Package 'FamAgg'

November 12, 2026

Type Package

Title Pedigree Analysis and Familial Aggregation

Version 1.40.0

Author J. Rainer, D. Taliun, C.X. Weichenberger

Maintainer Johannes Rainer <johannes.rainer@eurac.edu>

URL https://github.com/EuracBiomedicalResearch/FamAgg

BugReports https://github.com/EuracBiomedicalResearch/FamAgg/issues

Imports gap (>= 1.1-17), Matrix, BiocGenerics, utils, survey

Depends methods, kinship2, igraph

Suggests BiocStyle, knitr, RUnit, rmarkdown

VignetteBuilder knitr

Description Framework providing basic pedigree analysis and plotting
utilities as well as a variety of methods to evaluate
familial aggregation of traits in large pedigrees.

Collate Classes.R Generics.R Deprecated.R Constructors.R
Methods-FAData.R Methods-FAKinGroupResults.R
Methods-FAKinSumResults.R Methods-FAGenIndexResults.R
Methods-FAIncidenceRateResults.R
Methods-FAStdIncidenceRateResults.R Methods-FABinTestResults.R
utils.R matched-controls.R Methods.R plotting-functions.R
import-export.R zzz.R

biocViews Genetics

License MIT + file LICENSE

Copyright Haplopainter1.043.pl (http://haplopainter.sourceforge.net)
Copyright (c) 2004-2008 Holger Thiele

NeedsCompilation no

RoxygenNote 7.1.1

git_url https://git.bioconductor.org/packages/FamAgg

git_branch RELEASE_3_23

git_last_commit e5d4b24

git_last_commit_date 2026-04-28

Repository Bioconductor 3.23

Date/Publication 2026-11-11

Contents

FABinTestResults-class 2
FAData-class 5
FAGenIndexResults-class 14
FAIncidenceRateResults-class 19
FAKinGroupResults-class 24
FAKinSumResults-class 29
FAProbResults-class 33
FAStdIncidenceRateResults-class 37
getAll 43
kinshipPairs 46
PedigreeAnalysis 48
PedigreeUtils 57
plotPed 63

Index **67**

`FABinTestResults-class`

Binomial test for familial aggregation

Description

The `FABinTestResults` object contains the results from a simple binomial to test whether the number of affected in a trait are higher than expected by chance. For more details on the method please see `binomialTest`.

Usage

```
## S4 method for signature 'FABinTestResults'
result(object, method="BH")

## S4 replacement method for signature 'FABinTestResults'
trait(object) <- value
```

Arguments

(in alphabetic order)

method	The multiple hypothesis testing method. All methods supported by p.adjust are allowed.
object	The FABinTestResults object.
value	For trait<-: can be a named numeric, character or factor vector. The names (at least some of them) have to match the ids in the pedigree of the object.

Details

A call to the setter methods trait<- resets any test results present in the result slot, thus, the object can be re-used to perform a simulation analysis using the new trait data.

Value

Refer to the method and function description above for detailed information on the returned result object.

Objects from the Class

FABinTestResults objects are returned by the binomialTest function.

Extends

Class FAData directly.

Slots

result The results data.frame.

Methods and Functions

result Returns the result from the test as a data.frame with columns:

"trait_name": the name of the trait.

"total_phenotyped": total number of phenotyped individuals in the trait.

"total_affected": total number of affected individuals in the trait.

"family": the family id. If a global test is used (i.e. if the pedigree consists of a single family, or global = TRUE was provided, the column shows "full pedigree").

phenotyped: the number of phenotyped individuals in the family.

affected: the number of affected individuals in the family.

pvalue: the p-value from the binomial test (conducted using the binom.test function).

prob: the probability of being affected. Either a *local* probability calculated based on all affected and phenotyped individuals in the whole pedigree, or a *global* (population) probability that has to be provided with argument prob.

padj: the p-value adjusted for multiple hypothesis testing using the method defined with argument "method".

trait<- Set the trait information. This method will reset all simulation results saved in the sim slot.

