. Function process alignments for all HLA genes found in `hla_calls`.

`alnpath` can be used to provide path to directory containing custom alignment files. Each alignment file have to be named following EBI database convention GENENAME_prot.txt. By default `alnpath` points to directory containing alignment files available at EBI database.

Value

Matrix or data frame containing variable amino acid positions. See `as_df` parameter.

Rownames corresponds to ID column of input data frame, and colnames to alignment positions for given genes. If no variation in amino acid alignments is found function return one column matrix filled with 'NA's.

Examples

```
file <- system.file("extdata", "HLAHD_output_example.txt", package = "MiDAS")
hla_calls <- readHlaCalls(file)
hlaToAAVariation(hla_calls)
```

hlaToVariable *Convert HLA calls data frame according to match table*

Description

`hlaToVariable` convert HLA calls data frame to additional variables based on match table (dictionary).

Usage

```
hlaToVariable(hla_calls, dictionary, reduce = TRUE, na.value = 0,
  nacols.rm = TRUE)
```

Arguments

`hla_calls`	Data frame containing HLA allele calls, as return by `readHlaCalls` function.
`dictionary`	Path to the file containing HLA allele numbers matchings or data frame providing this information. See details for further explanations.
`reduce`	logical indicating if function should try to reduce alleles resolution when no matching is found. See details for more details.
`na.value`	Vector of length one speciyfing value for alleles with no values in dictionary. Default behaviour is to mark such instances with `0`, however in some cases `NA` might be more appropriate.
`nacols.rm`	logical indicating if result columns that contain only `NA` should be removed.

Details

`reduce` control if conversion should happen in a greedy way, such that if some hla numbers cannot be converted, their resolution is reduced by 2 and another attempt is taken. This iterative process stops when alleles cannot be further reduced or all have been successfully converted.

`dictionary` file should be a tsv format with header and two columns. First column should hold allele numbers and second corresponding additional variables. Optionally a data frame formatted in the same manner can be passed instead.

`dictionary` can be also used to access matching files shipped with the package. They can be referred to by using one of the following strings (to list available dictionaries use `listMiDASDictionaries`):

`allele_HLA-A_expression` Reference data to impute expression levels for HLA-A alleles.

`allele_HLA-B_Bw` B alleles can be grouped in allele groups Bw4 and Bw6. In some cases HLA alleles containing Bw4 epitope, on nucleotide level actually carries a premature stop codon. Meaning that although on nucleotide level the allele would encode a Bw4 epitope it's not really there and it is assigned to Bw6 group. However in 4-digit resolution these alleles can not be distinguished from other Bw4 groups. Since alleles with premature stop codons are rare in those ambiguous cases those are assigned to Bw4 group.

allele_HLA_Bw4+A23+A24+A32 Extends `allele_HLA-B_Bw` dictionary by inclusion of A*23, A*24 and A*32 HLA alleles.

`allele_HLA-C_C1-2` C alleles can be grouped in allele groups C1 and C2.

`allele_HLA-C_expression` Reference data to impute expression levels for HLA-C alleles.

`allele_HLA_supertype` A and B alleles can be assigned to so-called supertypes, a classification that group HLA alleles based on peptide binding specificities.

`allele_HLA_Ggroup` HLA alleles can be re-coded in G groups, which defines amino acid identity only in the exons relevant for peptide binding. Note that alleles "DRB1*01:01:01" and "DRB1*01:16" were matched with more than one G group, this ambiguity was removed by deleting matching with "DRB5*01:01:01G" group. Moreover in the original match file there were alleles named "DPA*...", here they are renamed to "DPA1*..." to adhere with HLA nomenclature.

Value

Data frame of HLA numbers converted to additional variables according to match table.

Examples

```
file <- system.file("extdata", "HLAHD_output_example.txt", package = "MiDAS")
hla_calls <- readHlaCalls(file)
hlaToVariable(hla_calls, dictionary = "allele_HLA_supertype")
```

`isCharacterOrNULL` *Check if object is character vector or NULL*

Description

`isCharacterOrNULL` checks if object is character vector or NULL.

Usage

```
isCharacterOrNULL(x)
```

Arguments

`x`	object to test.

