Package ‘Mega2R’

February 28, 2019

Version 1.0.5
Date 2019-02-27
Title Accessing and Processing a 'Mega2' Genetic Database

Description Uses as input genetic data that have been reformatted and stored in a 'SQLite' database; this database is initially created by the standalone 'mega2' C++ program (available freely from <https://watson.hgen.pitt.edu/register/>). Loads and manipulates data frames containing genotype, phenotype, and family information from the input 'SQLite' database, and decompresses needed subsets of the genotype data, on the fly, in a memory efficient manner. We have also created several more functions that illustrate how to use the data frames as well as perform useful tasks: these permit one to run the 'pedgene' package to carry out gene-based association tests on family data using selected marker subsets, to run the 'SKAT' package to carry out gene-based association tests using selected marker subsets, to run the 'famSKATRC' package to carry out gene-based association tests on families (optionally) and with rare or common variants using selected marker subsets, to output the 'Mega2R' data as a VCF file and related files (for phenotype and family data), and to convert the data frames into CoreArray Genomic Data Structure (GDS) format.

URL https://watson.hgen.pitt.edu/mega2/mega2r/

BugReports https://groups.google.com/forum/#!forum/mega2-users

Depends R (>= 3.5.0), SKAT, pedgene, gdsfmt
License GPL-2

LinkingTo Rcpp
Imports AnnotationDbi, DBI, GenomeInfoDb, RSQLite, methods, famSKATRC, kinship2

Suggests knitr, rmarkdown, formatR, TxDb.Hsapiens.UCSC.hg19.knownGene, org.Hs.eg.db

NeedsCompilation yes
RoxygenNote 6.1.0
LazyData true
VignetteBuilder knitr
Author Robert V. Baron [aut],
    Daniel E. Weeks [aut, cre],
    University of Pittsburgh [cph]
Maintainer Daniel E. Weeks <weeks@pitt.edu>
Repository CRAN
Date/Publication 2019-02-28 05:50:12 UTC

R topics documented:

applyFnToGenes .............................................. 3
applyFnToMarkers ........................................... 5
applyFnToRanges ............................................ 6
clean_mega2rtutorial_data ................................. 8
computeDosage ................................................ 9
dbmega2_import .............................................. 10
DOfamSKATRC .................................................. 10
DOpedgene .................................................... 12
DOSKAT ......................................................... 13
dump_mega2rtutorial_data ................................. 15
getgenotypes .................................................. 15
getgenotypessedos ........................................... 16
getgenotypessgenabel ....................................... 17
getgenotypessraw ............................................. 19
init_famSKATRC ............................................... 20
init_pedgene .................................................. 21
init_SKAT ...................................................... 22
Mega2ENVGenABEL ............................................ 23
Mega2famSKATRC ............................................. 24
Mega2gdsfmt .................................................. 25
Mega2GenABEL ................................................ 26
Mega2GenABELtst ............................................. 27
Mega2pedgene ................................................ 28
Mega2R ......................................................... 29
Mega2R-TBLS .................................................. 29
Mega2R-TBLSFilter .......................................... 29
Mega2RVersion ................................................ 30
Mega2SKAT ...................................................... 30
Mega2VCF ....................................................... 32
mkfam ......................................................... 33
mkMarkers ..................................................... 34
mkphenotype .................................................. 35
read.Mega2DB ................................................ 36
setAnnotations ................................................. 36
applyFnToGenes

Description

This function generates base pair ranges from its input arguments. Each range specifies a chromosome, a start base pair and end base pair. Typically, a range could be a gene transcript, though it could be a whole chromosome, or a run of base pairs on a chromosome. Once the ranges are generated, applyFnToRanges is called to find all the rows (i.e. markers) from the markers data frame that fall in each range. For these markers, a matrix of the genotypes is generated. Finally, the op function is called for each range with the arguments: markers, range, and 'environment'.

Usage

applyFnToGenes(op = function (markers, range, envir) {},
genesis_args = NULL,
ranges_args = matrix(ncol = 3, nrow = 0),
chrs_args = vector("integer", 0),
markers_args = vector("character", 0),
type_args = "TX",
fuzz_args = 0,
envir = ENV)

Arguments

op Is a function of three arguments. It will be called repeatedly by applyFnToGenes in a try/catch context. The arguments are:

markers Marker data for each marker selected. A marker is a data frame with the following 5 observations:

locus_link is the ordinal ranking of this marker among all loci
locus_link_fill is the position of corresponding marker genotype data in the unified_genotype_table
MarkerName is the text name of the marker
chromosome is the integer chromosome number
position is the integer base pair position of marker

range An indicator of which range argument these markers correspond to.
applyFnToGenes

**envir** An 'environment' holding Mega2R data frames and state data.

**genes_arg**
a character vector of gene names. All the transcripts identified with the specified gene in BioConductor Annotation, `TxDb.Hsapiens.UCSC.hg19.knownGene`, are selected. This produces multiple "range" elements containing chromosome, start base pair, and end base pair. (If the gene name is "*", all the transcript will be selected.) Note: BioConductor Annotation `org.Hs.eg.db` is used to convert from gene name to ENTREZ gene id.

**ranges_arg**
an integer matrix of three columns. The columns define a range: a chromosome number, a start base pair value, and an end base pair value.

**chrs_arg**
an integer vector of chromosome numbers. All of the base pairs on each chromosomes will be selected as a single range.

**markers_arg**
a data frame with the following 5 observations:

- **locus_link** is the ordinal ranking of this marker among all loci
- **locus_link_fill** is the position of corresponding marker genotype data in the `unified_genotype_table`
- **MarkerName** is the text name of the marker
- **chromosome** is the integer chromosome number
- **position** is the integer base pair position of marker

**type_arg**
a character vector of length 1 that contains "TX" or does not. If it is "TX", which is the default, the `TX` fields of BioConductor Annotation, `TxDb.Hsapiens.UCSC.hg19.knownGene` are used to define the base pair ranges and chromosome. Otherwise, the `CDS` fields are used.

**fuzz_arg**
is an integer vector of length one or two. The first argument is used to reduce the start base pair selected from each transcript and the second to increase the end base pair position. (If only one value is present, it is used for both adjustments.) Note: The values can be positive or negative.

**envir** an 'environment' that contains all the data frames created from the SQLite database.

**Value**
None

**Note**
If you want subsequent calls to `op` to share information, data can be placed in a data frame that is added to the 'environment'.

**Examples**

```r
db = system.file("exdata", "seqsimm.db", package="Mega2R")
ENV = read.Mega2DB(db)

show = function(m, r, e) {
  print(r)
```

applyFntoMarkers

apply a function to the genotypes from a set of markers

Description

A matrix of the genotypes for all the specified markers is generated. Then, the call back function, op, is called with the markers, NULL (for the range), and the 'environment'.

Usage

applyFntoMarkers(op = function (markers, range, envir) {},
markers_arg,
envir = ENV)

Arguments

op Is a function of three arguments. It will be called once by applyFntoMarkers in a try/catch context. The arguments are:

markers Marker data for each marker in geno. A marker is a data frame with the following 5 observations:

locus_link is the ordinal ranking of this marker among all loci
locus_link_fill is the position of corresponding marker genotype data in the unified_genotype_table
MarkerName is the text name of the marker
chromosome is the integer chromosome number
position is the integer base pair position of marker
range NULL: to indicate no explicit range was specified.
envir An 'environment' holding Mega2R data frames and state data.
markers_arg a data frame with the following 5 observations:
locus_link is the ordinal ranking of this marker among all loci
locus_link_fill is the position of corresponding marker genotype data in the
unified_genotype_table
MarkerName is the text name of the marker
chromosome is the integer chromosome number
position is the integer base pair position of marker
envir an 'environment' that contains all the data frames created from the SQLite
database.