Note

Subsetting (using the [operator) is not supported.

Author(s)

Johannes Rainer, Christian Weichenberger

See Also

FAData, kinship, trait, probabilityTest, kinshipGroupTest, kinshipSumTest, genealogicalIndexTest, familialIncidenceRateTest, fsirTest, plotPed

Examples

```
##########################
##
##  Perform the analysis
##
## Load the test data.
data(minnbreast)

## Subset to some families and generate a pedigree data.frame.
mbsub <- minnbreast[minnbreast$famid == 4 | minnbreast$famid == 5 |
                    minnbreast$famid == 6 | minnbreast$famid == 7 |
                    minnbreast$famid == 8, ]
PedDf <- mbsub[, c("famid", "id", "fatherid", "motherid", "sex")]
colnames(PedDf) <- c("family", "id", "father", "mother", "sex")

## Generate the FAData.
fad <- FAData(pedigree=PedDf)

## Specify the trait.
tcancer <- mbsub$cancer
names(tcancer) <- mbsub$id

## Perform the test:
bir <- binomialTest(fad, trait=tcancer, traitName="cancer")
result(bir)

## Calculating the probability of being affected from the whole data set.
prob <- sum(minnbreast$cancer, na.rm = TRUE) / sum(!is.na(minnbreast$cancer))
bir <- binomialTest(fad, trait = tcancer, prob = prob)
result(bir)

## Plot the pedigree of the family with the smallest p-value.
plotPed(bir, family = "8")
```

FAData-class *Pedigree data information*

Description

FAData objects conveniently store pedigree along with trait information. This object is the central data structure from the FamAgg package. Basic usage pedigree analysis methods are described on this page and on the PedigreeUtils help page, familial aggregation analysis methods on the PedigreeAnalysis help page.

See the section about the pedigree data.frame below for a detailed description of the encoding of missing trait data or founder individuals in FamAgg.

Usage

```
## S4 method for signature 'FAData'
affectedIndividuals(object)

## S4 method for signature 'FAData'
age(object)

## S4 replacement method for signature 'FAData'
age(object) <- value

## S4 method for signature 'FAData'
buildPed(object, id=NULL, family = NULL, max.generations.up=3,
                            max.generations.down=16, prune=FALSE, ...)

## S4 method for signature 'FAData'
export(object, con, format="ped", ...)

FAData(pedigree, age, trait, traitName, header=FALSE, sep="\t", id.col="id",
       family.col="family", father.col="father", mother.col="mother",
       sex.col="sex")

## S4 method for signature 'FAData'
family(object, id=NULL, family=NULL,
                          return.type="data.frame")

## S4 method for signature 'FAData'
kinship(id, ...)

## S4 method for signature 'FAData'
pedigree(object, return.type="data.frame")

## S4 replacement method for signature 'FAData'
pedigree(object) <- value
```

```
## S4 method for signature 'FAData'
pedigreeSize(object)

## S4 method for signature 'FAData'
phenotypedIndividuals(object)

## S4 method for signature 'FAData'
plotPed(object, id=NULL, family=NULL, filename=NULL,
                          device="plot", symbol.related=NA,
                          proband.id=NULL, highlight.ids=NULL,
                          only.phenotyped=FALSE,
                          label1=age(object), label2=NULL, label3=NULL,
                          ...)

## S4 method for signature 'FAData'
show(object)

## S4 method for signature 'FAData'
trait(object, na.rm=FALSE)

## S4 replacement method for signature 'FAData'
trait(object) <- value
```

