Package ‘midasHLA’

January 5, 2024

Title  R package for immunogenomics data handling and association analysis

Version  1.10.0

Description  MiDAS is a R package for immunogenetics data transformation and statistical analysis. MiDAS accepts input data in the form of HLA alleles and KIR types, and can transform it into biologically meaningful variables, enabling HLA amino acid fine mapping, analyses of HLA evolutionary divergence, KIR gene presence, as well as validated HLA-KIR interactions. Further, it allows comprehensive statistical association analysis workflows with phenotypes of diverse measurement scales. MiDAS closes a gap between the inference of immunogenetic variation and its efficient utilization to make relevant discoveries related to T cell, Natural Killer cell, and disease biology.

License  MIT + file LICENCE

Encoding  UTF-8

LazyData  true

Depends  R (>= 4.1), MultiAssayExperiment (>= 1.8.3)

Imports  assertthat (>= 0.2.0), broom (>= 0.5.1), dplyr (>= 0.8.0.1), formattable (>= 0.2.0.1), HardyWeinberg (>= 1.6.3), kableExtra (>= 1.1.0), knitr (>= 1.21), magrittr (>= 1.5), methods, stringi (>= 1.2.4), rlang (>= 0.3.1), S4Vectors (>= 0.20.1), stats, SummarizedExperiment (>= 1.12.0), tibble (>= 2.0.1), utils, qdapTools (>= 1.3.3)

Suggests  broom.mixed (>= 0.2.4), cowplot (>= 1.0.0), devtools (>= 2.0.1), ggplot2 (>= 3.1.0), ggpubr (>= 0.2.5), rmarkdown, seqinr (>= 3.4-5), survival (>= 2.43-3), testthat (>= 2.0.1), tidyr (>= 1.1.2)

RoxygenNote  7.1.1

VignetteBuilder  knitr

Collate  'asserts.R' 'class.R' 'data.R' 'global.R' 'midasHLA.R'
 'parsingFunctions.R' 'stats.R' 'summarise.R'
 'transformationFunctions.R' 'utils.R'

biocViews  CellBiology, Genetics, StatisticalMethod

git_url  https://git.bioconductor.org/packages/midasHLA
git_branch  RELEASE_3_18
git_last_commit  a67beb5
git_last_commit_date  2023-10-24
Repository  Bioconductor 3.18
Date/Publication  2024-01-05
Author  Christian Hammer [aut], Maciej Migdał [aut, cre]
Maintainer  Maciej Migdał <mcjmigdal@gmail.com>

R topics documented:

aaVariationToCounts  .................................................. 4
adjustPValues  .......................................................... 5
allele_frequencies  ..................................................... 5
analyzeAssociations  ................................................... 6
analyzeConditionalAssociations  ................................. 7
applyInheritanceModel  ................................................ 9
as.data.frame.MiDAS  .................................................. 10
backquote  .............................................................. 10
characterMatches  ...................................................... 11
checkAlleleFormat  .................................................... 11
checkColDataFormat  ................................................... 12
checkHlaCallsFormat  .................................................. 13
checkKirCallsFormat  .................................................. 13
checkKirGenesFormat  .................................................. 14
checkStatisticalModel  .................................................. 14
colnamesMatches  ...................................................... 15
convertAlleleToVariable  ............................................. 15
countsToVariables  .................................................... 16
dfToExperimentMat  ...................................................... 17
dict_dist_grantham  .................................................... 17
distGrantham  .......................................................... 18
experimentMatToDf  ..................................................... 18
filterByFrequency  ..................................................... 19
filterByOmnibusGroups  .............................................. 20
filterByVariables  ...................................................... 20
filterExperimentByFrequency  ................................. 21
filterExperimentByVariables  .................................. 22
filterListByElements  ................................................. 23
formatResults  .......................................................... 23
getAAFrequencies  ...................................................... 24
getAlleleResolution  .................................................. 25
getAllelesForAA  ....................................................... 26
getExperimentFrequencies  ...................................... 26
getExperimentPopulationMultiplier  .......................... 27
getExperiments  ......................................................... 28
R topics documented:

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>getFrequencies</td>
<td>28</td>
</tr>
<tr>
<td>getFrequencyMask</td>
<td>30</td>
</tr>
<tr>
<td>getHlaCalls</td>
<td>31</td>
</tr>
<tr>
<td>getHlaCallsGenes</td>
<td>31</td>
</tr>
<tr>
<td>getHlaFrequencies</td>
<td>32</td>
</tr>
<tr>
<td>getHlaKirInteractions</td>
<td>33</td>
</tr>
<tr>
<td>getKirCalls</td>
<td>34</td>
</tr>
<tr>
<td>getKIRFrequencies</td>
<td>34</td>
</tr>
<tr>
<td>getObjectDetails</td>
<td>35</td>
</tr>
<tr>
<td>getOmnibusGroups</td>
<td>35</td>
</tr>
<tr>
<td>getPlaceholder</td>
<td>36</td>
</tr>
<tr>
<td>getReferenceFrequencies</td>
<td>36</td>
</tr>
<tr>
<td>getVariableAAPos</td>
<td>37</td>
</tr>
<tr>
<td>hasTidyMethod</td>
<td>38</td>
</tr>
<tr>
<td>hlaAlignmentGrantham</td>
<td>38</td>
</tr>
<tr>
<td>hlaCallsGranthamDistance</td>
<td>39</td>
</tr>
<tr>
<td>hlaCallsToCounts</td>
<td>40</td>
</tr>
<tr>
<td>hlaToAAVariation</td>
<td>40</td>
</tr>
<tr>
<td>hlaToVariable</td>
<td>41</td>
</tr>
<tr>
<td>HWETest</td>
<td>42</td>
</tr>
<tr>
<td>isCharacterOrNULL</td>
<td>44</td>
</tr>
<tr>
<td>isClass</td>
<td>44</td>
</tr>
<tr>
<td>isClassOrNULL</td>
<td>45</td>
</tr>
<tr>
<td>isCountOrNULL</td>
<td>45</td>
</tr>
<tr>
<td>isCountsOrZeros</td>
<td>46</td>
</tr>
<tr>
<td>isExperimentCountsOrZeros</td>
<td>46</td>
</tr>
<tr>
<td>isExperimentInheritanceModelApplicable</td>
<td>47</td>
</tr>
<tr>
<td>isFlagOrNULL</td>
<td>47</td>
</tr>
<tr>
<td>isNumberOrNULL</td>
<td>48</td>
</tr>
<tr>
<td>isStringOrNULL</td>
<td>48</td>
</tr>
<tr>
<td>isTRUEorFALSE</td>
<td>49</td>
</tr>
<tr>
<td>iterativeLRT</td>
<td>49</td>
</tr>
<tr>
<td>iterativeModel</td>
<td>50</td>
</tr>
<tr>
<td>kableResults</td>
<td>50</td>
</tr>
<tr>
<td>kir_frequencies</td>
<td>51</td>
</tr>
<tr>
<td>lapply_tryCatch</td>
<td>52</td>
</tr>
<tr>
<td>listMiDASDictionaries</td>
<td>53</td>
</tr>
<tr>
<td>LRTest</td>
<td>53</td>
</tr>
<tr>
<td>MiDAS-class</td>
<td>54</td>
</tr>
<tr>
<td>midasToWide</td>
<td>56</td>
</tr>
<tr>
<td>MiDAS_tut_HLA</td>
<td>56</td>
</tr>
<tr>
<td>MiDAS_tut_KIR</td>
<td>57</td>
</tr>
<tr>
<td>MiDAS_tut_object</td>
<td>58</td>
</tr>
<tr>
<td>MiDAS_tut_pheno</td>
<td>59</td>
</tr>
<tr>
<td>objectHasPlaceholder</td>
<td>59</td>
</tr>
<tr>
<td>omnibusTest</td>
<td>60</td>
</tr>
<tr>
<td>prepareMiDAS</td>
<td>61</td>
</tr>
<tr>
<td>prepareMiDAS_hla_aa</td>
<td>63</td>
</tr>
</tbody>
</table>
Transform amino acid variation data frame into counts table

**Description**

aaVariationToCounts convert amino acid variation data frame into counts table.

**Usage**

```r
aaVariationToCounts(aa_variation)
```

**Arguments**

- `aa_variation` Amino acid variation data frame as returned by `hlaToAAVariation`.

**Value**

Amino acid counts data frame. First column holds samples ID’s, further columns, corresponding to specific amino acid positions, give information on the number of their occurrences in each sample.
**adjustPValues**

*Adjust P-values for Multiple Comparisons*

**Description**

Given a set of p-values, returns p-values adjusted using one of several methods.

**Usage**

```r
adjustPValues(p, method, n = length(p))
```

**Arguments**

- `p` numeric vector of p-values (possibly with NAs). Any other R object is coerced by `as.numeric`.
- `method` correction method. Can be abbreviated.
- `n` number of comparisons, must be at least `length(p)`; only set this (to non-default) when you know what you are doing! Note that for Bonferroni correction it is possible to specify number lower than `length(p)`.

**Details**

This function modifies `stats::p.adjust` method such that for Bonferroni correction it is possible to specify `n` lower than `length(p)`. This feature is useful in cases when knowledge about the biology or redundancy of alleles reduces the need for correction.

See `p.adjust` for more details.

**Value**

A numeric vector of corrected p-values (of the same length as `p`, with names copied from `p`).

---

**allele_frequencies**

*Alleles frequencies scraped from allelefrequencies.net*

**Description**

Accessed on 28.07.20

**Usage**

```r
allele_frequencies
```
analyzeAssociations

Format
A data frame with 2096 rows and 3 variables:
- **var**: allele number, character
- **population**: reference population name, character
- **frequency**: allele frequency in reference population, float

Details
A dataset containing allele frequencies across 5697 alleles. For details visit the search results page in the allelefrequencies.net database website.

Source
www.allelefrequencies.net

---

**analyzeAssociations**  
*Association analysis*

Description

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

Usage

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

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object</code></td>
<td>An existing fit from a model function such as <code>lm</code>, <code>glm</code> and many others.</td>
</tr>
<tr>
<td><code>variables</code></td>
<td>Character vector specifying variables to use in association tests.</td>
</tr>
<tr>
<td><code>placeholder</code></td>
<td>String specifying term in object’s formula which should be substituted with variables during analysis.</td>
</tr>
<tr>
<td><code>correction</code></td>
<td>String specifying multiple testing correction method. See details for further information.</td>
</tr>
</tbody>
</table>
analyzeConditionalAssociations

n_correction Integer specifying number of comparisons to consider during multiple testing correction calculations. For Bonferroni correction it is possible to specify a number lower than the number of comparisons being made. This is useful in cases when knowledge about the biology or redundancy of alleles reduces the need for correction. For other methods it must be at least equal to the 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 variables. The first column "term" hold the names of variables. Further columns depends on the used model and are determined by associated tidy function. Generally they will include "estimate", "std.error", "statistic", "p.value", "conf.low", "conf.high", "p.adjusted".