Value
None

Examples
db = system.file("exdata", "seqsim.db", package="Mega2R")
ENV = read.Mega2DB(db)
show = function(m, r, e) {
  print(r)
  print(m)
  print(head(getgenotypes(m, envir = e)))
}

# apply function "show" to all genotypes > 5,000,000 bp
applyFnToMarkers(show, ENV$markers[ENV$markers$position > 5000000,])

Description
First, for each range, determine the markers that fall between the start and end base pair of the
range. Then, for each set of markers generate a matrix of the genotypes of those markers. Finally,
the op function is called for each range with the arguments: markers, range, and 'environment'.

Usage
applyFnToRanges = function (markers, range, envir) {}
  ranges_arg = NULL,
  indices_arg = NULL,
  fuzz_arg = 0,
  envir = ENV)
Arguments

op

Is a function of three arguments. It will be called repeatedly by \texttt{applyFnToRanges} in a try/catch context. The arguments are:

marker

Marker data for each marker in \texttt{geno}. A marker is a data frame with the following 5 observations:

- \texttt{locus\_link} is the ordinal ranking of this marker among all loci
- \texttt{locus\_link\_fill} is the position of corresponding marker genotype data in the \texttt{unified\_genotype\_table}
- MarkerName is the text name of the marker
- \texttt{chromosome} is the integer chromosome number
- \texttt{position} is the integer base pair position of marker

range

An indicator of which range argument of \texttt{applyFnToRanges} these markers correspond to.

envir

An 'environment' holding Mega2R data frames and state data.

ranges_arg

is a data frame that contains at least 4 observations: a name, a chromosome, a start base pair position and an end base pair position.

indices_arg

is a vector of 3 integers that specify the location of chromosome, start base pair column and end base pair column of the ranges_arg data frame. An optional fourth integer indicates the column containing the name of the ranges.

fuzz_arg

is an integer vector of length one or two. The first argument is used to reduce the start base pair selected from each range and the second to increase the end base pair position. (If only one value is present, it is used for both changes.) Note: The values can be positive or negative.

envir

an 'environment' that contains all the data frames created from the SQLite database.

Value

None

Note

If the \texttt{ranges\_arg} and \texttt{indices\_arg} are NULL or missing, then the default ranges that have been set by \texttt{setRanges} are used. If \texttt{setRanges} has not been called, a default set of the ranges is used.

Examples

```r
db = system.file("exdata", "seqsim.db", package="Mega2R")
ENV = read.Mega2DB(db)

show = function(m, r, e) {
  print(r)
  print(m)
  print(head(getgenotypesraw(m, envir = e)))
}

# apply function "show" to all genotypes on chromosomes 1 for two base pair
```

# ranges
applyFnToRanges(show,
    ranges_arg =
    matrix(c(1, 2244000, 2245000,
            1, 3762500, 3765000),
           ncol = 3, nrow = 2, byrow = TRUE),
    indices_arg = 1:3)

# apply function "show" to all genotypes on chromosomes 1 for two base pair
# ranges
applyFnToRanges(show,
    ranges_arg =
    matrix(c(1, 2240000, 2245000, "range1",
            1, 3760000, 3765000, "range2"),
           ncol = 4, nrow = 2, byrow = TRUE),
    indices_arg = 1:4)

---

clean_mega2rtutorial_data

remove tutorial data

---

Description

This function removes the Mega2R tutorial (inst/exdata) data that was copied to the specified directory.

Usage

clean_mega2rtutorial_data(dir = file.path(tempdir(), "Mega2Rtutorial"))

Arguments

dir The directory to remove the tutorial data to. By default, this is tempdir()/Mega2Rtutorial

Value

None

Examples

clean_mega2rtutorial_data()
computeDosage

**computeDosage function**

**Description**
Convert the genotypesraw() allele patterns of 0x10001, 0x10002 (or 0x20001), 0x20002, 0 to the numbers 0, 1, 2, 9 for each marker. (Reverse, the order iff allele "1" has the minor allele frequency.)

**Usage**
```
computeDosage(markers_arg, range_arg, envir)
```

**Arguments**

- `markers_arg` a data.frame with the following 5 observations:
  - `locus_link` is the ordinal ranking of this marker among all loci
  - `locus_link_fill` is the position of corresponding genotype data in the `unified_genotype_table`
  - `MarkerName` is the text name of the marker
  - `chromosome` is the integer chromosome number
  - `position` is the integer base pair position of marker

- `range_arg` one row of a `ranges_arg`. The latter is a data frame of at least three integer columns. The columns indicate a range: a chromosome number, a start base pair value, and an end base pair value.

- `envir` 'environment' containing SQLite database and other globals especially the phenotype_table, phe.

**Value**
a matrix of samples X markers for all the markers that have nonzero changes.

**See Also**
- `DOfamSKATRC`

**Examples**
```
db = system.file("exdata", "seqsimm.db", package="Mega2R")
ENV = init_famSKATRC(db, verbose = TRUE)
dimDosage = function(m, r, e) {print(dim(computeDosage(m, r, e)))
applyFnToRanges(dimDosage, ENV$refRanges[50:60, ], ENV$refIndices, envir=ENV)
# This will use return dosage matrices for the markers in the ranges 50 - 60, # but is basically ignores the results.
dbmega2_import  
**read Mega2 SQLite database into R**

**Description**

Read the fields of SQLite database tables that are required for Mega2R into data frames. These data frames are stored in an 'environment' which is returned. This function also adds some state data, extra data frames, and computed data frames to the 'environment'.

**Usage**

```R
dbmega2_import(dbname,
    bpPosMap = NULL,
    verbose = FALSE)
```

**Arguments**

- **dbname**: file path to SQLite database.
- **bpPosMap**: index that specifies which map in the map_table should be used for marker chromosome/position. If it is NULL, the internal variable `base_pair_position_index` is used instead. `showMapNames()` shows the association between map name and map number.
- **verbose**: print out statistics on the name/size of each table read and show column headers. Also, save the verbose value for use by other Mega2R functions.

**Value**

`envir` an environment that contains all the data frames made from the SQLite database.

**Examples**