Arguments

(in alphabetic order)

age	For FAData: either a character(1) specifying the file name from which the age should be read or a named numeric vector of ages with the names corresponding to the ids of the individuals in the pedigree.
con	For export: the file name or connection to a file to which the pedigree information should be exported.
device	For plotPed: the device of file format in which the plot should be saved. See details for allowed values.
family	For buildPed: the id of the family for which the pedigree should be returned. For family: the id of the family for which the pedigree should be returned (full pedigree of the family). For plotPed: the id of the family for which the pedigree should be plotted.
family.col	For FAData: the name of the column containing the id of the families.
father.col	For FAData: the name of the column containing the id of the father.
filename	For plotPed: a character string specifying the name of the file to which the plot should be saved. If none is submitted, the plot is saved to a temporary file.
format	For export: the format in which the pedigree should be exported. At present only "ped" and "fam" are exported, i.e. the file formats from plink (http://pngu.mgh.harvard.edu/~purcell/plink/data.shtml).

header — For FAData: only used if argument pedigree is a character(1), i.e. the file name from which the pedigree should be read. The header argument is passed to the read.table function, i.e. should be set to TRUE if the file contains column headers.

highlight.ids — A list of character vector(s) of ids that should be labeled. The name(s) of the character vector(s) is/are used as the text to label the individuals (the text is shown below the symbol of the individuals). Up to 3 character vectors are supported. Alternatively, a single character vector of ids can be submitted in which case the individuals are labeled with an asterisc ("*").

id — For method kinship: the FAData object from which the kinship matrix should be extracted, for all other methods the id of the individual.

For method plotPed: the id of the individual for which the pedigree should be built (see buildPed) and plotted.

Note: id can be a numeric or a character. Numeric ids are internally converted to character.

id.col — For FAData: the name of the column containing the id of the individuals.

label1 — For plotPed: labels that should be plotted below the symbol for each individual. Should be either a named vector with names corresponding to the ids of the individuals in the pedigree or a vector of the same length than individuals that are to be plotted. For the former it is sufficient to just specify the labels for the individuals that should be shown.

label2 — For plotPed: see label1. The labels are plotted in the second line below the symbol if HaploPainter is used to generate the plot, or on the top left corner of the individual's symbol for kinship2 plotting.

label3 — For plotPed: see label1. The labels are plotted in the third line below the symbol if HaploPainter is used to generate the plot, or on the top right corner of the individual's symbol for kinship2 plotting.

max.generations.down — For buildPed: the maximal number of generations to look for children.

max.generations.up — For buildPed: the maximal number of generations to look for ancestors.

mother.col — For FAData: the name of the column containing the id of the mother.

na.rm — For trait: whether missing values in trait should be returned or not.

object — The FAData object.

only.phenotyped — Wheter only phenotyped individuals, i.e. individuals with a non-NA value in column affected (the trait information). Requires this information to be present.

pedigree — For FAData: either a data.frame with the pedigree information or a character(1) specifying the file name from which the pedigree should be read. See description below for more details.

proband.id — For plotPed: character vector with the id(s) of one ore more individuals that should be highlighted as probands. HaploPainter indicates probands with a "P" next to the symbol and an arrow pointing to the symbol.

prune	For buildPed: whether the smallest possible (connected) pedigree for the submitted ids should be build. This makes only sense if more than one id is submitted.
return.type	Either "data.frame" or "pedigree" if the pedigree information should be returned as a data.frame or pedigreeList object as defined in the kinship2 package.
sep	For FAData: only used if argument pedigree is a character(1), i.e. the file name from which the pedigree should be read. The sep argument is passed to the read.table function and specifies the field separator.
sex.col	For FAData: the name of the column spefifying the sex of the individuals.
symbol.related	For plotPed: the symbol which should be used to label individuals sharing kinship with the id for which the pedigree is generated and plotted.
trait	For FAData: a numeric vector with 0, 1 and NA or a logical vector indicating unaffected (but phenotyped), affected and not phenotyped individuals.
traitName	For FAData: an optional name for the trait.
value	For age<-: a named numeric vector. The names (at least some of them) have to match the ids in the pedigree of the object. For pedigree<-: For trait<-: a named numeric vector with 0, 1 and NA or a logical vector with FALSE, TRUE, NA for not affected, affected and not tested. The names (at least some of them) have to match the ids in the pedigree of the object.
...	Additional arguments to be passed to the plotting functions (doPlotPed for plotPed).