Examples

midas <- prepareMiDAS(hla_calls = MiDAS_tut_HLA, colData = MiDAS_tut_pheno, experiment = "hla_alleles")

# analyzeAssociations expects model data to be a data.frame
midas_data <- as.data.frame(midas)

# define base model
object <- lm(disease ~ term, 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 are no more significant variables based on a self-defined threshold.
analyzeConditionalAssociations(
  object,
  variables,
  placeholder = "term",
  correction = "bonferroni",
  n_correction = NULL,
  th,
  th_adj = TRUE,
  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.
placeholder       String specifying term to substitute with value from x. Ignored if set to NULL.
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. For Bonferroni correction it is possible to specify a number lower than the number of comparisons being made. This is useful in cases when knowledge about the biology or redundancy of alleles reduces the need for correction. For other methods it must be at least equal to the number of comparisons being made; only set this (to non-default) when you know what you are doing!
th               Number specifying 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 variable selection nonsense. 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.

Value

Tibble with stepwise conditional testing results or a list of tibbles, see keep argument. The first column "term" hold the names of variables. Further columns depends on the used model and are determined by associated tidy function. Generally they will include "estimate", "std.error", "statistic", "p.value", "conf.low", "conf.high", "p.adjusted".
**applyInheritanceModel**

**Examples**

```r
midas <- prepareMiDAS(hla_calls = MiDAS_tut_HLA,  
colData = MiDAS_tut_pheno,  
experiment = "hla_alleles")

# analyzeConditionalAssociations expects model data to be a data.frame  
midas_data <- as.data.frame(midas)

# define base model  
object <- lm(disease ~ term, data = midas_data)  
analyzeConditionalAssociations(object,  
variables = c("B*14:02", "DRB1*11:01"),  
th = 0.05)
```

---

**applyInheritanceModel**  Apply inheritance model

**Description**

Helper function transforming experiment counts to selected **inheritance_model**.

**Usage**

```r
applyInheritanceModel(  
  experiment,  
  inheritance_model = c("dominant", "recessive", "additive", "overdominant")
)
```

```r
## S3 method for class 'matrix'
applyInheritanceModel(  
  experiment,  
  inheritance_model = c("dominant", "recessive", "additive", "overdominant")
)
```

```r
## S3 method for class 'SummarizedExperiment'
applyInheritanceModel(  
  experiment,  
  inheritance_model = c("dominant", "recessive", "additive", "overdominant")
)
```

**Arguments**

- **experiment**: Matrix or SummarizedExperiment object.
- **inheritance_model**: String specifying inheritance model to use. Available choices are "dominant”, “recessive”, “additive”. 

Details
Under "dominant" model homozygotes and heterozygotes are coded as 1. In "recessive" model homozygotes are coded as 1 and other as 0. In "additive" model homozygotes are coded as 2 and heterozygotes as 1. In "overdominant" homozygotes (both 0 and 2) are coded as 0 and heterozygotes as 1.

Value
experiment converted to specified inheritance model.

---

as.data.frame.MiDAS  Coerce MiDAS to Data Frame

Description
Coerce MiDAS to Data Frame

Usage
```r
## S3 method for class 'MiDAS'
as.data.frame(x, ...)
```

Arguments
- `x` any R object.
- `...` additional arguments to be passed to or from methods.

Value
Data frame representation of MiDAS object. Consecutive columns hold values of variables from MiDAS's experiments and colData. The metadata associated with experiments is not preserved.

---

backquote  Backquote character

Description
backquote places backticks around elements of character vector

Usage
```r
backquote(x)
```

Arguments
- `x` Character vector.
**characterMatches**

**Details**

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

**Value**

Character vector with its elements backticked.

---

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

**Description**

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

**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`.

---

**checkAlleleFormat**

**Check HLA allele format**

**Description**

checkAlleleFormat test if the input character follows HLA nomenclature specifications.

**Usage**

```r
checkAlleleFormat(allele)
```

**Arguments**

- `allele` : Character vector with 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. They are interpreted as valid HLA alleles designations.

Value

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

Examples

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

Description

checkColDataFormat asserts if the colData data frame has proper format.

Usage

```r
checkColDataFormat(data_frame)
```

Arguments

- `data_frame` Data frame containing colData data used to construct MiDAS object.

Value

Logical indicating if data_frame is properly formatted. Otherwise raise an error.
checkHlaCallsFormat

Description

checkHlaCallsFormat asserts if hla calls data frame have proper format.

Usage

cHECKHlACallsFormat(hla_calls)

Arguments

hla_calls HLA calls data frame, as returned by readHlACalls function.

Value

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

checkKirCallsFormat

Description

checkKirCallsFormat asserts if KIR counts data frame have proper format.

Usage

checkKirCallsFormat(kir_calls)

Arguments

kir_calls KIR calls data frame, as returned by readKirCalls function.

Value

Logical indicating if kir_calls follow KIR counts data frame format. Otherwise raise an error.
checkKirGenesFormat  
**Check KIR genes format**

**Description**
checkKirGenesFormat test if the input character follows KIR gene names naming conventions.

**Usage**
checkKirGenesFormat(genes)

**Arguments**
genes  
Character vector with KIR gene names.

**Details**
KIR genes: "KIR3DL3", "KIR2DS2", "KIR2DL2", "KIR2DL3", "KIR2DP1", "KIR2DL1", "KIR3DP1", "KIR2DL1", "KIR3DP1", "KIR2DL4", "KIR3DL1", "KIR3DS1", "KIR2DL5", "KIR2DS3", "KIR2DS5", "KIR2DS4", "KIR2DS1", "KIR3DL2".

**Value**
Logical vector specifying if genes elements follow KIR genes naming conventions.

**Examples**
checkKirGenesFormat(c("KIR3DL3", "KIR2DS2", "KIR2DL2"))

checkStatisticalModel  
**Assert statistical model**

**Description**
checkStatisticalModel asserts if object is an existing fit from a model functions such as lm, glm and many others. Containing MiDAS object as its data attribute.

**Usage**
checkStatisticalModel(object)

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

*Value*

Logical indicating if object is an existing fit from a model functions such as lm, glm and many others. Containing MiDAS object as its data attribute. Otherwise raise an error.

---

**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.

---

**convertAlleleToVariable**

*Convert allele numbers to additional variables*

**Description**

convertAlleleToVariable converts input HLA allele numbers to additional variables based on the supplied dictionary.

**Usage**

convertAlleleToVariable(allele, dictionary)

**Arguments**

- **allele**  
  Character vector with HLA allele numbers.
- **dictionary**  
  Path to file containing HLA allele dictionary or a data frame.
countsToVariables

Convert counts table to variables

description

CountsToVariables converts counts table to additional variables.

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 file containing variables dictionary or data frame. See details for further
explanations.

Na.value

Vector of length one specifying value for variables with no matching entry in
dictionary. Default is to use 0.

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:

KIR haplotypes KIR genes to KIR haplotypes dictionary.
Value

Data frame with variable number of columns. First column named "ID" corresponds to "ID" column in counts, further columns hold indicators for converted variables. 1 and 0 code presence and absence of a variable respectively.

Examples

countsToVariables(MiDAS_tut_KIR, "kir_haplotypes")

dfToExperimentMat

Description

Function deletes 'ID' column from a df, then transpose it and sets the column names to values from deleted 'ID' column.

Usage

dfToExperimentMat(df)

Arguments

df

Data frame

Value

Matrix representation of df.

dict_dist_grantham

Grantham distance

Description

Integer vector giving Grantham distance values between pairs of amino acid residues.

Usage

dict_dist_grantham

Format

Named integer vector of length 400.
distGrantham

*Calculate Grantham distance between amino acid sequences*

**Description**

distGrantham calculates normalized Grantham distance between two amino acid sequences. For details on calculations see Grantham R. 1974..

**Usage**

```r
distGrantham(aa1, aa2)
```

**Arguments**

- `aa1` Character vector giving amino acid sequence using one letter codings. Each element must correspond to single amino acid.
- `aa2` Character vector giving amino acid sequence using one letter codings. Each element must correspond to single amino acid.

**Details**

Distance between amino acid sequences is normalized by length of compared sequences. Lengths of `aa1` and `aa2` must be equal.

**Value**

Numeric vector of normalized Grantham distance between `aa1` and `aa2`.

---

experimentMatToDf

*Helper transform experiment matrix to data frame*

**Description**

Function transpose `mat` and inserts column names of input `mat` as a 'ID' column.

**Usage**

```r
experimentMatToDf(mat)
```

**Arguments**

- `mat` Matrix

**Value**

Data frame representation of `mat`. 
filterByFrequency

Filter MiDAS object by frequency

Description
Filter MiDAS object by frequency

Usage
filterByFrequency(
  object,
  experiment,
  lower_frequency_cutoff = NULL,
  upper_frequency_cutoff = NULL,
  carrier_frequency = FALSE
)

Arguments
object MiDAS object.
experiment String specifying experiment.
lower_frequency_cutoff Number giving lower frequency threshold. Numbers greater than 1 are interpreted as the number of feature occurrences, numbers between 0 and 1 as fractions.
upper_frequency_cutoff Number giving upper frequency threshold. Numbers greater than 1 are interpreted as the number of feature occurrences, numbers between 0 and 1 as fractions.
carrier_frequency Logical flag indicating if carrier frequency should be returned.

Value
Filtered MiDAS object.

Examples
filterByFrequency(object = MiDAS_tut_object,
  experiment = "hla_alleles",
  lower_frequency_cutoff = 0.05,
  upper_frequency_cutoff = 0.95,
  carrier_frequency = TRUE)
filterByOmnibusGroups  
*Filter MiDAS object by omnibus groups*

**Description**
Filter MiDAS object by omnibus groups

**Usage**
```r
callByOmnibusGroups(object, experiment, groups)
```

**Arguments**
- **object**: MiDAS object.
- **experiment**: String specifying experiment.
- **groups**: Character vector specifying omnibus groups to select. See `getOmnibusGroups` for more details.

**Value**
Filtered MiDAS object.

**Examples**
```r
callByOmnibusGroups(object = MiDAS_tut_object, 
                    experiment = "hla_aa", 
                    groups = c("A_3", "A_6", "C_1"))
```

---

filterByVariables  
*Filter MiDAS object by features*

**Description**
Filter MiDAS object by features

**Usage**
```r
callByVariables(object, experiment, variables)
```

**Arguments**
- **object**: MiDAS object.
- **experiment**: String specifying experiment.
- **variables**: Character vector specifying features to select.
filterExperimentByFrequency

Value

Filtered MiDAS object.