```R
db = system.file("exdata", "seqsim.db", package="Mega2R")
ENV = dbmega2_import(db, verbose = TRUE)
ENV = dbmega2_import(db)
```

---

**DOfamSKATRC**  
**DofamSKATRC call back function**

**Description**

Convert the genotypesraw() allele patterns of 0x10001, 0x10002 (or 0x20001), 0x20002, 0 to the numbers 0, 1, 2, 9 for each marker. (Reverse, the order iff allele "1" has the minor allele frequency.) Ignore markers that have no variants. Finally, invoke `famskat_RC` with the converted genotype matrix. Save information about the range and the p.value calculated by `famskat_RC` in `envir$famSKATRC_results`. If you want to change the argument values to this function they should be changed instead when calling the Mega2FamSKATRC function.
Usage

DOfamSKATRC(markers_arg, range_arg, envir, pheno = 3, id = NULL, covariates = NULL, sqrtweights_c = NULL, sqrtweights_r = NULL, binomialimpute = TRUE, acc = 1e-06, maf = 0.05, phi = c(0, 0.2, 0.5, 0.9))

Arguments

markers_arg a data.frame with the following 5 observations:

- **locus_link** is the ordinal ranking of this marker among all loci
- **locus_link_fill** is the position of corresponding genotype data in the `unified_genotype_table`
- **MarkerName** is the text name of the marker
- **chromosome** is the integer chromosome number
- **position** is the integer base pair position of marker

range_arg one row of a ranges_arg. The latter is a data frame of at least three integer columns. The columns indicate a range: a chromosome number, a start base pair value, and an end base pair value.

envir 'environment' containing SQLite database and other globals especially the phenotypes_table, phe.

pheno is an index into the phenotypes_table to select the phenotype. Missing phenotypes are represented by NA.

id a vector of individuals to be included in the test, a subset of the family members. If NULL is given, all members will be used.

covariates a matrix of covariated for the phenotype.

sqrtweights_c weight function for common variants, if NULL use weight set in init_famSKAT

sqrtweights_r weight function for rare variants, if NULL use weight set in init_famSKAT.

binomialimpute if TRUE, impute missing genotypes using a binomial distribution.

acc accuracy used in Davies approximation.

maf threshold used to separate rare from common variants.

phi a vector of ratios ratios; each indicates the contribution of rare variants.

Value

None

Note

This function accumulates output in the data frame, `envir$famSKATRC_results`. It will print out the lines as they are generated if `envir$verbose` is TRUE. It does not write the data frame to a file. You must save the data frame. You also must initialize the data frame when necessary.

See Also

- `init_famSKATRC`
- `Mega2famSKATRC`
Examples

```r
db = system.file("exdata", "seqsim.db", package="Mega2R")
ENV = init_famsSKATRC(db, verbose = TRUE)
ENV$famsSKATRC_results = ENV$famsSKATRC_results[0, ]
Mega2famSKATRC(gs=1:1, envir=ENV, pheno=3)
# this sets one of the many arguments for DOfamSKATRC
# but basically prepares the ENV for the direct use of DOfamSKATRC (below).

# don't test check: try this below instead if there is time
Mega2famSKATRC(genes=c("CEP104"), envir=ENV, pheno=3 )

# DOfamsSKATRC is called within Mega2famSKATRC. init_famsSKATRC and Mega2famSKATRC need to be
# called to set up the environment for famSKAT_RC to run. BUT, you should ignore DOfamSKATRC
# and use Mega2famSKATRC instead.

applyFnToRanges(DOfamSKATRC, ENV$refRanges[50:60, ], ENV$refIndices, envir=ENV)
# this will use all the default argument values for DOfamSKATRC
```

---

**DOpedgene**

`pedgene call back function`

**Description**

First, ignore call backs that have less than two markers. Second, convert the genotypes' raw() patterns of 0x10001, 0x10002 or 0x20001, 0x20002, 0 from the genotype matrix to the numbers 0, 1, 2, 0 for each marker. (Reverse, the order if allele "1" has the minor allele frequency.) Next, prepend the pedigree and person columns of the family data to this modified genotype matrix. Finally, invoke pedgene with the family data and genotype matrix for several different weights. Save the kernel and burden, value and p-value for each measurement in `envir$pedgene_results`.

**Usage**

```r
DOpedgene(markers_arg, range_arg, envir = ENV)
```

**Arguments**

- `markers_arg`: a data.frame with the following 5 observations:
  - `locus_link`: is the ordinal ranking of this marker among all loci
  - `locus_link_fill`: is the position of corresponding genotype data in the `unified_genotype_table`
  - `MarkerName`: is the text name of the marker
  - `chromosome`: is the integer chromosome number
  - `position`: is the integer base pair position of marker

- `range_arg`: one row of a `ranges_arg`. The latter is a data frame of at least three integer columns. The columns indicate a range: a chromosome number, a start base pair value, and an end base pair value.
envir 'environment’ containing SQLite database and other globals

Value

None

Note

This function appends output to the data frame, envir$pedgene_results. It will print out the lines as they are generated if envir$verbose is TRUE. The data frame envir$pedgene_results is initialized by init_pedgene, and is appended to each time DOpedgene is run.

See Also

init_pedgene

Examples

```r
db = system.file("exdata", "seqsimm.db", package="MegaR")
ENV = init_pedgene(db)
ENV$verbose = TRUE
applyFnToRanges(DOpedgene, ENV$refRanges[50:60,], ENV$refIndices)

# donttestcheck: try this below if there is time
applyFnToGenes(DOpedgene, genes_arg = c("CEP104"))
```

DOSKAT

---

**Description**

Convert the genotypesraw() allele patterns of 0x10001, 0x10002 (or 0x20001), 0x20002, 0 to the numbers 0, 1, 2, 9 for each marker. (Reverse, the order if allele "1" has the minor allele frequency.) Ignore markers that have no variants (unless allMarkers is TRUE). Finally, invoke SKAT with the converted genotype matrix, Null model saved in envir$obj, and any additionally supplied arguments. Save information about the range and the p.value calculated by SKAT in envir$SKAT_results.

**Usage**

DOSKAT(markers_arg, range_arg, envir, ...)

---

SKAT call back function
**Arguments**

- `markers_arg` a data.frame with the following 5 observations:
  - `locus_link` is the ordinal ranking of this marker among all loci
  - `locus_link_fill` is the position of corresponding genotype data in the `unified_genotype_table`
  - `MarkerName` is the text name of the marker
  - `chromosome` is the integer chromosome number
  - `position` is the integer base pair position of marker

- `range_arg` one row of a `ranges_arg`. The latter is a data frame of at least three integer columns. The columns indicate a range: a chromosome number, a start base pair value, and an end base pair value.

- `envir` 'environment' containing SQLite database and other globals

- `...` extra arguments for SKAT

**Value**

None

**Note**

This function accumulates output in the data frame, `envir$SKAT_results`. It will print out the lines as they are generated if `envir$verbose` is TRUE. It does not write the data frame to a file. You must save the data frame. You also must initialize the data frame when necessary.

**See Also**

`init_SKAT`, `Mega2SKAT`

**Examples**

```r
db = system.file("exdata", "seqsim.db", package="Mega2R")
ENV = init_SKAT(db, verbose = TRUE, allMarkers = FALSE)
Mega2SKAT(ENV$phex[, 3] - 1 ~ 1, "D", gs=1:1)

# donttestcheck: try this below instead if there is time
Mega2SKAT(ENV$phex[, 3] - 1 ~ 1, "D", kernel = "linear.weighted",
weights.beta = c(0.5, 0.5), genes=c("CEP104") )