Details

See sections below for a description of the individual methods.

The buildPed method is a combination of the methods getAncestors, getChildren and getMissingMate, i.e. it first gets all ancestors for the specified id(s), determines then the children of all of the ids (submitted ids and their ancestors) and at last looks for any missing mates/spouses to complete the pedigree.

The plotPed function uses either the external perl program HaploPainter or the plotting capabilities of the kinship2 package. With HaploPainter, as it is an external too, it is not possible to display the plot directly, but each plot is automatically saved to a file (either "pdf", "ps", "svg" or "png"; can be specified with the device parameter). HaploPainter plotting supports also device = "txt" in which case the data table is exported (in the format expected by HaploPainter) to a tabulator delimited text file and the name of this text file is returned - no plot is created. Plotting with kinship2 (the default) allows to display the plot (device="plot") or export it to a file (device="pdf" or device="png").

The switchPlotfun function can be used to change the plotting system.

Value

Refer to the method and function description above for detailed information on the returned result object.

Objects from the Class

FAData objects are created by the constructor function FAData and should not be directly created by a call to new.

Slots

age A (named) numerical vector with the age of the individuals. It is suggested to use the getter and setter methods described below to access this slot.

pedigree A data.frame with the pedigree. It is suggested to use the getter and setter methods described below to access this slot.

.kinship The kinship matrix for the kinship of each individual in the pedigree with each other. This slot should not be accessed directly, but the kinship method should be used instead.

traitname The name of the trait being stored in the object.

.trait A numerical vector with the trait information, 0, 1, NA, for phenotyped but not affected, affected and not tested, respectively. This slot should not be accessed directly, but the trait and trait<- methods should be used instead that ensure that the data is matched to the information in the pedigree.

Constructors, importing and exporting data

FAData Constructor function to create a new FAData instance. In addition to submitting the pedigree information as data.frame, pedigree or pedigreeLinst it is possible to specify the name of the file from which the pedigree information should be read. The recognizes and imports plink *ped* and fam files (http://pngu.mgh.harvard.edu/~purcell/plink/data.shtml) or from generic text files. For the latter, arguments header, sep, family.col, id.col, father.col, mother.col and sex.col allow to further specify which columns of the file contain what information etc. If argument pedigree is a data.frame, the column names "family", "id", "father", "mother" and "sex" are expected. Any additional columns are dropped.

The sex is expected to be encoded either as a numeric 1 (male), 2 (female) with any other number or NA representing unknown, or as a character vector or factor with "M", "m", "Male" or "male" for male and "F", "f", "Female" or "female" for female.

export Export pedigree data to a file.

Accessors and subsetting

object$name Access name column in the pedigree of the FAData object. The function returns a named vector wirh the names corresponding to the ids of the individuals or NULL if name does not correspoind to a column name in the pedigree. The trait data can be accessed either by object$trait or object$affected.

age Returns the age of the individuals as a named numeric vector. If the pedigree is set, the order of the values corresponds always to the ordering of the individuals in the pedigree with NA for individuals for which the age is unknown. In case the age was never set it returns a vector of NAs with length equal to the number of individuals.

age<- Setter for the age. Value has to be a named numeric vector.

pedigree Returns the pedigree either as a `data.frame` or a `pedigreeList` object (defined in the `kinship2` package) depending on the value of the parameter `return.type` (i.e. either `return.type="data.frame"` or `return.type="pedigree"`). If `pedigree` is called on any other object than a `FAData` object (or any object that inherits from that object), the `pedigree` method from the `kinship2` package is called.