Examples

filterByVariables(object = MiDAS_tut_object,
experiment = "hla_alleles",
variables = c("A*25:01", "A*26:01", "B*07:02"))

filterExperimentByFrequency

Filter experiment by frequency

Description

Helper function for experiments filtering

Usage

filterExperimentByFrequency(
  experiment,
  carrier_frequency = FALSE,
  lower_frequency_cutoff = NULL,
  upper_frequency_cutoff = NULL
)

## S3 method for class 'matrix'
filterExperimentByFrequency(
  experiment,
  carrier_frequency = FALSE,
  lower_frequency_cutoff = NULL,
  upper_frequency_cutoff = NULL
)

## S3 method for class 'SummarizedExperiment'
filterExperimentByFrequency(
  experiment,
  carrier_frequency = FALSE,
  lower_frequency_cutoff = NULL,
  upper_frequency_cutoff = NULL
)
filterExperimentByVariables

Filter experiment by variable

Description
Helper function for experiments filtering

Usage
filterExperimentByVariables(experiment, variables)

## S3 method for class 'matrix'
filterExperimentByVariables(experiment, variables)

## S3 method for class 'SummarizedExperiment'
filterExperimentByVariables(experiment, variables)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>experiment</td>
<td>Matrix or SummarizedExperiment object.</td>
</tr>
<tr>
<td>variables</td>
<td>Character vector specifying features to choose.</td>
</tr>
</tbody>
</table>

Value
Filtered experiment object.
filterListByElements  
*Filter list by elements*

**Description**

Filter two level list by its secondary elements and remove empty items

**Usage**

```r
filterListByElements(list, elements)
```

**Arguments**

- `list`: A list.
- `elements`: Character vector of elements to keep.

**Value**

List filtered according to `elements` argument.

---

formatResults  
*Pretty format statistical analysis results helper*

**Description**

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

**Usage**

```r
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,
  scroll_box_height = "400px"
)
```
getAAFrequencies

Calculate amino acid frequencies

description

getAAFrequencies calculates amino acid frequencies in amino acid data frame.

usage

getAAFrequencies(aa_variation)
getAlleleResolution

Arguments

aa_variation  Amino acid variation data frame as returned by `hlaToAAVariation`.

Details

Both gene copies are taken into consideration for frequencies calculation, \(\text{frequency} = \frac{n}{2 \times j}\)
where \(n\) is the number of amino acid occurrences and \(j\) is the number of samples in `aa_variation`.

Value

Data frame with each row holding specific amino acid position, it's count and frequency.

Examples

```r
aa_variation <- hlaToAAVariation(MiDAS_tut_HLA)
getAAFrequencies(aa_variation)
```

getAlleleResolution

Infer HLA allele resolution

Description

getAlleleResolution returns the resolution of input HLA allele numbers.

Usage

getAlleleResolution(allele)

Arguments

allele  Character vector with 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.

NA values are accepted and returned as NA.

Value

Integer vector specifying allele resolutions.

Examples

```r
allele <- c("A*01:01", "A*01:02")
getAlleleResolution(allele)
```
getAllelesForAA  
*Get HLA alleles for amino acid position*

**Description**
List HLA alleles and amino acid residues at a given position.

**Usage**
```r
getAllelesForAA(object, aa_pos)
```

**Arguments**
- **object**: MiDAS object.
- **aa_pos**: String specifying gene and amino acid position, example "A_9".

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

**Examples**
```r
getAllelesForAA(object = MiDAS_tut_object, aa_pos = "A_9")
```

getExperimentFrequencies  
*Calculate experiment’s features frequencies*

**Description**
getExperimentFrequencies calculate features frequencies.

**Usage**
```r
getExperimentFrequencies(
  experiment, pop_mul = NULL, carrier_frequency = FALSE, ref = NULL
)
```

```r
## S3 method for class 'matrix'
getExperimentFrequencies(
  experiment,
```

```r
```
getExperimentPopulationMultiplicator

Arguments

experiment Matrix or SummarizedExperiment object.
pop_mul Number by which number of samples should be multiplied to get the population size.
carrier_frequency Logical flag indicating if carrier frequency should be returned.
ref Wide format data frame with first column named "var" holding features matching experiment and specific populations frequencies in following columns. See getReferenceFrequencies for more details.

Value

Data frame with each row holding specific variable, it's count and frequency.

description
g feminists is population multiplicator

description
g feminists is population multiplicator extracts population multiplicator from experiment's metadata.

usage
g feminists is population multiplicator(experiment)

## S3 method for class 'matrix'
g feminists is population multiplicator(experiment)

## S3 method for class 'SummarizedExperiment'
g feminists is population multiplicator(experiment)
getFrequencies

Arguments

experiment Matrix or SummarizedExperiment object.

Value

Experiment’s population multiplicator number.

getExperiments Get available experiments in MiDAS object.

Description

Get available experiments in MiDAS object.

Usage

getExperiments(object)

Arguments

object MiDAS object.

Value

Character vector giving names of experiments in object.

Examples

getExperiments(object = MiDAS_tut_object)

generateFrequencies

Calculate features frequencies for a given experiment in MiDAS object.

Description

Calculate features frequencies for a given experiment in MiDAS object.
getFrequencies

Usage

getFrequencies(
  object,  
  experiment, 
  carrier_frequency = FALSE,
  compare = FALSE,
  ref_pop = list(hla_alleles = c("USA NMDP African American pop 2", "USA NMDP Chinese",  
    "USA NMDP European Caucasian", "USA NMDP Hispanic South or Central American",  
    "USA NMDP Japanese", "USA NMDP North American Amerindian",  
    "USA NMDP South Asian Indian"), kir_genes = c("USA California African American KIR",  
    "USA California Asian American KIR", "USA California Caucasians KIR",  
    "USA California Hispanic KIR")),
  ref = list(hla_alleles = allele_frequencies, kirgenes = kir_frequencies)
)

Arguments

  object          MiDAS object.
  experiment      Matrix or SummarizedExperiment object.
  carrier_frequency Logical flag indicating if carrier frequency should be returned.
  compare         Logical flag indicating if hla_calls frequencies should be compared to refer-  
                   ence frequencies given in ref.
  ref_pop         Named list of character vectors giving names of reference populations in ref to  
                   compare with. Optionally vectors can be named, then those names will be used  
                   as population names. Each vector should correspond to a specific experiment.
  ref             Named list of reference frequencies data frames. Each element should give reference for a specific experiment. See allele_frequencies for an example on how reference frequency data frame should be formatted.

Value

Data frame with features from selected experiment and their corresponding frequencies. Column "term" hold features names, "Counts" hold number of feature occurrences, "Freq" hold feature frequencies. If argument compare is set to TRUE, further columns will hold frequencies in reference populations.

Examples

  # using default reference populations
  getFrequencies(object = MIDAS_tut_object, 
                  experiment = "hla_alleles", 
                  compare = TRUE)

  # using customized set of reference populations
  getFrequencies(  
    object = MIDAS_tut_object, 
    experiment = "hla_alleles", 
    compare = TRUE)
compare = TRUE,
ref_pop = list(
  hla_alleles = c("USA NMDP Chinese", "USA NMDP European Caucasian"),
  ref = list(hla_alleles = allele_frequencies)
)

---

**getFrequencyMask**  
*Helper function for filtering frequency data frame*

**Description**

Helper function for filtering frequency data frame

**Usage**

```r
getFrequencyMask(
  df,
  lower_frequency_cutoff = NULL,
  upper_frequency_cutoff = NULL
)
```

**Arguments**

- **df**  
  Data frame as returned by `getExperimentFrequencies`.

- **lower_frequency_cutoff**  
  Positive number or NULL. Numbers greater than 1 are interpreted as number of feature occurrences, numbers between 0 and 1 as fractions.

- **upper_frequency_cutoff**  
  Positive number or NULL. Numbers greater than 1 are interpreted as number of feature occurrences, numbers between 0 and 1 as fractions.

**Value**

Character vector containing names of variables after filtration.
getHlaCalls

Get HLA calls from MiDAS object.

Description
Get HLA calls from MiDAS object.

Usage
getHlaCalls(object)

Arguments

  object MiDAS object.

Value
HLA calls data frame.

Examples
getHlaCalls(object = MiDAS_tut_object)

getHlaCallsGenes

Get HLA calls genes

Description
getHlaCallsGenes get’s genes found in HLA calls.

Usage
getHlaCallsGenes(hla_calls)

Arguments

  hla_calls HLA calls data frame, as returned by readHlaCalls function.

Value
Character vector of genes in hla_calls.
getHlaFrequencies  
*Calculate HLA allele frequencies*

**Description**

getHlaFrequencies calculates allele frequencies in HLA calls data frame.

**Usage**

```r
getHlaFrequencies(
  hla_calls,
  carrier_frequency = FALSE,
  compare = FALSE,
  ref_pop = c("USA NMDP African American pop 2", "USA NMDP Chinese",
               "USA NMDP European Caucasian", "USA NMDP Hispanic South or Central American",
               "USA NMDP Japanese", "USA NMDP North American Amerindian",
               "USA NMDP South Asian Indian"),
  ref = allele_frequencies
)
```

**Arguments**

- `hla_calls`  
  HLA calls data frame, as returned by `readHlaCalls` function.

- `carrier_frequency`  
  Logical flag indicating if carrier frequency should be returned.

- `compare`  
  Logical flag indicating if `hla_calls` frequencies should be compared to reference frequencies given in `ref`.

- `ref_pop`  
  Character vector giving names of reference populations in `ref` to compare with. Optionally vector can be named, then those names will be used as population names.

- `ref`  
  Data frame giving reference allele frequencies. See `allele_frequencies` for an example.

**Details**

Both gene copies are taken into consideration for frequencies calculation, 
\[ \text{frequency} = \frac{n}{2 \times j} \]
where \( n \) is the number of allele occurrences and \( j \) is the number of samples in `hla_calls`.

**Value**

Data frame with each row holding HLA allele, it’s count and frequency.

**Examples**

```r
getHlaFrequencies(MiDAS_tut_HLA)
```
**getHlaKirInteractions**  
*Get HLA - KIR interactions*

**Description**

getHlaKirInteractions calculate presence-absence matrix of HLA - KIR interactions.

**Usage**

```r
getHlaKirInteractions(
  hla_calls,  
  kir_calls, 
  interactions_dict = system.file("extdata", "Match_counts_hla_kir_interactions.txt", package = "midasHLA")
)
```

**Arguments**

- **hla_calls**: HLA calls data frame, as returned by `readHlaCalls` function.
- **kir_calls**: KIR calls data frame, as returned by `readKirCalls` function.
- **interactions_dict**: Path to HLA - KIR interactions dictionary.

**Details**

- **hla_calls**: are first reduced to all possible resolutions and converted to additional variables, such as G groups, using dictionaries shipped with the package.
- **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 (e.g. "C2 & KIR2DL1" will match all samples with C2 and KIR2DL1). The package is shipped with an interactions file based on Pende et al., 2019.

**Value**

Data frame with variable number of columns. First column named "ID" corresponds to "ID" column in counts, further columns hold indicators for HLA - KIR interactions. 1 and 0 code presence and absence of a variable respectively.

**Examples**

```r
getHlaKirInteractions(
  hla_calls = MiDAS_tut_HLA, 
  kir_calls = MiDAS_tut_KIR, 
  interactions_dict = system.file("extdata", "Match_counts_hla_kir_interactions.txt", package = "midasHLA")
)
```
getKirCalls

Get KIR calls from MiDAS object.