# DOSKAT is called internally to Mega2SKAT. init_SKAT and Mega2SKAT need to be
# called to set up the environment for DOSKAT to run. You should ignore DOSKAT
# and use Mega2SKAT instead
#
applyFnToRanges(DOSKAT, ENV$refRanges[50:60, ], ENV$refIndices)
```
dump_mega2rtutorial_data

dump tutorial data

Description

This function retrieves data stored in the Mega2rtutorial (inst/exdata). It dumps them in the specified directory.

Usage

dump_mega2rtutorial_data(dir = file.path(tempdir(), "Mega2Rtutorial"))

Arguments

dir The directory to store the tutorial data to. By default, this is tempdir()/Mega2Rtutorial

Value

None

Examples

dump_mega2rtutorial_data()

getgenotypes
etch genotype character matrix for specified markers

Description

This function calls a C++ function that does all the heavy lifting. It passes the arguments necessary for the C++ function: some from the caller’s arguments and some from data frames that are in the "global" environment, envir. From the markers_arg argument, it fetches the locus_index and the index in the unified_genotype_table. It also passes the allele nucleotide separator argument. From the "global" environment, envir, it gets a bit vector of compressed genotype information, the alleles for each marker, and some bookkeeping related data. Note: This function also contains a dispatch/switch on the type of compression in the genotype vector. A different C++ function is called when there is compression versus when there is no compression.

Usage

getgenotypes(markers_arg, sepstr = ",", envir = ENV)
Arguments

- **markers_arg**: a data.frame with the following 5 observations:
  - **locus_link**: the ordinal ranking of this marker among all loci
  - **locus_link_fill**: is the position of corresponding genotype data in the *unified_genotype_table*
  - **MarkerName**: is the text name of the marker
  - **chromosome**: is the integer chromosome number
  - **position**: is the integer base pair position of marker
- **sepstr**: separator string inserted between the alleles (default is none). When present, this is typically a space, a tab or "/".
- **envir**: an environment that contains all the data frames created from the SQLite database.

Details

The *unified_genotype_table* contains one raw vector for each person. In the vector there are two bits for each genotype. This function creates an output matrix by fixing the marker and collecting genotype information for each person and then repeating for all the needed markers. (Currently, this appears slightly faster than a scan which fixes the person and iterates over markers.)

Value

A matrix of genotypes represented as two allele pairs. The matrix has one column for each marker in **markers_arg** argument. There is one row for each person in the family (*fam*) table.

Examples

```r
db = system.file("exdata", "seqsim.db", package="Mega2R")
ENV = read.Mega2DB(db)
getgenotypes(ENVmarkers)
```

Description

This function calls a C++ function that does all the heavy lifting. It passes the arguments necessary for the C++ function: some from the caller's arguments and some from data frames that are in the "global" environment, **envir**. From its **markers_arg** argument, it gets the locus_index and the index in the *unified_genotype_table*. From the "global" environment, **envir**, it gets a bit vector of compressed genotype information, and some bookkeeping related data. Note: This function also contains a dispatch/switch on the type of compression in the genotype vector. A different C++ function is called when there is compression versus when there is no compression.

Usage

```r
getgenotypesdos(markers_arg, envir = ENV)
```
getgenotypesgenabel

Arguments

`markers_arg`  a data.frame with the following 5 observations:

- `locus_link`  is the ordinal ranking of this marker among all loci
- `locus_link_fill`  is the position of corresponding genotype data in the `unified_genotype_table`
- `MarkerName`  is the text name of the marker
- `chromosome`  is the integer chromosome number
- `position`  is the integer base pair position of marker

`envir`  an environment that contains all the data frames created from the SQLite database.

Details

The `unified_genotype_table` contains one raw vector for each person. In the vector, there are two bits for each genotype. This function creates an output matrix by fixing the marker and collecting genotype information for each person and then repeating for all the specified markers.

Value

A list of 3 values, named "ncol", "zero", "geno".

- `geno`  is a matrix of dosages as integers. The value 0 is given to the Major allele value, 1 is given to the heterozygote value, and 2 is given to the Minor allele. In the matrix, there is usually one column for each marker in the `markers_arg` argument. But if there would be only the one allele 0 or 2 in the column, the column is ignored (not present). There is one row for each person in the family (`fam`) table.

- `ncol`  Is the count of the actual number of columns in the geno matrix.

- `zero`  Is a vector with one entry per marker. The value will be 0 if the marker is not in the geno matrix. Otherwise, the value is the column number in the geno matrix where the marker data appears.

Examples

```r
db = system.file("exdata", "seqsim.db", package="Mega2R")
ENV = read.Mega2DB(db)

getgenotypesdos(ENV$markers[ENV$markers$chromosome == 1,])
```

---

**getgenotypesgenabel**  process the genotype matrix for specified markers and return the corresponding GenABEL genotype matrix
Description

This function calls a C++ function that does all the heavy lifting. It passes the arguments necessary for the C++ function: some from the caller’s arguments and some from data frames that are in the "global" environment, `envir`. From its markers_arg argument, it gets the locus_index and the index in the `unified_genotype_table`. From the "global" environment, `envir`, it gets a bit vector of compressed genotype information, allele information, and some bookkeeping related data. Note: This function also contains a dispatch/switch on the type of compression in the genotype vector. A different C++ function is called when there is compression versus when there is no compression.

Usage

```r
getgenotypesgenabel(markers_arg, envir = ENV)
```

Arguments

- `markers_arg` a data.frame with the following 5 observations:
  - `locus_link` is the ordinal ranking of this marker among all loci
  - `locus_link_fill` is the position of corresponding genotype data in the `unified_genotype_table`
  - `MarkerName` is the text name of the marker
  - `chromosome` is the integer chromosome number
  - `position` is the integer base pair position of marker

- `envir` an environment that contains all the data frames created from the SQLite database.

Details

This function reads the genotype data in Mega2 compressed format and converts it to the GenABEL compressed format. The `unified_genotype_table` contains one raw vector for each person. In the vector, there are two bits for each genotype; each byte has the data for 4 markers. In GenABEL, there is one raw vector per marker, and each byte has the data for 4 persons. The C++ function does the conversion as well as adjusts the bits’ contents. For example, in GenABEL the genotype represented by bits == 0, is what Mega2 represents with 2. Doing the conversion in C++ is 10 - 20 times faster than converting the Mega2 data to PLINK .tped files and then having GenABEL read in and process/convert those files.

Value

the GenABEL gwaa.data-class object component that contains the genotype data.

Note

This function is called from `Mega2ENVGenABEL`; it is not intended to be called by the programmer.

Examples

```r
db = system.file("exdata", "seqsim.db", package="Mega2R")
ENV = read.Mega2DB(db)