Value

Logical indicating if object is character vector or NULL

See Also

Other assert functions: `characterMatches`, `checkAdditionalData`, `checkHlaCallsFormat`, `checkKirCountsFormat`, `checkStatisticalModel`, `colnamesMatches`, `hasTidyMethod`, `isClassOrNULL`, `isCountOrNULL`, `isCountsOrZeros`, `isFlagOrNULL`, `isNumberOrNULL`, `isStringOrNULL`, `isTRUEorFALSE`, `stringMatches`

`isClassOrNULL` *Check if object is of class x or null*

Description

`isClassOrNULL` checks if object is an instance of a specified class or is null.

Usage

```
isClassOrNULL(x, class)
```

Arguments

`x`	object to test.
`class`	String specifying class to test.

Value

Logical indicating if `x` is an instance of `class`.

See Also

Other assert functions: `characterMatches`, `checkAdditionalData`, `checkHlaCallsFormat`, `checkKirCountsFormat`, `checkStatisticalModel`, `colnamesMatches`, `hasTidyMethod`, `isCharacterOrNULL`, `isCountOrNULL`, `isCountsOrZeros`, `isFlagOrNULL`, `isNumberOrNULL`, `isStringOrNULL`, `isTRUEorFALSE`, `stringMatches`

`isCountOrNULL` *Check if object is count or NULL*

Description

`isCountOrNULL` check if object is a count (a single positive integer) or NULL.

Usage

```
isCountOrNULL(x)
```

Arguments

`x`	object to test.

Value

Logical indicating if object is count or NULL

See Also

Other assert functions: `characterMatches`, `checkAdditionalData`, `checkHlaCallsFormat`, `checkKirCountsFormat`, `checkStatisticalModel`, `colnamesMatches`, `hasTidyMethod`, `isCharacterOrNULL`, `isClassOrNULL`, `isCountsOrZeros`, `isFlagOrNULL`, `isNumberOrNULL`, `isStringOrNULL`, `isTRUEorFALSE`, `stringMatches`

isCountsOrZeros *Check if vector contains only counts or zeros*

Description

isCountsOrZeros checks if vector contains only positive integers or zeros.

Usage

```
isCountsOrZeros(x, na.rm = TRUE)
```

Arguments

x	Numeric vector or object that can be unlist to numeric vector.
na.rm	Logical indicating if NA values should be accepted.

Value

Logical indicating if provided vector contains only positive integers or zeros.

See Also

Other assert functions: characterMatches, checkAdditionalData, checkHlaCallsFormat, checkKirCountsFormat, checkStatisticalModel, colnamesMatches, hasTidyMethod, isCharacterOrNULL, isClassOrNULL, isCountOrNULL, isFlagOrNULL, isNumberOrNULL, isStringOrNULL, isTRUEorFALSE, stringMatches

isFlagOrNULL *Check if object is flag or NULL*

Description

isFlagOrNULL checks if object is flag (a length one logical vector) or NULL.

Usage

```
isFlagOrNULL(x)
```

Arguments

x	object to test.

Value

Logical indicating if object is flag or NULL

See Also

Other assert functions: characterMatches, checkAdditionalData, checkHlaCallsFormat, checkKirCountsFormat, checkStatisticalModel, colnamesMatches, hasTidyMethod, isCharacterOrNULL, isClassOrNULL, isCountOrNULL, isCountsOrZeros, isNumberOrNULL, isStringOrNULL, isTRUEorFALSE, stringMatches

isNumberOrNULL *Check if object is number or NULL*

Description

`isNumberOrNULL` checks if object is number (a length one numeric vector) or NULL.

Usage

```
isNumberOrNULL(x)
```

Arguments

x	object to test.

Value

Logical indicating if object is number or NULL

See Also

Other assert functions: `characterMatches`, `checkAdditionalData`, `checkHlaCallsFormat`, `checkKirCountsFormat`, `checkStatisticalModel`, `colnamesMatches`, `hasTidyMethod`, `isCharacterOrNULL`, `isClassOrNULL`, `isCountOrNULL`, `isCountsOrZeros`, `isFlagOrNULL`, `isStringOrNULL`, `isTRUEorFALSE`, `stringMatches`

isStringOrNULL *Check if object is string or NULL*

Description

`isStringOrNULL` checks if object is string (a length one character vector) or NULL.