Description
Get KIR calls from MiDAS object.

Usage
getKirCalls(object)

Arguments
object MiDAS object.

Value
KIR calls data frame.

Examples
getKirCalls(object = MiDAS_tut_object)

getKIRFrequencies

Calculate KIR genes frequencies

Description
getKIRFrequencies calculates KIR genes frequencies in KIR calls data frame.

Usage
getKIRFrequencies(kir_calls)

Arguments
kir_calls KIR calls data frame, as returned by readKirCalls function.

Value
Data frame with each row holding KIR gene, it’s count and frequency.

Examples
getKIRFrequencies(MiDAS_tut_KIR)
**getObjectDetails**

*Get attributes of statistical model object*

**Description**

`getObjectDetails` extracts some of the statistical model object attributes that are needed for `runMiDAS` internal calculations.

**Usage**

`getObjectDetails(object)`

**Arguments**

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

**Value**

List with following elements:

- `call` : Object’s call
- `formula_vars` : Character containing names of variables in object formula
- `data` : MiDAS object associated with model

**getOmnibusGroups**

*Get omnibus groups from MiDAS object.*

**Description**

Get omnibus groups from MiDAS object.

**Usage**

`getOmnibusGroups(object, experiment)`

**Arguments**

- `object` : MiDAS object.
- `experiment` : String specifying experiment.

**Details**

For some experiments features can be naturally divided into groups (here called omnibus groups). For example, in “hla_66” experiment features can be grouped by amino acid position (“B_46_E”, “B_46_A”) can be grouped into B_46 group). Such groups can be then used to perform omnibus test, see `runMiDAS` for more details.
getValueFrequencies

List of omnibus groups for a given experiment.

Examples

getOmnibusGroups(object = MiDAS_tut_object,
                 experiment = "hla_aa")

getPlaceholder

Get placeholder name from MiDAS object.

Description

Get placeholder name from MiDAS object.

Usage

getPlaceholder(object)

Arguments

object MiDAS object.

Value

String giving name of placeholder.

Examples

getPlaceholder(object = MiDAS_tut_object)

getReferenceFrequencies

Helper transforming reference frequencies

Description

Helper transforming reference frequencies

Usage

generateFrequencies(ref, pop, carrier_frequency = FALSE)
getVariableAAPos

Arguments

- **ref**  
  Long format data frame with three columns "var", "population", "frequency".

- **pop**  
  Character giving names of populations to include

- **carrier_frequency**  
  Logical indicating if carrier frequency should be returned instead of frequency. Carrier frequency is calculated based on Hardy-Weinberg equilibrium model.

Value

Wide format data frame with population frequencies as columns.

getVariableAAPos  
*Find variable positions in sequence alignment*

Description

getVariableAAPos finds variable amino acid positions in protein sequence alignment.

Usage

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

Arguments

- **alignment**  
  Matrix containing amino acid level alignment, as returned by `readHlaAlignments`,

- **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 as variable or not. This can be achieved by passing appropriate regular expression in `varchar`. Eg. when `varchar = "[A-Z]\"` occurrence 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

```r
alignment <- readHlaAlignments(gene = "TAP1")
getVariableAAPos(alignment)
```
hasTidyMethod  

*Check if tidy method for class exist*

**Description**

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

**Usage**

```r
hasTidyMethod(class)
```

**Arguments**

- `class`  
  String giving object class.

**Value**

Logical indicating if there is a tidy method for a given class.

---

hlaAlignmentGrantham  

*Helper function returning alignment for Grantham distance calculations*

**Description**

Helper function returning alignment for Grantham distance calculations

**Usage**

```r
hlaAlignmentGrantham(gene, aa_sel = 2:182)
```

**Arguments**

- `gene`  
  Character vector specifying HLA gene.

- `aa_sel`  
  Numeric vector specifying amino acids that should be extracted.

**Value**

HLA alignment processed for grantham distance calculation. Processing includes extracting specific amino acids, masking indels, gaps and stop codons.
hlaCallsGranthamDistance

*Calculate Grantham distance between HLA alleles*

Description

`hlaCallsGranthamDistance` calculate Grantham distance between two HLA alleles of a given, using original formula by Grantham R. 1974.

Usage

```r
hlaCallsGranthamDistance(
  hla_calls,
  genes = c("A", "B", "C"),
  aa_selection = "binding_groove"
)
```

Arguments

- `hla_calls`: HLA calls data frame, as returned by `readHlaCalls` function.
- `genes`: Character vector specifying genes for which allelic distance should be calculated.
- `aa_selection`: String specifying variable region in peptide binding groove which should be considered for Grantham distance calculation. Valid choices includes: "binding_groove", "B_pocket", "F_pocket". See details for more information.

Details

Grantham distance is calculated only for class I HLA alleles. First exons forming the variable region in the peptide binding groove are selected. Here we provide option to choose either "binding_groove" - exon 2 and 3 (positions 1-182 in IMGT/HLA alignments, however here we take 2-182 as many 1st positions are missing), "B_pocket" - residues 7, 9, 24, 25, 34, 45, 63, 66, 67, 70, 99 and "F_pocket" - residues 77, 80, 81, 84, 95, 116, 123, 143, 146, 147. Then all the alleles containing gaps, stop codons or indels are discarded. Finally distance is calculated for each pair.

See Robinson J. 2017. for more details on the choice of exons 2 and 3.

Value

Data frame of normalized Grantham distances between pairs of alleles for each specified HLA gene. First column (ID) is the same as in `hla_calls`, further columns are named as given by `genes`.

Examples

```r
hlaCallsGranthamDistance(MiDAS_tut_HLA, genes = "A")
```
**hlaCallsToCounts**  
*Transform HLA calls to counts table*

**Description**

*hlaCallsToCounts* converts HLA calls data frame into a counts table.

**Usage**

```r
hlaCallsToCounts(hla_calls, check_hla_format = TRUE)
```

**Arguments**

- **hla_calls**: HLA calls data frame, as returned by `readHlaCalls` function.
- **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**

HLA allele counts data frame. First column holds samples ID’s, further columns, corresponding to specific alleles, give information on the number of their occurrences in each sample.

**hlaToAAVariation**  
*Generate amino acid variation matrix*

**Description**

*hlaToAAVariation* convert HLA calls data frame to a matrix of variable amino acid positions.

**Usage**

```r
hlaToAAVariation(hla_calls, indels = TRUE, unkchar = FALSE, as_df = TRUE)
```

**Arguments**

- **hla_calls**: HLA calls data frame, as returned 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.
- **as_df**: Logical indicating if data frame should be returned. Otherwise a matrix is returned.
**hlaToVariable**

**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 processes alignments for all HLA genes found in hla_calls.
Variable amino acid position uses protein alignments from EBI database.

**Value**
Matrix or data frame containing variable amino acid positions. Rownames corresponds to ID column in hla_calls, and colnames to alignment positions. If no variation is found one column matrix filled with NA’s is returned.

**Examples**
```
hlaToAAVariation(MiDAS_tut_HLA)
```

---

**hlaToVariable** *Convert HLA calls to variables*

**Description**
hlaToVariable converts HLA calls data frame to additional variables.

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

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hla_calls</td>
<td>HLA calls data frame, as returned by readHlaCalls function.</td>
</tr>
<tr>
<td>dictionary</td>
<td>Path to file containing HLA allele dictionary or a data frame.</td>
</tr>
<tr>
<td>reduce</td>
<td>Logical indicating if function should try to reduce allele resolution when no matching entry in the dictionary is found. See details.</td>
</tr>
<tr>
<td>na.value</td>
<td>Vector of length one specifying value for alleles with no matching entry in dictionary. Default is to use 0.</td>
</tr>
<tr>
<td>nacols.rm</td>
<td>Logical indicating if result columns that contain only NA should be removed.</td>
</tr>
</tbody>
</table>
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.

dictionary can be also used to access dictionaries shipped with the package. They can be referred to by using one of the following strings:

"allele_HLA_Bw" Translates HLA-B alleles together with A*23, A*24 and A*32 into Bw4 and Bw6 allele groups. 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, Bw4 group is assigned.

"allele_HLA-B_only_Bw" Translates HLA-B alleles (without A*23, A*24 and A*32) into Bw4 and Bw6 allele groups.

"allele_HLA-C_C1-2" Translates HLA-C alleles into C1 and C2 allele groups.

"allele_HLA_supertype" Translates HLA-A and HLA-B alleles into supertypes, a classification that group HLA alleles based on peptide binding specificities.

"allele_HLA_Ggroup" Translates HLA alleles into 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 match more than one G group, here this ambiguity was removed by deleting matching with DRB5*01:01:01G group.

reduce control if conversion should happen in a greedy way, such that if some HLA number cannot be converted, it's resolution is reduced by 2 and another attempt is taken. This process stops when alleles cannot be further reduced or all have been successfully converted.

Value

Data frame with variable number of columns. First column named "ID" corresponds to "ID" column in hla_calls, further columns holds converted HLA variables.

Examples

hlaToVariable(MiDAS_tut_HLA, dictionary = "allele_HLA_supertype")

---

**HWETest**

Test for Hardy Weinberg equilibrium

---

**Description**

Test experiment features for Hardy Weinberg equilibrium.
Usage

`HWETest(
  object,
  experiment = c("hla_alleles", "hla_aa", "hla_g_groups", "hla_supertypes",
  "hla_NK_ligands"),
  HWE_group = NULL,
  HWE_cutoff = NULL,
  as.MiDAS = FALSE
)
``

Arguments

- **object**: MiDAS object.
- **experiment**: String specifying experiment to test. Valid values includes "hla_alleles", "hla_aa", "hla_g_groups", "hla_supertypes", "hla_NK_ligands".
- **HWE_group**: Expression defining samples grouping to test for Hardy Weinberg equilibrium. By default samples are not grouped.
- **HWE_cutoff**: Number specifying p-value threshold. When `HWE_group` is specified both groups are thresholded.
- **as.MiDAS**: Logical flag indicating if MiDAS object should be returned.

Details

Setting `as.MiDAS` to TRUE will filter MiDAS object based on p-value cut-off given by `HWE_cutoff`.

Value

Data frame with Hardy Weinberg Equilibrium test results or a filtered MiDAS object.

Examples

```r
# create MiDAS object
midas <- prepareMiDAS(hla_calls = MiDAS_tut_HLA,
  colData = MiDAS_tut_pheno,
  experiment = "hla_alleles"
)

# get HWE p-values as data frame
HWETest(midas, experiment = "hla_alleles")

# get HWE in groups defined by disease status
# grouping by `disease == 1` will divide samples into two groups:
# `disease == 1` and `not disease == 1`
HWETest(midas, experiment = "hla_alleles", HWE_group = disease == 1)

# filter MiDAS object by HWE test p-value
HWETest(midas, experiment = "hla_alleles", HWE_cutoff = 0.05, as.MiDAS = TRUE)
```
**isCharacterOrNULL**  
*Check if object is character vector or NULL*

**Description**

isCharacterOrNULL checks if the object is a character vector or NULL.