aa = getgenotypesgenabel(ENV$markers[ENV$markers$chromosome == 1,])
```
Description

This function calls a C++ function that does all the heavy lifting. It passes the arguments necessary for the C++ function: some from the caller’s arguments and some from data frames that are in the "global" environment, envir. From its markers_arg argument, it gets the locus_index and the index in the unified_genotype_table. From the "global" environment, envir, it gets a bit vector of compressed genotype information, and some bookkeeping related data. Note: This function also contains a dispatch/switch on the type of compression in the genotype vector. A different C++ function is called when there is compression versus when there is no compression.

Usage

getgenotypesraw(markers_arg, envir = ENV)

Arguments

- markers_arg: a data.frame with the following 5 observations:
  - locus_link: is the ordinal ranking of this marker among all loci
  - locus_link_fill: is the position of corresponding genotype data in the unified_genotype_table
  - MarkerName: is the text name of the marker
  - chromosome: is the integer chromosome number
  - position: is the integer base pair position of marker

- envir: an environment that contains all the data frames created from the SQLite database.

Details

The unified_genotype_table contains one raw vector for each person. In the vector, there are two bits for each genotype. This function creates an output matrix by fixing the marker and collecting genotype information for each person and then repeating for all the needed markers.

Value

A matrix of genotypes represented as integers. Each 32 bit integer represents contains two allele values: the high 16 bits contains the index of allele1 and the low 16 bits contains the index of allele2. In the matrix, there is one column for each marker in the markers_arg argument. There is one row for each person in the family (fam) table.
Examples

db = system.file("exdata", "seqsimm.db", package="Mega2R")
ENV = read.Mega2DB(db)

# two ints in upper/lower half integer representing allele # for all persons in chromosome 1
getgenotypesraw(ENV$markers[ENV$markers$chromosome == 1,])

init_famSKATRC

Description

This populates the R data frames with the specified Mega2 SQLite database. It initializes the fam(ily) table and makes sure the person entries are unique. Finally, it generates a kinship matrix from the family data. It also stores a weighting for the common and rare variant that may be used later if NULL is specified as a weight in Mega2famSKATRC. The common weighting is the function dbeta(maf, 1, 25). The rare weighting is the function dbeta(maf, 0.5, 0.5).

Usage

init_famSKATRC(db = NULL, verbose = FALSE, ALPHA = FALSE, ...)

Arguments

db specifies the path of a Mega2 SQLite database containing study data.
verbose TRUE indicates that diagnostic printouts should be enabled. This value is saved in the returned environment.
ALPHA TRUE indicates that two runs of famSKAT_RC should be enabled. One with ALPHA numeric ID's and one with numeric IDs ... this is temporary. The default is FALSE.
...

Value

"environment" containing data frames from an SQLite database and some computed values.

Note

init_famSKATRC creates a new data frame, envir$phe, containing phenotype observations. In addition, it initializes a matrix to aid in translating a genotype allele matrix to a genotype count matrix. It also initializes the data frame envir$famSKATRC_results to zero rows.
init_pedgene

See Also

Mega2famSKATRC

Examples

db = system.file("exdata", "seqsimm.db", package="Mega2R")
ENV = init_famSKATRC(db, verbose = FALSE)
ls(ENV)

init_pedgene  | load Mega2 SQLite database and perform initialization for pedgene usage

Description

This populates the R data frames from the specified Mega2 SQLite database.

Usage

init_pedgene(db = NULL, verbose = FALSE, traitname = "default", ...)

Arguments

db               | specifies the path of a Mega2 SQLite database containing study data.
verbose          | TRUE indicates that diagnostic printouts should be enabled. This value is saved in the returned environment.

traitname        | Name of the affection status trait to use to set the case/control status; default value = "default".

...               | fed to dbmega2_import(); should be bpPosMap= to select from the maps of base pairs, if the default is not desired.

Value

"environment" containing data frames from an SQLite database and some computed values.

Note

init_pedgene calculates schaidPed and pedPer that are used later in the Dopedgene calculation. In addition, it initializes a matrix to aid in translating a genotype allele matrix to a genotype count matrix.

It also initializes the dataframe envir$pedgene_results to zero rows.

See Also

D0pedgene, Mega2pedgene, mkfam
init_SKAT

Description
This populates the R data frames from the specified Mega2 SQLite database. It then prunes the samples to only include members that have a definite case or control status. Undefined samples are ignored.

Usage
```
init_SKAT(db = NULL, verbose = FALSE, allMarkers = FALSE, ...)  
```

Arguments
- `db`: specifies the path of a Mega2 SQLite database containing study data.
- `verbose`: TRUE indicates that diagnostic printouts should be enabled. This value is saved in the returned environment.
- `allMarkers`: TRUE means use all markers in a given transcript even if there is no variation. FALSE means ignore markers that show no variation; this is the default.
- `...`: fed to dbmega2_import(); should be bpPosMap= to select from the maps of base pairs, if the default is not desired.

Value
"environment" containing data frames from an SQLite database and some computed values.

Note
init_SKAT creates a data frame, envir$phe, of phenotype observations. In addition, it initializes a matrix to aid in translating a genotype allele matrix to a genotype count matrix. It also initializes the data frame envir$SKAT_results to zero rows.

See Also
Mega2SKAT

Examples
```
db = system.file("exdata", "seqsimm.db", package="Mega2R")
ENV = init_pedgene(db, traitname = "default")
ls(ENV)
```
**Examples**

```r
db = system.file("exdata", "seqsim.db", package="Mega2R")
ENV = init_SKAT(db, verbose = FALSE, allMarkers = FALSE)
ls(ENV)
```

**Description**

create a gwaa.data-class object from the data frames in a Mega2 environment. This function is a front end that eventually calls a C++ Rcpp function that reads the genotype data in Mega2 compressed format and converts it to the GenABEL compressed format. The results of Mega2ENVGenABEL are/should be the same as Mega2GenABEL, but the calculation is much faster, typically a factor of 10 to 20.

**Usage**

```r
Mega2ENVGenABEL(markers = NULL, force = TRUE, makemap = FALSE, sort = TRUE, envir = ENV)
```

**Arguments**

- `markers`: data frame of markers to be processed
- `force`: pass value to gwaa conversion function
- `makemap`: pass value to gwaa conversion function
- `sort`: pass value to gwaa conversion function
- `envir`: 'environment' containing SQLite database and other globals

**Value**

gwaa.data-class object created from Mega2R database

**Examples**

```r
## Not run:
require("GenABEL")
db = system.file("exdata", "seqsim.db", package="Mega2R")
ENV = read.Mega2DB(db)
gwaa = Mega2ENVGenABEL(markers=ENV$markers[1:10,])

str(gwaa)
head(summary(gwaa))

## End(Not run)
```
Mega2famSKATRC  

**execute the CRAN famSKAT_RC function on a subset of the gene transcripts**

---

**Description**

If the `gene` argument is NULL, execute the `famSKAT_RC` function on the first `gs` gene transcripts (default is gs = 1:100). Update the `envir$famSKATRC_results` data frame with the results. Otherwise, `gene` is a string vector of genes to process. The special value `'*'` stands for all the known genes.

**Usage**

```r
Mega2famSKATRC(gs = 1:100, genes = NULL, envir = ENV, ...)
```

**Arguments**

- `gs` a subrange of the default transcripts (`refRanges`) over which to calculate the `DOfamSKATRC` function.
- `genes` a list of genes over which to calculate the `DOfamSKATRC` function. The value, `'*'`, means use all the transcripts in the selected Bioconductor database. If genes is NULL, the gs range of the internal `refRanges` will be used.
- `envir` 'environment' containing SQLite database and other globals.
- `...` extra arguments that are acceptable to `famSKAT_RC`. These are listed with the `DOfamSKATRC` function.

**Value**

The data frame with the results is stored in the environment and named `famSKATRC_results`, viz. `envir$famSKATRC_results`.

**Note**

A helper function `SKAT3arg` is defined for the 3 argument callback function which in turn calls `DOfamSKATRC` with the appropriate arguments (including those specific to the `Mega2famSKATRC` function).

**See Also**

- `init_famSKATRC`, `DOfamSKATRC`
Examples

```r
db = system.file("exdata", "seqsimm.db", package="Mega2R")
ENV = init_famSKATRC(db, verbose = FALSE)
ENV$verbose = FALSE
ENV$famSKATRC_results = ENV$famSKATRC_results[0, ]
Mega2famSKATRC(gs=50:60, envir=ENV, pheno=3)

# donttestcheck: try this below if there is time
Mega2famSKATRC(genes=c("CEP104"), envir=ENV, pheno=3)

ENV$famSKATRC_results
```

Mega2gdsfmt  *transcode mega2 to gdsfmt/SNP_ARRAY*

Description

Reads the data frames in "envir" and builds a GDSFMT COREARRAY file from them.

Usage