Usage

```
isStringOrNULL(x)
```

Arguments

x	object to test.

Value

Logical indicating if object is string or NULL

See Also

Other assert functions: `characterMatches`, `checkAdditionalData`, `checkHlaCallsFormat`, `checkKirCountsFormat`, `checkStatisticalModel`, `colnamesMatches`, `hasTidyMethod`, `isCharacterOrNULL`, `isClassOrNULL`, `isCountOrNULL`, `isCountsOrZeros`, `isFlagOrNULL`, `isNumberOrNULL`, `isTRUEorFALSE`, `stringMatches`

`isTRUEorFALSE` *Check if object is TRUE or FALSE flag*

Description

`isTRUEorFALSE` check if object is a flag (a length one logical vector) except NA.

Usage

```
isTRUEorFALSE(x)
```

Arguments

`x`	object to test.

Value

Logical indicating if object is TRUE or FALSE flag

See Also

Other assert functions: `characterMatches`, `checkAdditionalData`, `checkHlaCallsFormat`, `checkKirCountsFormat`, `checkStatisticalModel`, `colnamesMatches`, `hasTidyMethod`, `isCharacterOrNULL`, `isClassOrNULL`, `isCountOrNULL`, `isCountsOrZeros`, `isFlagOrNULL`, `isNumberOrNULL`, `isStringOrNULL`, `stringMatches`

`kirHaplotypeToCounts` *Convert KIR haplotypes to gene counts*

Description

`kirHaplotypeToCounts` convert vector of KIR haplotypes to data frame of KIR gene counts.

Usage

```
kirHaplotypeToCounts(x, hap_dict = system.file("extdata",
  "Match_kir_haplotype_gene.txt", package = "MiDAS"), binary = TRUE)
```

Arguments

`x`	Character vector specifying KIR haplotypes.
`hap_dict`	String specifying path to KIR haplotypes dictionary. By default file shipped together with package is being used. See details for more information.
`binary`	Logical flag indicating if haplotypes should be converted only to gene presence / absence indicators (it is the only way that allows unambiguous conversion). This argument is currently ignored. `hap_dict` have to be a tab separated values formatted file with first column holding KIR haplotypes and gene counts in others. File should have header with first column unnamed and gene names in the others.

Value

Data frame with haplotypes and corresponding gene counts. `NA`'s in `x` are removed during conversion.

See Also

`readKirCalls`, `getHlaKirInteractions`, `checkKirCountsFormat`, `prepareMiDAS`.

Examples

```
x <- c(NA, "1+3|16+3", "1+1", NA)
kirHaplotypeToCounts(x)
```

`listMiDASDictionaries` *List HLA alleles dictionaries*

Description

`listMiDASDictionaries` lists dictionaries shipped with MiDAS package.

Usage

```
listMiDASDictionaries(pattern = ".*", file.names = FALSE)
```

Arguments

`pattern`	String used to match dictionary names, it can be a regular expression. By default all names are matched.
`file.names`	Logical value. If FALSE, only the names of dictionaries are returned. If TRUE their paths are returned.

Value

Character vector with names of HLA alleles dictionaries.

`prepareMiDAS` *Prepare MiDAS data for statistical analysis*

Description

`prepareMiDAS` transform HLA alleles calls and KIR calls according to selected analysis type and join obtained transformation with additional data like phenotypic observations or covariates.