**Usage**

```r
isCharacterOrNULL(x)
```

**Arguments**

- `x`: object to test.

**Value**

Logical indicating if object is character vector or NULL.

---

**isClass**  
*Check if object is of class x*

**Description**

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

**Usage**

```r
isClass(x, class)
```

**Arguments**

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

**Value**

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

---

**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.
isCountsOrZeros  
*Check if vector contains only counts or zeros*

**Description**

isCountsOrZeros checks if vector contains only positive integers or zeros.

**Usage**

```r
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.

---

isExperimentCountsOrZeros  
*Check if frequencies can be calculated for an experiment*

**Description**

isExperimentCountsOrZeros checks if experiment contains only positive integers or zeros.

**Usage**

```r
isExperimentCountsOrZeros(x, na.rm = TRUE)
```

**Arguments**

- **x**: Matrix or SummarizedExperiment object.
- **na.rm**: Logical indicating if NA values should be accepted.

**Value**

Logical indicating if x contains only positive integers or zeros.
isExperimentInheritanceModelApplicable

Check if experiment is inheritance model applicable

Description

isExperimentInheritanceModelApplicable checks the experiment's metadata for the presence of the "inheritance_model_applicable" flag, indicating if the inheritance model can be applied.

Usage

isExperimentInheritanceModelApplicable(experiment)

## S3 method for class 'matrix'
isExperimentInheritanceModelApplicable(experiment)

## S3 method for class 'SummarizedExperiment'
isExperimentInheritanceModelApplicable(experiment)

Arguments

experiment: Matrix or SummarizedExperiment object.

Value

Logical flag.

isFlagOrNULL

Check if object is flag or NULL

Description

isFlagOrNULL checks if the object is a 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.
isNumberOrNULL

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.

isStringOrNULL

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.
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

iterativeLRT

Iterative likelihood ratio test

Description

iterativeLRT performs likelihood ratio test in an iterative manner over groups of variables given in omnibus_groups.

Usage

iterativeLRT(object, placeholder, omnibus_groups)

Arguments

object An existing fit from a model function such as lm, glm and many others.
placeholder String specifying term to substitute with value from x. Ignored if set to NULL.
omnibus_groups List of character vectors giving sets of variables for which omnibus test should be applied.

Value

Data frame containing summarised likelihood ratio test results.
**iterativeModel**

*Iteratively evaluate model for different variables*

**Description**

Information about variable statistic from each model is extracted using tidy function.

**Usage**

iterativeModel(object, placeholder, variables, exponentiate = FALSE)

**Arguments**

- **object**: An existing fit from a model function such as lm, glm and many others.
- **placeholder**: String specifying term to substitute with value from x. Ignored if set to NULL.
- **variables**: Character vector specifying variables to use in association tests.
- **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.

**Value**

Tibble containing per variable summarised model statistics. The exact output format is model-dependent and controlled by model’s dedicated tidy function.

---

**kableResults**

*Create association analysis results table in HTML or LaTeX*

**Description**

kableResults convert results table (runMiDAS output) to HTML or LaTeX format.

**Usage**

kableResults(
  results, colnames = NULL,
  header = "MiDAS analysis results",
  pvalue_cutoff = NULL,
  format = getOption("knitr.table.format"),
  scroll_box_height = "400px"
)
**Arguments**

- **results**: Tibble as returned by `runMiDAS`.
- **colnames**: Character vector of form `c("new_name" = "old_name")`, used to rename `results` colnames.
- **header**: String specifying results table header.
- **pvalue_cutoff**: Number specifying p-value cutoff for results to be included in output. If `NULL` no filtering is done.
- **format**: String "latex" or "html".
- **scroll_box_height**: A character string indicating the height of the table.

**Value**

Association analysis results table in HTML or LaTeX.

**Examples**

```r
midas <- prepareMiDAS(hla_calls = MiDAS_tut_HLA,
                        colData = MiDAS_tut_pheno,
                        experiment = "hla_alleles")
object <- lm(disease ~ term, data = midas)
res <- runMiDAS(object, experiment = "hla_alleles", inheritance_model = "additive")
kableResults(results = res,
              colnames = c("HLA allele" = "allele"))
```

---

**kir_frequencies**

*KIR genes frequencies scraped from allelefrequencies.net*

**Description**

Accessed on 28.08.20

**Usage**

`kir_frequencies`

**Format**

A data frame with 3744 rows and 3 variables:

- **var**: allele number, character
- **population**: reference population name, character
- **frequency**: KIR genes carrier frequency in reference population, float
Details

A dataset containing KIR genes frequencies across 16 genes. For details visit the search results page in the allelefrequencies.net database website.

Source

www.allelefrequencies.net

---

lapply_tryCatch  
**lapply with tryCatch routine**

Description

Used to run function iteratively over list, while using tryCatch to catch warnings and errors to finally present a summary of issues rather than error on each and every one. Used in iterativeLRT and iterativeModel.

Usage

lapply_tryCatch(X, FUN, err_res, ...)

Arguments

- **X**  
a vector (atomic or list) or an *expression* object. Other objects (including classed objects) will be coerced by base::as.list.
- **FUN**  
the function to be applied to each element of X; see ‘Details’. In the case of functions like +, %*%, the function name must be backquoted or quoted.
- **err_res**  
Function creating a result that should be output in case of error.
- **...**  
optional arguments to FUN.

Value

List of elements as returned by FUN.
listMiDASDictionaries  

List HLA alleles dictionaries

Description

listMiDASDictionaries lists dictionaries shipped with the MiDAS package. See hlaToVariable for more details on dictionaries.

Usage

listMiDASDictionaries(pattern = "allele", 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 giving names of available HLA alleles dictionaries.

---

LRTest  

Likelihood ratio test

Description

LRTest carry out an asymptotic likelihood ratio test for two models.

Usage

LRTest(mod0, mod1)

Arguments

- **mod0**: An existing fit from a model function such as lm, glm and many others.
- **mod1**: Object of the same class as mod0 with extra terms included.

Details

mod0 have to be a reduced version of mod1. See examples.
### Value

Data frame with the results of likelihood ratio test of the supplied models. Column `term` holds new variables appearing in `mod1`, `df` difference in degrees of freedom between models, `logLik` difference in log likelihoods, `statistic Chisq` statistic and `p.value` corresponding p-value.

---

### Description

The MiDAS class is a `MultiAssayExperiment` object containing data and metadata required for MiDAS analysis.

Valid MiDAS object must have unique features names across all experiments and colData. It’s metadata list needs to have a `placeholder` element, which is a string specifying name of column in colData used when defining statistical model for downstream analyses (see `runMiDAS` for more details). Optionally the object’s metadata can also store `hla_calls` and `kir_calls` data frames (see `prepareMiDAS` for more details).

### Usage

```r
## S4 method for signature 'MiDAS'
getExperiments(object)