```r
Mega2gdsfmt(filename = "test.gds", markers = NULL, snp.order = FALSE,
          SeqArray = FALSE, envir = ENV)
```

Arguments

- **filename**: gdsfmt file to create
- **markers**: data frame of markers to be processed
- **snp.order**: TRUE indicates that the "genotype" data matrix has SNP as the first index which changes more quickly than subsequent indices. FALSE indicates that SAMPLE is the the first index.
- **SeqArray**: TRUE uses SeqArray labels for the gdsfmt vector elements. FALSE it uses labels shown in SNPRelate
- **envir**: 'environment' containing SQLite database and other globals

Value

writes the "filename" file containing the CoreArray data. Then returns an internal pointer, class .gds, to the file data.

See Also

gdsfmt
Examples

db = system.file("exdata", "seqsimm.db", package="Mega2R")
ENV = read.Mega2DB(db)
gdsfmtfile = file.path(where_mega2rtutorial_data(), "test.gds")
append_genotype_a = TRUE
append_genotype_b = append_genotype_c = FALSE
gn = Mega2gdgsfmt(gdsfmtfile, envir=ENV)
gn

---

**Mega2GenABEL**

*generate gwaa.data-class object from a Mega2R database*

Description

Call the *Mega2R* functions to: create a .tped file, a .tfam file and a .phe file. Then call the GenABEL functions to process these files: the .tped and the .tfam file are processed by `convert.snp.tped` to produce a tped.raw file. The latter is combined with a .phe (phenotype) file by `load.gwaa.data` to create a gwaa.data-class object in memory. All these files are deleted when the exits.

Usage

```r
Mega2GenABEL(markers = NULL, mapno = 0, envir = ENV)
```

Arguments

- **markers**: data frame of markers to be processed
- **mapno**: specify which map index to use for physical distances
- **envir**: 'environment' containing SQLite database and other globals

Value

gwaa.data-class object generated from the Mega2R database

Examples

```r
## Not run:
require("GenABEL")
db = system.file("exdata", "seqsimm.db", package="Mega2R")
ENV = read.Mega2DB(db)
seqsimgwaa = Mega2GenABEL(markers=ENV$markers[1:10,])

str(seqsimgwaa)
head(summary(seqsimgwaa))

## End(Not run)
```
Mega2GenABELtst

**compare two gwaa.data-class objects**

**Description**

Verify by fields, all the fields in two gwaa.data-class objects. Show more detailed marker information iff the coding values are different. (When comparing two gwaa.data-class objects, one native and one created via **Mega2R** sometimes when an allele frequency is .5 for both alleles, the allele order 1/2 vs 2/1 cannot be currently be determined.)

**Usage**

Mega2GenABELtst(mega_ = mega, gwaa_ = srdta, full = TRUE, envir = ENV)

**Arguments**

- `mega_` name of first gwaa.data-class object
- `gwaa_` name of second gwaa.data-class object
- `full` if TRUE convert genotypes to text as.character(gwaa_@gtdata) and as.character(mega_@gtdata). Then standardize the order for heterozygous alleles and finally compare. This step is optional because it can be rather slow.
- `envir` ’environment’ containing SQLite database and other globals

**Value**

None

**Examples**

```r
## Not run:
db = system.file("exdata", "seqsim.db", package="Mega2R")
require("GenABEL")
ENV = read.Mega2DB(db)

y = Mega2ENVGenABEL()
Mega2GenABELtst(y, y, full = FALSE)

## End(Not run)

## Not run:
# donttestcheck: if you have more time, try ...  
x = Mega2GenABEL()
Mega2GenABELtst(x, y, full = FALSE)

## End(Not run)
```
Mega2pedgene

Execute the pedgene function on a transcript ranges

Description

Execute the pedgene function on the first gs default gene transcript ranges (gs = 1:100). Update the envir$pedgene_results data frame with the results.

Usage

Mega2pedgene(gs = 1:100, genes = NULL, envir = ENV)

Arguments

gs a subrange of the default transcript ranges over which to calculate the Dopedgene function.
genesis a list of genes over which to calculate the DOpedgene function. The value "*" means use all the transcripts in the selected Bioconductor database. If genes is NULL, the gs range of the internal refRanges will be used.

envir 'environment' containing SQLite database and other globals

Value

None the data frame with the results is stored in the environment and named pedgene_results, viz. envir$pedgene_results

See Also

init_pedgene

Examples

db = system.file("exdata", "seqsimm.db", package="Mega2R")
ENV = init_pedgene(db)
ENV$verbose = TRUE
Mega2pedgene(gs = 50:60)
**Mega2R**

### Mega2R package

**Description**
This package reads a Mega2 SQLite3 database into data frames and makes the contained genotypes/phenotypes/linkage data available for analysis.

**Author(s)**
Robert V. Baron and Daniel E. Weeks

### Mega2R-SQLite3 tables

**Description**
This character vector indicates the names of the Mega2 SQLite3 database tables to load. (Not all of the existing tables are loaded.)

**Usage**
```
TBLS
```

**Format**
An object of class character of length 15.

**Author(s)**
Robert V Baron

### Mega2R-SQLite3 table filter

**Description**
This list contains named values. The name corresponds to an SQLite database table. The value is a character string of column names from the "named" table that should be fetched. A table is in this list, if not all the database table columns are needed. The columns for each table are separated by commas.

**Usage**
```
TBLSFilter
```
Format

An object of class list of length 7.

Note

For the data base tables not in this list, all columns are stored in the corresponding data frame.

Author(s)

Robert V Baron

<table>
<thead>
<tr>
<th>Mega2RVersion</th>
<th>Mega2R version</th>
</tr>
</thead>
</table>

Description

This string indicates the current release of Mega2R

Usage

Mega2RVersion

Format

An object of class character of length 1.

Author(s)

Robert V Baron

<table>
<thead>
<tr>
<th>Mega2SKAT</th>
<th>execute the CRAN SKAT function on a subset of the gene transcripts</th>
</tr>
</thead>
</table>

Description

Execute the SKAT function on the first gs default gene transcripts (gs = 1:100). Update the envir$SKAT_results data frame with the results.

Usage

Mega2SKAT(f, ty, gs = 1:100, genes = NULL, skat = SKAT::SKAT, envir = ENV, ...)
Arguments

- **f**: SKAT_Null_Model formula. If this is non NULL, envir$obj is initialized by calling SKAT_Null_Model(f, out_type = ty). If you need to specify additional arguments to the Model viz. (data, Adjustment, n.Resampling, type.Resampling) or need to use a different model viz. SKAT_NULL_emmaX, SKAT_Null_Model_chrX set the formula to NULL, then before Mega2SKAT is called, build the model you need and assign it to ENV$obj.

- **ty**: type of phenotype C/D = Continuous/Binary 5 (internal type 1/2)

- **gs**: a subrange of the default transcripts (refRanges) over which to calculate the DOSKAT function.

- **genes**: a list of genes over which to calculate the DOSKAT function. The value, "*", means use all the transcripts in the selected Bioconductor database. If genes is NULL, the gs range of the internal refRanges will be used.

- **skat**: alternate SKAT function, viz. SKATBinary, SKAT_CommonRare. If it is also necessary is to pass additional arguments to the SKAT function, they may be added to the end of the Mega2SKAT function and will be passed. See examples

- **envir**: 'environment' containing SQLite database and other globals

... extra arguments for SKAT

Value

None the data frame with the results is stored in the environment and named SKAT_results, viz. envir$SKAT_results

Note

The SKAT_Null_Model is called if the formula, f, is not NULL. A helper function SKAT3arg is defined for the 3 argument callback function which in turn calls DOSKAT with the appropriate arguments (including those additional to the Mega2SKAT function).

See Also

- init_SKAT

Examples