Usage

```
prepareMiDAS(hla_calls, ..., kir_counts = NULL,
  analysis_type = c("hla_allele", "aa_level", "expression_level",
  "allele_g_group", "allele_supertype", "allele_group", "kir_genes",
  "hla_kir_interactions", "custom"), inheritance_model = "additive",
  indels = TRUE, unkchar = FALSE)
```

Arguments

`hla_calls`	Data frame containing HLA allele calls, as return by `readHlaCalls` function.
`...`	Data frames holding additional variables like phenotypic observations or covariates.
`kir_counts`	Data frame with KIR genes counts. Required for `"kir_genes"` analysis type.
`analysis_type`	String indicating analysis type for which data should be prepared. Valid choices are `"hla_allele"`, `"aa_level"`, `"expression_level"`, `"allele_group"`, `"custom"`. Each prepared variable will be labeled with corresponding `analysis_type`. See details for further explanations.
`inheritance_model`	String specifying inheritance model to use. Available choices are `"dominant"`, `"recessive"`, `"additive"`. In `"dominant"` model homozygotes and heterozygotes are coded as `1`. In `"recessive"` model homozygotes are coded as `1` and all other as `0`. In `"additive"` model homozygotes are coded as `2` and heterozygotes as `1`.
`indels`	Logical indicating whether indels should be considered when checking variability.
`unkchar`	Logical indicating whether unknown characters in the alignment should be considered when checking variability.

Details

Data frames passed as arguments to the function are joined using `hla_calls` as a reference. As a consequence all samples with HLA data are kept, independent of whether there's actually phenotypes / covariates. Moreover phenotypes / covariates without HLA data are discarded.

`...` should be data frames with first column holding sample IDs and named `ID`. Those should correspond to `ID` column in `hla_calls` and `kir_counts`.

Choices for `analysis_type`:

`hla_allele` `hla_calls` are transformed into counts under `inheritance_model` of choice (see `hlaCallsToCounts` for more details).

`aa_level` `hla_calls` are first converted to amino acid level, taking only variable positions under consideration. Than variable amino acid positions are transformed to counts under `inheritance_model` of choice (see `hlaToAAVariation` and `aaVariationToCounts` for more details).

`expression_level` `hla_calls` are transformed to expression levels using expression dictionaries shipped with package (see `hlaToVariable` for more details). Expression levels from both alleles are summed into single variable for each HLA gene.

`allele_g_group` `hla_calls` are transformed to HLA alleles groups using G group dictionary shipped with the package. Than they are transformed to counts under `inheritance_model` of choice (see `hlaToVariable` and `hlaCallsToCounts` for more details).

.

`allele_supertype` `hla_calls` are transformed to HLA alleles groups using supertypes dictionary shipped with the package. Than they are transformed to counts under `inheritance_model` of choice (see `hlaToVariable` and `hlaCallsToCounts` for more details).

`allele_group` `hla_calls` are transformed to HLA alleles groups using Bw4/6, C1/2 and Bw4+A23+A24+A32 dictionaries shipped with the package. Than they are transformed to counts under `inheritance_model` of choice (see `hlaToVariable` and `hlaCallsToCounts` for more details).

`kir_genes` `kir_counts` data frame is joined with other inputs.

hla_kir_interactions hla_calls are processed with kir_counts into HLA - KIR interactions variables (see getHlaKirInteractions for more details).

custom No data transformation is done. All inputs are joined together.

Value

Data frame containing prepared data.

See Also

Other MiDAS statistical functions: analyzeAssociations, analyzeConditionalAssociations, runMiDAS

Examples

```
hla_calls_file <- system.file("extdata", "HLAHD_output_example.txt", package = "MiDAS")
hla_calls <- readHlaCalls(hla_calls_file)
pheno_file <- system.file("extdata", "pheno_example.txt", package = "MiDAS")
pheno <- read.table(pheno_file, header = TRUE)
covar_file <- system.file("extdata", "covar_example.txt", package = "MiDAS")
covar <- read.table(covar_file, header = TRUE)
prepareMiDAS(hla_calls, pheno, covar, analysis_type = "expression_level")
```

---

readHlaAlignments *Read HLA allele alignments*

Description

readHlaAlignments read HLA allele alignments from file.

Usage

```
readHlaAlignments(file, gene = NULL, trim = TRUE, unkchar = "",
  resolution = 8)
```

Arguments

file	Path to input file.
gene	Character vector of length one specifying the name of a gene for which alignment is required. See details for further explanations.
trim	Logical indicating if alignment should be trimmed to start codon of the mature protein.
unkchar	Character to be used to represent positions with unknown sequence.
resolution	Numeric vector of length one specifying output resolution.