For the default return type (i.e. `return.type="data.frame"`) a `data.frame` is returned with the following columns: `"family"`: the ID of the family, `"id"`: the ID of the individual, `"father"`: the ID of the individual's father. Founder individuals, i.e. individuals for whom the father and mother is not known in the data set, contain a `NA` in this column. `"mother"`: the ID of the individual's father. Founder individuals, i.e. individuals for whom the father and mother is not known in the data set, contain a `NA` in this column. `"sex"`: the sex of the individuals encoded as a `factor` with levels `"M"` and `"F"` for male and female, or `NA` for not known. If trait information is available in the object the returned `data.frame` will also contain a column named `affected` with the information whether the individual is affected (`1`), not affected (`0`) or was not tested/phenotyped `NA`.

pedigree<- Setter for the pedigree slow. Value can be a `data.frame` with columns containing the family id, individual id, father id, mother id and sex (in this order) or a `pedigree` or `pedigreeList` object as defined in the `kinship2` package.

object[i,] Subsets the `FAData` object to individuals specified with `i` which can be a logical, numeric or character vector. For the latter, the elements have to be the ids of the individuals (i.e. rownames of `pedigree(object)`). Returns the sub-setted object. Note that subsetting other than by family might result in a non-valid pedigree (e.g. if mother or father ID are not available in the sub-setted pedigree).

trait Get the trait vector from the object. By default, the ordering is the same as pedigree, setting argument `na.rm=TRUE` removes all `NA` values, thus the ordering and length might be different. Returns a named vector with the names corresponding to the ids of the individuals.

trait<- Setter for the trait slot. Can be a named numeric vector (values `0`, `1` and `NA`) or logical vector (values `FALSE`, `TRUE` and `NA`) with the names matching the ids of the individuals in the pedigree. The method internally matches and re-orders the trait vector to match the ordering of the ids in the pedigree.

Basic usage

affectedIndividuals Returns a character vector with the ids of the affected individuals, i.e. the id of the individuals with a value other than 0 or `NA` in the trait. If no trait data is available the method returns `NULL`.

buildPed Builds a pedigree for the specified id(s) containing generations defined by `max.generations.up` and `max.generations.down` and returns it as a `data.frame`. The pedigree contains all individuals in the family sharing kinship with the input individual(s) and mates needed to complete the pedigree. For `prune=TRUE` the function tries to find the smallest connected pedigree for all the submitted ids.

family Returns the pedigree for a full family. In contrast to `buildPed` which constructs a (sub)pedigree for a specific individual, this method returns the pedigree of the complete family for an individual (if `id` is specified). The function returns either a `data.frame` or a `pedigreeList` with the pedigree for the family.

kinship Extracts the pre-calculated kinship matrix, i.e. a symmetric matrix with the kinship between all individuals in the pedigree. The matrix is calculated using the `kinship` method

provided by the kinship2 package [Sinwell (2014)]. The function returns a dsCMatrix from the Matrix package.

pedigreeSize Returns the size, i.e. the number of individuals (rows) in the pedigree.

phenotypedIndividuals Returns a character vector with the ids of the phenotyped individuals, i.e. the id of all individuals that have a non-NA value in thetrait. If no trait data is available the method returns NULL.

plotPed Creates the pedigree for the submitted id(s) or family and plots it (i.e. saves it to the specified file). See details above for more information. Returns the file name of the file to which the pedigree plot was exported or NULL for kinship2 plotting and device="plot".

For HaploPainter plotting and device = "txt" the name of the file to which the plotting data has been exported is returned.

See doPlotPed for more information.

Pedigree analysis methods

Methods for familial aggregation and other pedigree analysis methods are described on the PedigreeAnalysis help page.

Pedigree utilities

A variety of different pedigree utilities are defined for FAData objects. For the full list of methods see the PedigreeUtils help page.

Note

The ids of individuals, father, mother and family can be either numeric or characters, internally, all ids will however be handled as characters.

The pedigree<- setter method removes all white spaces in columns "id", "family", "father" and "mother" of the pedigree.

Author(s)

Johannes Rainer.

References

Sinwell JP, Therneau TM & Schaid DJ (2014) The kinship2 R package for pedigree data. *Human heredity* 78:91-93.