```r
db = system.file("exdata", "seqsim.db", package="Mega2R")
ENV = init_SKAT(db, verbose = FALSE, allMarkers = FALSE)
ENV$verbose = FALSE
ENV$SKAT_results = ENV$SKAT_results[0, ]
Mega2SKAT(ENV$phe[, 3] - 1 - 1, "D", kernel = "linear.weighted",
weights.beta = c(0.5, 0.5), gs=50:60 )
```

# donttestcheck: try this below if there is time

```r
Mega2SKAT(ENV$phe[, 3] - 1 - 1, "D", kernel = "linear.weighted",
weights.beta = c(0.5, 0.5), genes=c("CEP104") )
```
ENVSKAT_results

Mega2VCF

**generate a VCF file set for a collection of markers**

**Description**

Generate a VCF file from the specified Mega2 SQLite database. The file is named "prefix".vcf. If the markers argument is NULL, the entire `envir$markers` set is used, otherwise markers argument MUST be rows of the markers (`envir$markers`) data frame. In addition, several other files are generated to hold additional database information: "prefix".fam, "prefix".freq, "prefix".map, "prefix".phe, and "prefix".pen, which contain the pedigree, allele frequency, marker genetic and physical map position, member phenotype and phenotype penetrance data.

**Usage**

```
Mega2VCF(prefix, markers = NULL, mapno = 0, alleleOrder = "default", 
         envir = ENV)
```

**Arguments**

- `prefix` prefix of output files including the VCF file (see Description section above). This prefix can include a path.
- `markers` markers selected to be in the VCF output file
- `mapno` specify which map index to use for genetic distances. The function `showMapNames()` will print out the internal map numbers corresponding to all the maps in the Mega2 database.
- `alleleOrder` how to order alleles in VCF file. ‘default’ is Mega2order, ‘minor’ is minor allele freq first, ‘major’ is major allele freq first, and ‘name’ is ascending ascii character order of allele name.
- `envir` ‘environment’ containing SQLite database and other globals

**Value**

None

**Note**

This code in this package illustrates how to extract the various kinds of data in the Mega2 data frames to use for further processing. Some of the data internal representations are a bit quirky but the code "explains" it all.
Examples

```r
db = system.file("exdata", "seqsim.db", package="Mega2R")
ENV = read.Mega2DB(db)
vcfdir = file.path(where_mega2rtutorial_data(), "vcfr")
if (!dir.exists(vcfdir)) dir.create(vcfdir)
vcffile = file.path(where_mega2rtutorial_data(), "vcfr", "vcf.01")
Mega2VCF(vcffile, ENV$markers[ENV$markers$chromosome == 1, ][1:10,], envir = ENV)
list.files(vcfdir)
```

---

**Description**

Generate a data frame with a row for each person. The observations are:

- **pedigree**: family pedigree name
- **person**: person name
- **father**: father of person
- **mother**: mother of person
- **sex**: sex of person
- **trait**: value of case/control phenotype for person

**Usage**

```r
mkfam(brkloop = FALSE, traitname = "default", envir = ENV)
```

**Arguments**

- **brkloop**: I haven’t needed to set this TRUE yet. Maybe never will. If loops are broken, a person will be replaced by a dopple ganger in the same family with a different father/mother. The number of persons per family will be different when there are broken loops. Also, the person_link numbers will be different for all the persons after the first loop is broken.

- **traitname**: Name of the trait to use as case/control value; by default, "default"

- **envir**: An 'environment' that contains all the data frames created from the SQLite database.

**Value**

data frame that is described above

**Note**

The columns of this data frame come by selecting the values after merging the data frames: `pedigree_table`, `person_table`, and `trait_table`. 
Also, the father and mother columns from person_table are translated from the row index in the person_table to the corresponding name.

This function stores the data frame in the 'environment' and also returns it. The function setfam() stores the data frame into the 'environment' and adjusts the genotype_table and the phenotype_table.

Examples

```r
db = system.file("exdata", "seqsim.db", package="MegaR")
ENV = read.Mega2DB(db)

fam = mkfam()

fam
```

---

**Description**

Create the markers data frame. It contains 5 observations:

- **locus_link**: locus offset of this marker
- **locus_link_fill**: locus offset plus an accumulating fudge factor that jumps with each new chromosome because the count of markers per chromosome is force to be a multiple of 4. (This value corresponds to the offset of the marker in the unified_genotype_table.)
- **MarkerName**: name of the marker
- **chromosome**: chromosome number of the marker
- **position**: base pair position of the marker (selected by bpPosMap[below])

**Usage**

```r
mkMarkers(bpPosMap = 1, envir = ENV)
```

**Arguments**

- `bpPosMap` An integer that indicates the map (index) to use to merge the chromosome/position fields from the map_table data frame to the marker_table data frame. See showMapNames() for the string name to index mapping.
- `envir` an environment that contains all the data frames created from the SQLite database.

**Details**

Select a map (index) from the map_table to merge with the select marker_table data frame to make the marker data frame. See showMapNames() for the string name to index mapping.
**mkphenotype**

**Value**
None

**Examples**

```r
db = system.file("exdata", "seqsim.db", package="Mega2R")
ENV = read.Mega2DB(db, verbose = FALSE)

mkMarkers(1)

ENV$markers
```

---

**mkphenotype**

*generate a phenotype data frame*

**Description**

Convert data in phenotype_table to a data frame of columns that are phenotypes. The columns may be affection status or quantitative values.

**Usage**

`mkphenotype(envir)`

**Arguments**

- `envir` "environment" containing SQLite database and other globals

**Value**

is a data frame with FID column, then IID column, and then an additional column for each phenotype.

**Examples**

```r
db = system.file("exdata", "seqsim.db", package="Mega2R")
ENV = read.Mega2DB(db)
out = mkphenotype()

out
```
read.Mega2DB  

*load Mega2 database and initialize family data frame and markers data frame*

**Description**

Call `dbmega2` with the specified database and create an 'environment', with the SQLite table data loaded into data frames. Also run `mkfam()` to create the pedigree data frame `fam` and then store it with `setfam()`. `setfam()` modifies the `unified_genotype_table` (and `phenotype_table`) to match the family members that remain.

**Usage**

```r
read.Mega2DB(db, ...)
```

**Arguments**

- `db` specify SQLite database to load
- `...` additional arguments to pass to `dbmega2`.

**Value**

an 'environment' that contains all the data frames created from the SQLite database.

**Note**

By default, `mkfam` will remove one of each person that was replicated to break loops in the pedigree, see `mkfam` for details. If you want to leave loops broken, the code is available, but you will have to write your own version of `read.Mega2DB` with a different invocation of `mkfam()`.

**Examples**