Details

HLA allele alignment file should follow EBI database format, for details see `ftp://ftp.ebi.ac.uk/pub/databases/ipd/imgt/hla/alignments/README.md`.

All protein alignment files from EBI database are shipped with the package. They can be easily accessed using gene parameter. If gene is set to `NULL` file parameter is used instead and alignment is read from the provided file. In EBI database alignments for DRB1, DRB3, DRB4 and DRB5 genes are provided as a single file, here they are separated.

Value

Matrix containing HLA allele alignments.

Rownames corresponds to allele numbers and columns to positions in the alignment. Sequences following the termination codon are marked as empty character (`""`). Unknown sequences are marked with a character of choice, by default `""`. Stop codons are represented by a hash (X). Insertion and deletions are marked with period (.).

Examples

```
hla_alignments <- readHlaAlignments(gene = "A")
```

`readHlaCalls` *Read HLA allele calls data*

Description

`readHlaCalls` read HLA allele calls from file

Usage

```
readHlaCalls(file, resolution = 4, na.strings = c("Not typed", "-",
  "NA"))
```

Arguments

`file`	Path to input file.
`resolution`	Numeric vector of length one specifying output resolution.
`na.strings`	a character vector of strings which are to be interpreted as `NA` values. Blank fields are also considered to be missing values in logical, integer, numeric and complex fields. Note that the test happens *after* white space is stripped from the input, so `na.strings` values may need their own white space stripped in advance.

Details

Input file have to be a tsv formatted table with header. First column should contain sample IDs, further columns should hold corresponding HLA allele numbers.

`resolution` parameter can be used to reduce HLA allele numbers. If reduction is not needed `resolution` can be set to 8. `resolution` parameter can take following values: 2, 4, 6, 8. For more details about HLA allele numbers resolution see `http://hla.alleles.org/nomenclature/naming.html`.

Value

Data frame containing HLA allele calls.

Examples

```
file <- system.file("extdata", "HLAHD_output_example.txt", package = "MiDAS")
hla_calls <- readHlaCalls(file)
```

readKirCalls *Reads data table with KIR haplotypes calls*

Description

`readKirCalls` reads table with KIR haplotypes calls from file.

Usage

```
readKirCalls(file, hap_dict = system.file("extdata",
  "Match_kir_haplotype_gene.txt", package = "MiDAS"), counts = TRUE,
  binary = TRUE, na.strings = c("", "NA"))
```

Arguments

`file`	Path to input file.
`hap_dict`	String specifying path to KIR haplotypes dictionary. By default file shipped together with package is being used. See details for more information.
`counts`	Logical flag indicating if KIR haplotypes should be converted to gene counts.
`binary`	Logical flag indicating if haplotypes should be converted only to gene presence / absence indicators (it is the only way that allows unambiguous conversion). This argument is currently ignored. `hap_dict` have to be a tab separated values formatted file with first column holding KIR haplotypes and gene counts in others. File should have header with first column unnamed and gene names in the others.
`na.strings`	a character vector of strings which are to be interpreted as `NA` values. Blank fields are also considered to be missing values in logical, integer, numeric and complex fields. Note that the test happens *after* white space is stripped from the input, so `na.strings` values may need their own white space stripped in advance.

Details

Input file have to be a tsv formatted table with two columns and header. First column should contain samples IDs, second column should hold corresponding KIR haplotypes.

Value

Data frame containing KIR haplotypes calls or corresponding gene counts.

Examples

```
file <- system.file("extdata", "KIP_output_example.txt", package = "MiDAS")
readKirCalls(file)
```

reduceAlleleResolution   *Reduce HLA allele resolution*

Description

`reduceAlleleResolution` reduces HLA allele numbers vector to specified resolution.

Usage

```
reduceAlleleResolution(allele, resolution = 4)
```

Arguments

`allele`	Character vector containing HLA allele numbers.
`resolution`	Numeric vector of length one specifying output resolution.

Details

In cases when allele numbers contains additional suffix their resolution can not be unambiguously reduced. These cases are returned unchanged. Function behaves in the same manner if `resolution` is higher than resolution of input HLA allele numbers.

Value

Character vector containing reduced HLA allele numbers.