## S4 method for signature 'MiDAS'
getHlaCalls(object)

## S4 method for signature 'MiDAS'
getKirCalls(object)

## S4 method for signature 'MiDAS'
getPlaceholder(object)

## S4 method for signature 'MiDAS'
getOmnibusGroups(object, experiment)

## S4 method for signature 'MiDAS'
getFrequencies(
  object, experiment, carrier_frequency = FALSE, compare = FALSE,
  ref_pop = list(hla_alleles = c("USA NMDP African American pop 2", "USA NMDP Chinese", 
                              "USA NMDP European Caucasian", "USA NMDP Hispanic South or Central American", 
                              "USA NMDP Japanese", "USA NMDP North American Amerindian", 
                              "USA NMDP South Asian Indian"), kir_genes = c("USA California African American KIR", 
                                                           "USA California Chinese KIR", 
                                                           "USA California European Caucasian KIR", 
                                                           "USA California Hispanic South or Central American KIR", 
                                                           "USA California Japanese KIR", 
                                                           "USA California North American Amerindian KIR", 
                                                           "USA California South Asian Indian KIR")))
```
MiDAS-class

"USA California Asian American KIR", "USA California Caucasians KIR", "USA California Hispanic KIR"),
ref = list(hla_alleles = allele_frequencies, kir_genes = kir_frequencies)
)

## S4 method for signature 'MiDAS'
filterByFrequency(
  object,
  experiment,
  lower_frequency_cutoff = NULL,
  upper_frequency_cutoff = NULL,
  carrier_frequency = FALSE
)

## S4 method for signature 'MiDAS'
filterByOmnibusGroups(object, experiment, groups)

## S4 method for signature 'MiDAS'
filterByVariables(object, experiment, variables)

## S4 method for signature 'MiDAS'
getAllelesForAA(object, aa_pos)

Arguments

object MiDAS object.
experiment String specifying experiment.
carrier_frequency Logical flag indicating if carrier frequency should be returned.
compare Logical flag indicating if hla_calls frequencies should be compared to reference frequencies given in ref.
ref_pop Named list of character vectors giving names of reference populations in ref to compare with. Optionally vectors can be named, then those names will be used as population names. Each vector should correspond to a specific experiment.
ref Named list of reference frequencies data frames. Each element should give reference for a specific experiment. See allele_frequencies for an example on how reference frequency data frame should be formatted.
lower_frequency_cutoff Number giving lower frequency threshold. Numbers greater than 1 are interpreted as the number of feature occurrences, numbers between 0 and 1 as fractions.
upper_frequency_cutoff Number giving upper frequency threshold. Numbers greater than 1 are interpreted as the number of feature occurrences, numbers between 0 and 1 as fractions.
groups Character vector specifying omnibus groups to select. See getOmnibusGroups for more details.
variables Character vector specifying features to select.
aa_pos String specifying gene and amino acid position, example "A_9".

Value
Instance of class MiDAS

midasToWide Transform MiDAS to wide format data.frame

Description
Transform MiDAS to wide format data.frame

Usage
midasToWide(object, experiment)

Arguments
object Object of class MiDAS
experiment Character specifying experiments to include

Value
Data frame representation of MiDAS object. Consecutive columns holds values of variables from MiDAS’s experiments and colData. The metadata associated with experiments is not preserved.

MiDAS_tut_HLA MiDAS tutorial HLA data

Description
Example HLA calls data used in MiDAS tutorial

Usage
MiDAS_tut_HLA
MiDAS_tut_KIR

Format

Data frame with 1000 rows and 19 columns. First column holds samples ID’s, following columns holds HLA alleles calls for different genes.

<table>
<thead>
<tr>
<th>ID</th>
<th>Character sample ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_1</td>
<td>Character</td>
</tr>
<tr>
<td>A_2</td>
<td>Character</td>
</tr>
<tr>
<td>B_1</td>
<td>Character</td>
</tr>
<tr>
<td>B_2</td>
<td>Character</td>
</tr>
<tr>
<td>C_1</td>
<td>Character</td>
</tr>
<tr>
<td>C_2</td>
<td>Character</td>
</tr>
<tr>
<td>DPA1_1</td>
<td>Character</td>
</tr>
<tr>
<td>DPA1_2</td>
<td>Character</td>
</tr>
<tr>
<td>DPB1_1</td>
<td>Character</td>
</tr>
<tr>
<td>DPB1_2</td>
<td>Character</td>
</tr>
<tr>
<td>DQA1_1</td>
<td>Character</td>
</tr>
<tr>
<td>DQA1_2</td>
<td>Character</td>
</tr>
<tr>
<td>DQB1_1</td>
<td>Character</td>
</tr>
<tr>
<td>DQB1_2</td>
<td>Character</td>
</tr>
<tr>
<td>DRA_1</td>
<td>Character</td>
</tr>
<tr>
<td>DRA_2</td>
<td>Character</td>
</tr>
<tr>
<td>DRB1_1</td>
<td>Character</td>
</tr>
<tr>
<td>DRB1_2</td>
<td>Character</td>
</tr>
</tbody>
</table>

---

MiDAS_tut_KIR  MiDAS tutorial KIR data

Description

Example KIRR presence/absence data used in MiDAS tutorial

Usage

MiDAS_tut_KIR
MiDAS_tut_object

Format

Data frame with 1000 rows and 17 columns. First column holds samples ID’s, following columns holds presence/absence indicators for different KIR genes.

<table>
<thead>
<tr>
<th>ID</th>
<th>Character sample ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>KIR3DL3</td>
<td>Integer</td>
</tr>
<tr>
<td>KIR2DS2</td>
<td>Integer</td>
</tr>
<tr>
<td>KIR2DL2</td>
<td>Integer</td>
</tr>
<tr>
<td>KIR2DL3</td>
<td>Integer</td>
</tr>
<tr>
<td>KIR2DP1</td>
<td>Integer</td>
</tr>
<tr>
<td>KIR2DL1</td>
<td>Integer</td>
</tr>
<tr>
<td>KIR3DP1</td>
<td>Integer</td>
</tr>
<tr>
<td>KIR2DL4</td>
<td>Integer</td>
</tr>
<tr>
<td>KIR3DL1</td>
<td>Integer</td>
</tr>
<tr>
<td>KIR3DS1</td>
<td>Integer</td>
</tr>
<tr>
<td>KIR2DL5</td>
<td>Integer</td>
</tr>
<tr>
<td>KIR2DS3</td>
<td>Integer</td>
</tr>
<tr>
<td>KIR2DS5</td>
<td>Integer</td>
</tr>
<tr>
<td>KIR2DS4</td>
<td>Integer</td>
</tr>
<tr>
<td>KIR2DS1</td>
<td>Integer</td>
</tr>
<tr>
<td>KIR3DL2</td>
<td>Integer</td>
</tr>
</tbody>
</table>

---

MiDAS_tut_object  MiDAS tutorial MiDAS object

Description

Example MiDAS object created with data used in MiDAS tutorial: MiDAS_tut_HLA, MiDAS_tut_KIR, MiDAS_tut_pheno. Used in code examples and unit tests.

Usage

MiDAS_tut_object

Format

MiDAS object with following experiments defined:

- hla_alleles SummarizedExperiment with 447 rows and 1000 columns
- hla_aa SummarizedExperiment with 1223 rows and 1000 columns
- hla_g_groups SummarizedExperiment with 46 rows and 1000 columns
MiDAS_tut_pheno

### hla_supertypes
SummarizedExperiment with 12 rows and 1000 columns

### hla_NK_ligands
SummarizedExperiment with 5 rows and 1000 columns

### kir_genes
SummarizedExperiment with 16 rows and 1000 columns

### kir_haplotypes
SummarizedExperiment with 6 rows and 1000 columns

### hla_kir_interactions
SummarizedExperiment with 29 rows and 1000 columns

### hla_divergence
Matrix with 4 rows and 1000 columns

### hla_het
SummarizedExperiment with 9 rows and 1000 columns

---

**MiDAS_tut_pheno**  
*MiDAS tutorial phenotype data*

**Description**
Example phenotype data used in MiDAS tutorial

**Usage**
MiDAS_tut_pheno

**Format**
Data frame with 1000 rows and 4 columns.

<table>
<thead>
<tr>
<th>ID</th>
<th>Character sample ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>disease</td>
<td>Integer</td>
</tr>
<tr>
<td>lab_value</td>
<td>Numeric</td>
</tr>
<tr>
<td>outcome</td>
<td>Integer</td>
</tr>
</tbody>
</table>

---

**objectHasPlaceholder**  
*Check if placeholder is present in object formula*

**Description**

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

**Usage**
objectHasPlaceholder(object, placeholder)

**Arguments**

<table>
<thead>
<tr>
<th>object</th>
<th>statistical model to test.</th>
</tr>
</thead>
<tbody>
<tr>
<td>placeholder</td>
<td>string specifying name of placeholder.</td>
</tr>
</tbody>
</table>

**Value**
Logical indicating if placeholder is present in object formula.
omnibusTest  

Description

*OmnibusTest* calculates overall p-value for linear combination of variables using likelihood ratio test.

Usage

```r
omnibusTest(
  object,
  omnibus_groups,
  placeholder = "term",
  correction = "bonferroni",
  n_correction = NULL
)
```

Arguments

- **object**: An existing fit from a model function such as `lm`, `glm` and many others.
- **omnibus_groups**: List of character vectors giving sets of variables for which omnibus test should be applied.
- **placeholder**: String specifying term in object’s formula which should be substituted with variables during analysis.
- **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. For Bonferroni correction it is possible to specify a number lower than the number of comparisons being made. This is useful in cases when knowledge about the biology or redundancy of alleles reduces the need for correction. For other methods it must be at least equal to the number of comparisons being made; only set this (to non-default) when you know what you are doing!

Details

Likelihood ratio test is conducted by comparing a model given in an object with an extended model, that is created by including the effect of variables given in `variables` as their linear combination.

Value

Data frame with columns:

- "group" Omnibus group name
• "term" Elements of omnibus group added to base model
• "df" Difference in degrees of freedom between base and extended model
• "logLik" Difference in log likelihoods between base and extended model
• "statistic" Chisq statistic
• "p.value" P-value
• "p.adjusted" Adjusted p-value

Examples

midas <- prepareMiDAS(hla_calls = MiDAS_tut_HLA,
colData = MiDAS_tut_pheno,
experiment = "hla_aa")

# define base model
object <- lm(disease ~ term, data = midas)
omnibusTest(object,
onnibus_groups = list(
  A_29 = c("A_29_D", "A_29_A"),
  A_43 = c("A_43_Q", "A_43_R")
))

prepareMiDAS  
Construct a MiDAS object

Description

prepareMiDAS transform HLA alleles calls and KIR calls according to selected experiments creating a MiDAS object.

Usage

prepareMiDAS(
  hla_calls = NULL,
kir_calls = NULL,
colData,
experiment = c("hla_alleles", "hla_aa", "hla_g_groups", "hla_supertypes",
  "hla_NK_ligands", "kir_genes", "kir_haplotypes", "hla_kir_interactions",
  "hla_divergence", "hla_het", "hla_custom", "kir_custom"),
placeholder = "term",
lower_frequency_cutoff = NULL,
upper_frequency_cutoff = NULL,
indels = TRUE,
unkchar = FALSE,
hla_divergence_aa_selection = "binding_groove",
hla_het_resolution = 8,
hla_dictionary = NULL,
kir_dictionary = NULL
)


Arguments

hla_calls  HLA calls data frame, as returned by `readHlaCalls` function.

kir_calls  KIR calls data frame, as returned by `readKirCalls` function.

colData  Data frame holding additional variables like phenotypic observations or covariates. It have to contain ‘ID’ column holding samples identifiers corresponding to identifiers in hla_calls and kir_calls. Importantly rows of hla_calls and kir_calls without corresponding phenotype are discarded.

experiment  Character vector indicating analysis type for which data should be prepared. Valid choices are "hla_alleles", "hla_aa", "hla_g_groups", "hla_supertypes", "hla_NK_ligands", "kir_genes", "hla_kir_interactions", "hla_divergence", "hla_het". See details for further explanations.

placeholder  String giving name for dummy variable inserted to colData. This variable can be than used to define base statistical model used by `runMiDAS`.

lower_frequency_cutoff  Number giving lower frequency threshold. Numbers greater than 1 are interpreted as the number of feature occurrences, numbers between 0 and 1 as fractions.

upper_frequency_cutoff  Number giving upper frequency threshold. Numbers greater than 1 are interpreted as the number of feature occurrences, numbers between 0 and 1 as fractions.

indels  Logical indicating whether indels should be considered when checking amino acid variability in 'hla_aa' experiment.

unkchar  Logical indicating whether unknown characters in the alignment should be considered when checking amino acid variability in 'hla_aa' experiment.

hla_divergence_aa_selection  String specifying variable region in peptide binding groove which should be considered for Grantham distance calculation. Valid choices includes: "binding_groove", "B_pocket", "F_pocket". See details for more information.

hla_het_resolution  Number specifying HLA alleles resolution used to calculate heterogeneity in "hla_het" experiment.

hla_dictionary  Data frame giving HLA allele dictionary used in 'hla_custom' experiment. See `hlaToVariable` for more details.

kir_dictionary  Data frame giving KIR genes dictionary used in 'kir_custom' experiment. See `countsToVariables` for more details.

Details

experiment specifies analysis types for which hla_calls and kir_call should be prepared.

'hla_alleles' hla_calls are transformed to counts matrix describing number of allele occurrences for each sample. This experiment is used to test associations on HLA alleles level.

'hla_aa' hla_calls are transformed to a matrix of variable amino acid positions. See `hlaToAAVariation` for more details. This experiment is used to test associations on amino acid level.
"hla_g_groups" hla_calls are translated into HLA G groups and transformed to matrix describing number of G group occurrences for each sample. See hlaToVariable for more details. This experiment is used to test associations on HLA G groups level.

"hla_supertypes" hla_calls are translated into HLA supertypes and transformed to matrix describing number of G group occurrences for each sample. See hlaToVariable for more details. This experiment is used to test associations on HLA supertypes level.

"hla_NK_ligands" hla_calls are translated into NK ligands, which includes HLA Bw4/Bw6 and HLA C1/C2 groups and transformed to matrix describing number of their occurrences for each sample. See hlaToVariable for more details. This experiment is used to test associations on HLA NK ligands level.

"kir_genes" kir_calls are transformed to counts matrix describing number of KIR gene occurrences for each sample. This experiment is used to test associations on KIR genes level.

"hla_kir_interactions" hla_calls and kir_calls are translated to HLA - KIR interactions as defined in Pende et al., 2019. See getHlaKirInteractions for more details. This experiment is used to test associations on HLA - KIR interactions level.

"hla_divergence" Grantham distance for class I HLA alleles is calculated based on hla_calls using original formula by Grantham R. 1974. See hlaCallsGranthamDistance for more details. This experiment is used to test associations on HLA divergence level measured by Grantham distance.

"hla_het" hla_calls are transformed to heterozygosity status, where 1 designates a heterozygote and 0 homozygote. Heterozygosity status is calculated only for classical HLA genes (A, B, C, DQA1, DQB1, DRA, DRB1, DPA1, DPB1). This experiment is used to test associations on HLA divergence level measured by heterozygosity.

Value

Object of class MiDAS

Examples

midas <- prepareMiDAS(hla_calls = MiDAS_tut_HLA,
                       kir_calls = MiDAS_tut_KIR,
                       colData = MiDAS_tut_pheno,
                       experiment = "hla_alleles")

---

prepareMiDAS_hla_aa Prepare MiDAS data on HLA amino acid level

Description

Prepare MiDAS data on HLA amino acid level

Usage

prepareMiDAS_hla_aa(hla_calls, indels = TRUE, unkchar = FALSE, ...)
prepareMiDAS_hla_alleles

Prepare MiDAS data on HLA allele level

Description

Prepare MiDAS data on HLA allele level

Usage

prepareMiDAS_hla_alleles(hla_calls, ...)

Arguments

hla_calls HLA calls data frame, as returned by readHlaCalls function.

... Not used

Value

Matrix
prepareMiDAS_hla_custom

Prepare MiDAS data on custom HLA level

Description

Prepare MiDAS data on custom HLA level

Usage

prepareMiDAS_hla_custom(hla_calls, hla_dictionary, ...)

Arguments

- **hla_calls**: HLA calls data frame, as returned by `readHlaCalls` function.
- **hla_dictionary**: Data frame giving HLA allele dictionary. See `hlaToVariable` for more details.
- **...**: Not used

Value

Matrix

prepareMiDAS_hla_divergence

Prepare MiDAS data on HLA divergence level

Description

Prepare MiDAS data on HLA divergence level

Usage

prepareMiDAS_hla_divergence(
    hla_calls,
    hla_divergence_aa_selection = "binding_groove",
    ...
)

Arguments

- **hla_calls**: HLA calls data frame, as returned by `readHlaCalls` function.
- **hla_divergence_aa_selection**: String specifying variable region in peptide binding groove which should be considered for Grantham distance calculation. Valid choices includes: "binding_groove", "B_pocket", "F_pocket". See details for more information.
- **...**: Not used
prepareMiDAS_hla_g_groups

Prepare MiDAS data on HLA allele’s G groups level

Description

Prepare MiDAS data on HLA allele’s G groups level

Usage

prepareMiDAS_hla_g_groups(hla_calls, ...)

Arguments

hla_calls
HLA calls data frame, as returned by readHlaCalls function.

... Not used

Value

Matrix

prepareMiDAS_hla_het

Prepare MiDAS data on HLA heterozygosity level

Description

Prepare MiDAS data on HLA heterozygosity level

Usage

prepareMiDAS_hla_het(hla_calls, hla_het_resolution = 8, ...)

Arguments

hla_calls
HLA calls data frame, as returned by readHlaCalls function.

hla_het_resolution
Number specifying HLA alleles resolution used to calculate heterogeneity.

... Not used

Value

Matrix
prepareMiDAS_hla_kir_interactions

Prepare MiDAS data on HLA - KIR interactions level

Description

Prepare MiDAS data on HLA - KIR interactions level

Usage

prepareMiDAS_hla_kir_interactions(hla_calls, kir_calls, ...)

Arguments

hla_calls  HLA calls data frame, as returned by readHlaCalls function.
kir_calls  KIR calls data frame, as returned by readKirCalls function.
...        Not used

Value

Matrix

prepareMiDAS_hla_NK_ligands

Prepare MiDAS data on HLA allele’s groups level

Description

Prepare MiDAS data on HLA allele’s groups level

Usage

prepareMiDAS_hla_NK_ligands(hla_calls, ...)

Arguments

hla_calls  HLA calls data frame, as returned by readHlaCalls function.
...        Not used

Value

Matrix
prepareMiDAS_hla_supertypes

Prepare MiDAS data on HLA allele’s supertypes level

Description
Prepare MiDAS data on HLA allele’s supertypes level

Usage
prepareMiDAS_hla_supertypes(hla_calls, ...)

Arguments
  hla_calls HLA calls data frame, as returned by \texttt{readHlaCalls} function.
  ... Not used

Value
Matrix

prepareMiDAS_kir_custom

Prepare MiDAS data on custom KIR level

Description
Prepare MiDAS data on custom KIR level

Usage
prepareMiDAS_kir_custom(kir_calls, kir_dictionary, ...)

Arguments
  kir_calls KIR calls data frame, as returned by \texttt{readKirCalls} function.
  kir_dictionary Data frame giving KIR genes dictionary. See \texttt{countsToVariables} for more details.
  ... Not used

Value
Matrix
**prepareMiDAS_kir_genes**

*Prepare MiDAS data on KIR genes level*

**Description**

Prepare MiDAS data on KIR genes level

**Usage**