```r
db = system.file("exdata", "seqsimm.db", package="Mega2R")
ENV = read.Mega2DB(db, verbose = TRUE)
```

---

**setAnnotations**  

*set default name of transcription database and name of database mapping gene name to entrez gene id*

**Description**

This function takes two string parameters: one to specify entrez gene ids to transcripts, the other to map gene names to entrez gene id’s.
Usage

setAnnotations(txdb, entrezGene, envir = ENV)

Arguments

taxdb name of Bioconductor transcription database.
entrezGene name of Bioconductor mapping of gene name or gene alias to entrez gene id
envir an 'environment' that contains all the data frames created from the SQLite database.

Value

None

Note

Mega2R will take care to load the necessary databases, but you will have to install them from Bioconductor. This is explained at length in the package Vignette.

Examples

db = system.file("exdata", "seqsim.db", package="Mega2R")
ENV = read.Mega2DB(db)

setAnnotations("TxDb.Hsapiens.UCSC.hg19.knownGene", "org.Hs.eg.db")

ENV$txdb
ENV$entrezGene

---

Description

You should first modify the fam data frame to filter the members you need to remove. (For example, you might want to delete members that have an unknown case/control status.) This function takes a new data frame of pedigree information and replaces the fam data frame in the 'environment' with it. Additionally, changing fam data frame will filter the genotypes data frame to only contain persons matching those in the fam data frame. setfam also filters for the phenotype data records.

Usage

setfam(fam, envir = ENV)
Arguments

fam
data frame of family information filtered from \texttt{fam} data frame (generated by \texttt{mkfam}).

envir
an 'environment' that contains all the data frames created from the SQLite database.

Value

None

Examples

\begin{verbatim}
db = system.file("exdata", "seqsim.db", package="Mega2R")
ENV = read.Mega2DB(db)

fam = mkfam()
# remove founders
fam = fam[ !( (fam[, 5] == fam[, 6]) & (fam[, 5] == 0)), ]
setfam(fam)

ENV$fam
\end{verbatim}

---

setRanges

\emph{set default range data: chromosome and start/end base pair}

Description

This function sets the default list of ranges used by \texttt{applyFnToRanges}. \texttt{applyFnToRanges} examines each range and the set of markers that fall within the range will be processed.

Usage

\begin{verbatim}
setRanges(ranges, indices, envir = ENV)
\end{verbatim}

Arguments

ranges
a data frame that contains at least 4 observations: a name, a chromosome, a start base pair position and an end base pair position.

indices
a vector of 3 or 4 integers that specify the chromosome column, start base pair column and end base pair column of range data frame and lastly the name column. If the vector only contains 3 integers, a name will be generated from the three range elements and it will be appended to the ranges and the last range column will be added to the indices.

envir
an 'environment' that contains all the data frames created from the SQLite database.
showMapNames

Value

None

Examples

db = system.file("exdata", "seqsim.db", package="R2Mega")
ENV = read.Mega2DB(db)

ranges = matrix(c(1, 2240000, 2245000,
                  1, 2245000, 2250000,
                  1, 3760000, 3761000,
                  1, 3761000, 3762000,
                  1, 3762000, 3763000,
                  1, 3763000, 3764000,
                  1, 3764000, 3765000,
                  1, 3765000, 3763760,
                  1, 3763760, 3767000,
                  1, 3767000, 3768000,
                  1, 3768000, 3769000,
                  1, 3769000, 3770000),
ncol = 3, nrow = 12, byrow = TRUE)

setRanges(ranges, 1:3)

ENV$refRanges

ranges = matrix(c(1, 2240000, 2245000,
                  1, 2245000, 2250000,
                  1, 3760000, 3761000,
                  1, 3761000, 3762000,
                  1, 3762000, 3763000,
                  1, 3763000, 3764000,
                  1, 3764000, 3765000,
                  1, 3765000, 3763760,
                  1, 3763760, 3767000,
                  1, 3767000, 3768000,
                  1, 3768000, 3769000,
                  1, 3769000, 3770000),
ncol = 3, nrow = 12, byrow = TRUE)

ranges = data.frame(ranges)
ranges$name = LETTERS[1:12]

names(ranges) = c("chr", "start", "end", "name")

setRanges(ranges, 1:4)

ENV$refRanges

---

showMapNames  

show the association between mapno and mapname
showMega2ENV

Description
Mega2R allows several different physical and genetic maps to be stored and used to select positions. This function shows the association between map number and map name.

Usage
showMapNames(envir = ENV)

Arguments
envir an environment that contains all the data frames created from the SQLite database.

Value
None

Examples
db = system.file("exdata", "seqsim.db", package="Mega2R")
ENV = read.Mega2DB(db)

showMapNames()
showPhenoNames

Examples

db = system.file("exdata", "seqsimm.db", package="Mega2R")
ENV = read.Mega2DB(db)

showMega2ENV()

showPhenoNames  show the association between index no and phenotype

Description

Mega2R stores several phenotypes, both affective and quantitative. This function displays the mapping between phenotype (name), index, and the phenotype type (affective or quantitative).

Usage

showPhenoNames(envir = ENV)

Arguments

envir

an environment that contains all the data frames created from the SQLite database.

Value

None

Examples

db = system.file("exdata", "seqsimm.db", package="Mega2R")
ENV = read.Mega2DB(db)

showPhenoNames()

uniqueFamMember  regenerate fam data frame with unique values in member column

Description

Reads the fam data frame in "envir" and returns a new one with unique entries in the member column.

Usage

uniqueFamMember(envir = ENV)
where_mega2rtutorial_data

Arguments

envir 'environment' containing SQLite database and other globals

Value

a data frame with columns the same as the "fam" data frame but with the member column containing unique entries

See Also

mkfam

Examples

db = system.file("exdata", "seqsimm.db", package="Mega2R")
ENV = read.Mega2DB(db)
setfam(uniqueFamMember(envir = ENV))

directory = where_mega2rtutorial_data()
Index

*Topic datasets
   Mega2R-TBLS, 29
   Mega2R-TBLSFilt., 29
   Mega2RVersion, 30
  applyFnToGenes, 3
  applyFnToMarkers, 5
  applyFnToRanges, 6
  clean_mega2rtutorial_data, 8
  computeDosage, 9
  dbmega2_import, 10
  DoFamSKATRC, 9, 10, 24
  DoPedgene, 12, 21
  DOSKAT, 13
  dump_mega2rtutorial_data, 15
  gdsfmt, 25
  getgenotypes, 15
  getgenotypesdos, 16
  getgenotypesgenabel, 17
  getgenotypesraw, 19
  init_famSKATRC, 11, 20, 24
  init_pedgene, 13, 21, 28
  init_SKAT, 14, 22, 31
  Mega2ENVGenABEL, 23
  Mega2famSKATRC, 11, 21, 24
  Mega2gdsfmt, 25
  Mega2GenABEL, 26
  Mega2GenABELtst, 27
  Mega2pedgene, 21, 28
  Mega2R, 29
  Mega2R-package (Mega2R), 29
  Mega2R-TBLS, 29
  Mega2R-TBLSFilt., 29
  Mega2RVersion, 30
  Mega2SKAT, 14, 22, 30
  Mega2VCF, 32
  mkfam, 21, 33, 42
  mkMarkers, 34
  mkphenotype, 35
  read.Mega2DB, 36
  setAnnotations, 36
  setfam, 37
  setRanges, 38
  showMapNames, 39
  showMega2ENV, 40
  showPhenoNames, 41
  TBLS (Mega2R-TBLS), 29
  TBLSFilt (Mega2R-TBLSFilt), 29
  uniqueFamMember, 41
  where_mega2rtutorial_data, 42