`NA` values are accepted and returned as `NA`.

Examples

```
reduceAlleleResolution(c("A*01", "A*01:24", "C*05:24:55:54"), 2)
```

reduceHlaCalls *Reduce HLA calls data frame resolution*

Description

`reduceHlaCalls` reduce HLA calls data frame to specified resolution.

Usage

```
reduceHlaCalls(hla_calls, resolution = 4)
```

Arguments

`hla_calls`	Data frame containing HLA allele calls, as return by `readHlaCalls` function.
`resolution`	Numeric vector of length one specifying output resolution.

Details

If `resolution` is greater than resolution of `hla_calls` elements, those elements will be unchanged. Elements with optional suffixes are not reduced.

Value

Data frame containing HLA allele calls reduced to required resolution.

Examples

```
file <- system.file("extdata", "HLAHD_output_example.txt", package = "MiDAS")
hla_calls <- readHlaCalls(file)
reduceHlaCalls(hla_calls, resolution = 2)
```

runMiDAS *Analyze MiDAS data*

Description

`runMiDAS` perform association analysis on MiDAS data using statistical model specified by user. Function is intended for use with `prepareMiDAS`. See examples section.

Usage

```
runMiDAS(object, analysis_type = c("hla_allele", "aa_level",
  "expression_level", "allele_g_group", "allele_supertype", "allele_group",
  "kir_genes", "hla_kir_interactions"), pattern = NULL,
  variables = NULL, conditional = FALSE, keep = FALSE,
  lower_frequency_cutoff = NULL, upper_frequency_cutoff = NULL,
  pvalue_cutoff = NULL, correction = "bonferroni",
  n_correction = NULL, logistic = NULL, exponentiate = NULL,
  th = 0.05, rss_th = 1e-07)
```

Arguments

`object`	An existing fit from a model function such as lm, glm and many others.
`analysis_type`	String indicating the type of analysis to be performed, it's used to select appropriate variables for testing from the data associated with `object`. Valid values are `"hla_allele"`, `"aa_level"`, `"expression_level"`, `"allele_g_group"`, `"allele_supertype"`, `"allele_group"`, `"kir_genes"`, `"hla_kir_interactions"`. See details for further explanations.
`pattern`	String containing a regular expression that is used to further select variables selected by `analysis_type`.
`variables`	Character vector specifying additional variables to use in association tests except those selected by `analysis_type`. By default `NULL`.
`conditional`	Logical flag indicating if the analysis should be performed using stepwise conditional test. See `analyzeConditionalAssociations` for more details.
`keep`	Logical flag indicating if the output should be a list of results resulting from each selection step. Default is to return only the final result.
`lower_frequency_cutoff`	Number specifying lower threshold for inclusion of a variable. If it's a number between `0` and `1` variables with frequency below this number will not be considered during analysis. If it's greater or equal `1` variables with number of counts less that this will not be considered during analysis. Only applied to discrete variables.
`upper_frequency_cutoff`	Number specifying upper threshold for inclusion of a variable. If it's a number between `0` and `1` variables with frequency above this number will not be considered during analysis. If it's greater or equal `1` variables with number of counts greater that this will not be considered during analysis.Only applied to discrete variables.
`pvalue_cutoff`	Number specifying p-value cutoff for results to be included in output. If `NULL` cutoff of `0.05` on `p.adjusted` value is used instead.
`correction`	String specifying multiple testing correction method. See details for further information.
`n_correction`	Integer specifying number of comparisons to consider during multiple testing correction calculations, must be at least equal to number of comparisons being made; only set this (to non-default) when you know what you are doing!
`logistic`	Logical flag indicating if statistical model used is logistic (eg. `coxph`). If `NULL` function will try to figure this out. This is only used for results formatting.
`exponentiate`	Logical flag indicating if coefficient estimates should be exponentiated. This is typical for logistic and multinomial regressions, but a bad idea if there is no log or logit link. If `NULL` function will try to figure this out by testing if response is binary (`0` or `1`).
`th`	Number specifying p-value threshold for a variable to be considered significant.
`rss_th`	Number specifying residual sum of squares threshold at which function should stop adding additional variables. As the residual sum of squares approaches `0` the perfect fit is obtained making further attempts at variables selection nonsense, thus function is stopped. This behavior can be controlled using `rss_th`.