```r
prepareMiDAS_kir_genes(kir_calls, ...)
```

**Arguments**

- `kir_calls` KIR calls data frame, as returned by `readKirCalls` function.
- `...` Not used

**Value**

Matrix

---

**prepareMiDAS_kir_haplotypes**

*Prepare MiDAS data on KIR haplotypes level*

**Description**

Prepare MiDAS data on KIR haplotypes level

**Usage**

```r
prepareMiDAS_kir_haplotypes(kir_calls, ...)
```

**Arguments**

- `kir_calls` KIR calls data frame, as returned by `readKirCalls` function.
- `...` Not used

**Value**

Matrix
**readHlaAlignments**  
*Read HLA allele alignments*

**Description**

`readHlaAlignments` read HLA allele alignments from file.

**Usage**

```r
readHlaAlignments(file, gene = NULL, trim = FALSE, unkchar = "")
```

**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.

**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 the 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.  

Additionally, for the alleles without sequence defined in the original alignment files we have inferred their sequence based on known higher resolution alleles.

**Value**

Matrix containing HLA allele alignments.  
Rownames correspond 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**

```r
hla_alignments <- readHlaAlignments(gene = "A")
```
**readHlaCalls**

**Read HLA allele calls**

**Description**

`readHlaCalls` read HLA allele calls from file

**Usage**

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

**Arguments**

- `file` Path to input file.
- `resolution` Number specifying desired 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 has to be a tsv formatted table with a header. First column should contain sample IDs, further columns hold HLA allele numbers. See `system.file("extdata", "MiDAS_tut_HLA.txt", package = "midasHLA")` file for an example.

`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 the following values: 2, 4, 6, 8. For more details about HLA allele numbers resolution see [http://hla.alleles.org/nomenclature/naming.html](http://hla.alleles.org/nomenclature/naming.html).

**Value**

HLA calls data frame. First column hold sample IDs, further columns hold HLA allele numbers.

**Examples**

```r
file <- system.file("extdata", "MiDAS_tut_HLA.txt", package = "midasHLA")
hla_calls <- readHlaCalls(file)
```
readKirCalls  

**Description**

readKirCalls read KIR calls from file.

**Usage**

```r
readKirCalls(file, na.strings = c("", "NA", "uninterpretable"))
```

**Arguments**

- `file` Path to input file.
- `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 has to be a tsv formatted table. First column should be named "ID" and contain samples IDs, further columns should hold KIR genes presence / absence indicators. See `system.file("extdata", "MiDAS_tut_KIR", package = "midasHLA")` for an example.

**Value**

Data frame containing KIR gene's counts. First column hold samples IDs, further columns hold KIR genes presence / absence indicators.

**Examples**