Details

analysis_type is used to select variables from data associated with object using labels. In standard work flow data are first processed using prepareMiDAS, columns of its output data frame are labeled with the type of analysis they can be used for eg. hla_allele. By specifying analysis_type function will select all variables with corresponding label. This choice can be further refined by using pattern argument or extended with variables.

correction specifies p-value adjustment method to use, common choice is Benjamini & Hochberg (1995) ("BH"). Internally this is passed to p.adjust. Check there to get more details.

Value

Tibble containing results for tested variables.

See Also

p.adjust, tidy

Other MiDAS statistical functions: analyzeAssociations, analyzeConditionalAssociations, prepareMiDAS

Examples

```
library("survival")
hla_calls_file <- system.file("extdata", "HLAHD_output_example.txt", package = "MiDAS")
hla_calls <- readHlaCalls(hla_calls_file)
pheno_file <- system.file("extdata", "pheno_example.txt", package = "MiDAS")
pheno <- read.table(pheno_file, header = TRUE, stringsAsFactors = FALSE)
covar_file <- system.file("extdata", "covar_example.txt", package = "MiDAS")
covar <- read.table(covar_file, header = TRUE, stringsAsFactors = FALSE)
midas_data <- prepareMiDAS(hla_calls = hla_calls,
                               pheno = pheno,
                               covar = covar,
                               analysis_type = "hla_allele",
                               inheritance_model = "additive"
)

object <- coxph(Surv(OS, OS_DIED) ~ AGE + SEX, data = midas_data)
runMiDAS(object, analysis_type = "hla_allele")
```

---

stringMatches *Check if string matches one of possible values*

Description

stringMatches checks if string is equal to one of the choices.

Usage

```
stringMatches(x, choice)
```

Arguments

`x`	string to test.
`choice`	Character vector with possible values for `x`.

Value

Logical indicating if `x` matches one of the strings in `choice`.

See Also

Other assert functions: `characterMatches`, `checkAdditionalData`, `checkHlaCallsFormat`, `checkKirCountsFormat`, `checkStatisticalModel`, `colnamesMatches`, `hasTidyMethod`, `isCharacterOrNULL`, `isClassOrNULL`, `isCountOrNULL`, `isCountsOrZeros`, `isFlagOrNULL`, `isNumberOrNULL`, `isStringOrNULL`, `isTRUEorFALSE`

`summariseAAPosition` *Summarize amino acid position*

Description

List HLA alleles and amino acid residues at a given position

Usage

```
summariseAAPosition(hla_calls, pos, aln = NULL, na.rm = FALSE)
```

Arguments

`hla_calls`	Data frame containing HLA allele calls, as return by `readHlaCalls` function.
`pos`	String specifying gene and amino acid position, example `"A_9"`.
`aln`	Matrix containing amino acid sequence alignments as returned by `readHlaAlignments` function. By default function will use alignment files shipped with the package.
`na.rm`	Logical flag indicating if `NA` values should be considered for frequency calculations.

Value

Data frame containing HLA alleles, their corresponding amino acid residues and frequencies at requested position.

Examples

```
file <- system.file("extdata", "HLAHD_output_example.txt", package = "MiDAS")
hla_calls <- readHlaCalls(file)
summariseAAPosition(hla_calls, "A_9")
```

`updateModel` *Add new variables to statistical model*

Description

`updateModel` adds new variables to model and re-fit it.

Usage

```
updateModel(object, x, backquote = TRUE, collapse = " + ")
```

Arguments

`object`	An existing fit from a model function such as lm, glm and many others.
`x`	Character vector specifying variables to be added to model or a formula giving a template which specifies how to update.
`backquote`	Logical indicating if added variables should be quoted. Elements of this vector are recycled over `x`. Only relevant if `x` is of type character.
`collapse`	String specifying how new characters should be added to old formula. Only relevant if `x` is of type character.

Value

Updated fit of input model.

Examples

```
object <- lm(dist ~ 1, data = cars)
updateModel(object, "dist")
```

Index