```r
file <- system.file("extdata", "MiDAS_tut_KIR.txt", package = "midasHLA")
readKirCalls(file)
```

reduceAlleleResolution  

Reduce HLA numbers resolution.

**Description**

reduceAlleleResolution reduce HLA allele numbers resolution.
Usage
   reduceAlleleResolution(allele, resolution = 4)

Arguments
   allele          Character vector with HLA allele numbers.
   resolution      Number specifying desired resolution.

Details
   In cases when allele number contain 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.
   NA values are accepted and returned as NA.
   TODO here we give such warning when alleles have G or GG suffix (see http://hla.alleles.org/alleles/g_groups.html)
   "Reducing G groups alleles, major allele gene name will be used." I don’t really remember why
   we are doing this xd These allele numbers are processed as normal alleles (without suffix). Let me
   know if this warning is relevant or we could go without it. If we want to leave it lets also add text
   in documentation.

Value
   Character vector containing reduced HLA allele numbers.

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

reduceHlaCalls
   Reduce HLA calls resolution

Description
   reduceHlaCalls reduces HLA calls data frame to specified resolution.

Usage
   reduceHlaCalls(hla_calls, resolution = 4)

Arguments
   hlaCalls    HLA calls data frame, as returned by readHlaCalls function.
   resolution  Number specifying desired resolution.
Details

Alleles with resolution greater than resolution or optional suffixes are returned unchanged.

Value

HLA calls data frame reduced to specified resolution.

Examples

reduceHlaCalls(MiDAS_tut_HLA, resolution = 2)

runMiDAS

Run MiDAS statistical analysis

Description

runMiDAS perform association analysis on MiDAS data using statistical model of choice. Function is intended for use with prepareMiDAS. See examples section.

Usage

runMiDAS(
  object,
  experiment,
  inheritance_model = NULL,
  conditional = FALSE,
  omnibus = FALSE,
  omnibus_groups_filter = NULL,
  lower_frequency_cutoff = NULL,
  upper_frequency_cutoff = NULL,
  correction = "bonferroni",
  n_correction = NULL,
  exponentiate = FALSE,
  th = 0.05,
  th_adj = TRUE,
  keep = FALSE,
  rss_th = 1e-07
)

Arguments

object An existing fit from a model function such as lm, glm and many others.
experiment String indicating the experiment associated with object’s MiDAS data to use. Valid values includes: “hla_alleles”, “hla_aa”, “hla_g_groups”, “hla_supertypes”, “hla_NK_ligands”, “kir_genes”, “kir_haplotypes”, “hla_kir_interactions”, “hla_divergence”, “hla_het”, “hla_custom”, “kir_custom”. See prepareMiDAS for more information.
runMiDAS

inheritance_model
String specifying inheritance model to use. Available choices are "dominant", "recessive", "additive".

conditional
Logical flag indicating if conditional analysis should be performed.

omnibus
Logical flag indicating if omnibus test should be used.

omnibus_groups_filter
Character vector specifying omnibus groups to use.

lower_frequency_cutoff
Number giving lower frequency threshold. Numbers greater than 1 are interpreted as the number of feature occurrences, numbers between 0 and 1 as fractions.

upper_frequency_cutoff
Number giving upper frequency threshold. Numbers greater than 1 are interpreted as the number of feature occurrences, numbers between 0 and 1 as fractions.

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. For Bonferroni correction it is possible to specify a number lower than the number of comparisons being made. This is useful in cases when knowledge about the biology or redundancy of alleles reduces the need for correction. For other methods it must be at least equal to the 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.

th
Number specifying threshold for a variable to be considered significant.

th_adj
Logical flag indicating if adjusted p-value should be used as threshold criteria, otherwise unadjusted p-value is used.

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 variable selection nonsense. This behavior can be controlled using rss_th.

Details

By default statistical analysis is performed iteratively on each variable in selected experiment. This is done by substituting placeholder in the object's formula with each variable in the experiment.

Setting conditional argument to TRUE will cause the statistical analysis to be performed in a stepwise conditional testing manner, adding the previous top-associated variable as a covariate to object's formula. The analysis stops when there is no more significant variables, based on self-defined threshold (th argument). Either adjusted or unadjusted p-values can be used as the selection criteria, which is controlled using th_adj argument.
runMiDAS

Setting omnibus argument to TRUE will cause the statistical analysis to be performed iteratively on groups of variables (like residues at particular amino acid position) using likelihood ratio test.

Argument inheritance_model specifies the inheritance model that should be applied to experiment’s data. Following choices are available:

- "dominant" carrier status is sufficient for expression of the phenotype (non-carrier: 0, heterozygous & homozygous carrier: 1).
- "recessive" two copies are required for expression of the phenotype (non-carrier & heterozygous carrier: 0, homozygous carrier: 1).
- "additive" allele dosage matters, homozygous carriers show stronger phenotype expression or higher risk than heterozygous carriers (non-carrier = 0, heterozygous carrier = 1, homozygous carrier = 2).
- "overdominant" heterozygous carriers are at higher risk compared to non-carriers or homozygous carriers (non-carrier & homozygous carrier = 0, heterozygous carrier = 1).

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

Value

Analysis results, depending on the parameters:

- conditional=FALSE, omnibus=FALSE: Tibble with first column "term" holding names of tested variables (eg. alleles). Further columns depend on the used model and are determined by associated tidy function. Generally they will include "estimate", "std.error", "statistic", "p.value", "conf.low", "conf.high", "p.adjusted".
- conditional=TRUE, omnibus=FALSE: Tibble or a list of tibbles, see keep argument. The first column "term" hold names of tested variables. Further columns depend on the used model and are determined by associated tidy function. Generally they will include "estimate", "std.error", "statistic", "p.value", "conf.low", "conf.high", "p.adjusted".
- conditional=FALSE, omnibus=TRUE: Tibble with first column holding names of tested omnibus groups (eg. amino acid positions) and second names of variables in the group (eg. residues). Further columns are: "df" giving difference in degrees of freedom between base and extended model, "statistic" giving Chisq statistic, "p.value" and "p.adjusted".
- conditional=TRUE, omnibus=TRUE: Tibble or a list of tibbles, see keep argument. The first column hold names of tested omnibus groups (eg. amino acid positions), second column hold names of variables in the group (eg. residues). Further columns are: "df" giving difference in degrees of freedom between base and extended model, "statistic" giving Chisq statistic, "p.value" and "p.adjusted".

Examples

# create MiDAS object
midas <- prepareMiDAS(hla_calls = MiDAS_tut_HLA, 
colData = MiDAS_tut_pheno, 
experiment = c("hla_alleles", "hla_aa")
)
# construct statistical model
object <- lm(disease ~ term, data = midas)

# run analysis
runMiDAS(object, experiment = "hla_alleles", inheritance_model = "dominant")

# omnibus test
# omnibus_groups_filter argument can be used to restrict omnibus test only
# to selected variables groups, here we restrict the analysis to HLA-A
# positions 29 and 43.
runMiDAS(
  object,
  experiment = "hla_aa",
  inheritance_model = "dominant",
  omnibus = TRUE,
  omnibus_groups_filter = c("A_29", "A_43")
)

runMiDASGetVarsFreq  Get variables frequencies from MiDAS

Description

Helper getting variables frequencies from MiDAS object. Additionally for binary test covariate
frequencies per phenotype are added. Used in scope of runMiDAS.

Usage

runMiDASGetVarsFreq(midas, experiment, test_covar)

Arguments

midas  MiDAS object.
experiment  String specifying experiment from midas.
test_covar  String giving name of test covariate.

Value

Data frame with variable number of columns. First column, "term" holds experiment’s variables,
further columns hold number of variable occurrence and their frequencies.
stringMatches  

*Check if string matches one of possible values*

**Description**

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

**Usage**

```r
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`.

summariseAApo

*Summarize amino acid position*

**Description**

List HLA alleles and amino acid residues at a given position.

**Usage**

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

**Arguments**

- `hla_calls`  
  HLA calls data frame, as returned by `readHlaCalls` function.

- `aa_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.
updateModel

Examples
summariseAAPosition(MiDAS_tut_HLA, "A_9")

updateModel

**Extend and Re-fit a Model Call**

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

**Usage**
updateModel(object, x, placeholder = NULL, 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.
- **placeholder** String specifying term to substitute with value from x. Ignored if set to NULL.
- **backquote** Logical indicating if added variables should be quoted. Elements of this vector are recycled over x.
- **collapse** String specifying how variables should be combined. Defaults to " + " i.e. linear combination.

**Value**
Updated fitted object.

validateFrequencyCutoffs

**Validate frequency cutoffs**

**Description**
validateFrequencyCutoffs checks if lower_frequency_cutoff and upper_frequency_cutoff are valid.

**Usage**
validateFrequencyCutoffs(lower_frequency_cutoff, upper_frequency_cutoff)
Arguments

- `lower_frequency_cutoff`
  - Number
- `upper_frequency_cutoff`
  - Number

Details

- `lower_frequency_cutoff` and `upper_frequency_cutoff` should be positive numbers, giving either frequency or counts. `lower_frequency_cutoff` has to be lower than `upper_frequency_cutoff`.

Value

- Logical indicating if `lower_frequency_cutoff` and `upper_frequency_cutoff` are valid.
Index

* datasets
  - allele_frequencies, 5
  - dict_dist_grantham, 17
  - kir_frequencies, 51
  - MiDAS_tut_HLA, 56
  - MiDAS_tut_KIR, 57
  - MiDAS_tut_object, 58
  - MiDAS_tut_pheno, 59

  aaVariationToCounts, 4
  adjustPValues, 5
  allele_frequencies, 5, 29, 32, 55
  analyzeAssociations, 6
  analyzeConditionalAssociations, 7
  applyInheritanceModel, 9
  arrange, 24
  as.data.frame.MiDAS, 10
  as.list, 52
  as.numeric, 5

  backquote, 10

  characterMatches, 11
  checkAlleleFormat, 11
  checkColDataFormat, 12
  checkHlaCallsFormat, 13
  checkKIRcallsFormat, 13
  checkKIRGenesFormat, 14
  checkStatisticalModel, 14
  colnamesMatches, 15
  convertAlleleToVariable, 15
  convertToVariables, 16, 62, 68

  dfToExperimentMat, 17
  dict_dist_grantham, 17
  distGrantham, 18

  experimentMatToDf, 18
  expression, 52

  filter, 24

  filterByFrequency, 19
  filterByFrequency, MiDAS-method
    (MiDAS-class), 54
  filterByOmnibusGroups, 20
  filterByOmnibusGroups, MiDAS-method
    (MiDAS-class), 54
  filterByVariables, 20
  filterByVariables, MiDAS-method
    (MiDAS-class), 54
  filterExperimentByFrequency, 21
  filterExperimentByVariables, 22
  filterListByElements, 23
  formatResults, 23

  getAAFrequencies, 24
  getAlleleResolution, 25
  getAllelesForAA, 26
  getAllelesForAA, MiDAS-method
    (MiDAS-class), 54
  getExperimentFrequencies, 26
  getExperimentPopulationMultiplicator, 27
  getExperiments, 28
  getExperiments, MiDAS-method
    (MiDAS-class), 54
  getFrequencies, 28
  getFrequencies, MiDAS-method
    (MiDAS-class), 54
  getFrequencyMask, 30
  getHlaCalls, 31
  getHlaCalls, MiDAS-method (MiDAS-class), 54
  getHlaCallsGenes, 31
  getHlaFrequencies, 32
  getHlaKIRInteractions, 33, 63
  getKirCalls, 34
  getKirCalls, MiDAS-method (MiDAS-class), 54
  getKIRFrequencies, 34
  getObjectDetails, 35
getOmnibusGroups, 20, 35, 55
getOmnibusGroups,MiDAS-method (MiDAS-class), 54
getPlaceholder, 36
getPlaceholder,MiDAS-method (MiDAS-class), 54
getReferenceFrequencies, 27, 36
getVariableAAPos, 37
hasTidyMethod, 38
hlaAlignmentGrantham, 38
hlaCallsGranthamDistance, 39, 63
hlaCallsToCounts, 16, 40
hlaToAAVariation, 4, 25, 40, 62
hlaToVariable, 41, 53, 62, 63, 65
HWETest, 42

isCharacterOrNULL, 44
isClass, 44
isClassOrNULL, 45
isCountOrNULL, 45
isCountsOrZeros, 46
isExperimentCountsOrZeros, 46
isExperimentInheritanceModelApplicable, 47
isFlagOrNULL, 47
isNumberOrNULL, 48
isTRUEorFALSE, 48
iterativeLRT, 49
iterativeModel, 50

kableResults, 50
kir_frequencies, 51

lapply_tryCatch, 52
listMiDASDictionaries, 53
LRTest, 53

MiDAS, 12, 19–21, 26, 28, 29, 31, 34–36, 43, 55, 61, 63
MiDAS (MiDAS-class), 54
MiDAS-class, 54
MiDAS_tut_HLA, 56
MiDAS_tut_KIR, 57
MiDAS_tut_object, 58
MiDAS_tut_pheno, 59
midasToWide, 56
MultiAssayExperiment, 54

NA, 5, 71, 72
objectHasPlaceholder, 59
omnibusTest, 60
p.adjust, 5, 7, 76
prepareMiDAS, 54, 61, 74
prepareMiDAS_hla_aa, 63
prepareMiDAS_hla_alleles, 64
prepareMiDAS_hla_custom, 65
prepareMiDAS_hla_divergence, 65
prepareMiDAS_hla_g_groups, 66
prepareMiDAS_hla_het, 66
prepareMiDAS_hla_kir_interactions, 67
prepareMiDAS_hla_NK_ligands, 67
prepareMiDAS_hla_supertypes, 68
prepareMiDAS_kir_custom, 68
prepareMiDAS_kir_genes, 69
prepareMiDAS_kir_haplotypes, 69

readHlaAlignments, 37, 70, 78
readHlaCalls, 13, 31–33, 39–41, 62, 64–68, 71, 73, 78
readKirCalls, 13, 33, 34, 62, 67–69, 72
reduceAlleleResolution, 72
reduceHlaCalls, 73
runMiDAS, 24, 35, 50, 51, 54, 62, 74
runMiDASGetVarsFreq, 77

stringMatches, 78
summariseAAPosition, 78

tidy, 7, 8, 50, 75

updateModel, 79

validateFrequencyCutoffs, 79