See Also

pedigree, FAProbResults, FAKinGroupResults, FAKinSumResults, FAGenIndexResults, doPlotPed, PedigreeUtils, getAll, PedigreeAnalysis

Examples

```
##########################
##
##  Create a new FAData object
##
## Load the Minnesota Breast Cancer record and subset to the
## first families.
data(minnbreast)
mbsub <- minnbreast[minnbreast$famid==4 | minnbreast$famid==5, ]
mbped <- mbsub[, c("famid", "id", "fatherid", "motherid", "sex")]
## Renaming column names
colnames(mbped) <- c("family", "id", "father", "mother", "sex")
## Defining the optional argument age.
Age <- mbsub$endage
names(Age) <- mbsub$id
## Create the object
fad <- FAData(pedigree=mbped, age=Age)

fad

## Extract the ids directly...
head(fad$id)

## Extract the kinship matrix
dim(kinship(fad))

## What's the size of the pedigree?
pedigreeSize(fad)

## Importing a "ped" file.
pedFile <- system.file("txt/minnbreastsub.ped.gz", package="FamAgg")
## Quick glance at the file.
readLines(pedFile, n=1)
fad <- FAData(pedFile)

head(pedigree(fad))

## Creating the FAData reading data from a txt file.
pedFile <- system.file("txt/minnbreastsub.txt", package="FamAgg")
fad <- FAData(pedigree=pedFile, header=TRUE, id.col="id",
              family.col="famid", father.col="fatherid",
              mother.col="motherid")
## Adding the age
age(fad) <- Age
fad
## List all families in the pedigree along with the number of
## individuals
table(fad$family)

##########################
##
##  Basic usage
```

```
##
## Extracting the pedigree information
ped <- pedigree(fad)
## By default the pedigree is returned as a data.frame.
class(ped)
head(ped)

## In addition, we can extract the pedigree as a pedigreeList
pedigree(fad, return.type="pedigree")

## Return the ids of all ancestors of individual 6
## up to 3 generations
getAncestors(fad, id="6")

## Build the pedigree for individual 6: this includes all of its
## children and all of its ancestors up to the maximal number of
## specified generations.
buildPed(fad, id=6)
## Which is a sub-pedigree of the complete family:
family(fad, id=6)

## In addition we can specify manually some ids in the pedigree and
## generate the smallest possible pedigree containing all ids:
buildPed(fad, id=c(6, 23, 28), prune=TRUE)

## Get the list of all ids sharing kinship with individuals
## 5 and 9
shareKinship(fad, id=c("5", "9"))

## Subset the fad to family "4"
subFad <- fad[fad$family == "4", ]
subFad

## Export the pedigree from this family to a ped file
tmpFile <- tempfile()
export(subFad, con=tmpFile, format="ped")

head(read.table(tmpFile, sep="\t"))

##########################
##
##  Plotting
##
## Plot the pedigree for individual 6.
plotPed(fad, id=6)

## Alternatively, exporte it to a temporary file
pfile <- plotPed(fad, id=6, device="pdf")
pfile

## Highlighting some of the individuals:
## first get to know which other individuals are in the pedigree
plotPed(fad, id=6, highlight.ids=list(hello=c(1, 2, 4)))
```

```
#########################
##
##  Adding trait data
##
fad <- FAData(pedigree=mbped, age=Age)
tcancer <- mbsub$cancer
names(tcancer) <- mbsub$id
trait(fad) <- tcancer
## Now we can plot the pedigree also showing the affected status.
plotPed(fad, id=6)


## Alternatively, create the FAData with the trait data
fad <- FAData(pedigree=mbped, trait=mbsub$cancer, traitName="cancer")
plotPed(fad, id=6)
```

FAGenIndexResults-class

Genealogical Index

Description

The genealogical index [Hill, 1980], also referred to as the *genealogical index of familiality* (GIF) in the literature, is a method to identify familial clustering of diseases or other traits. For a given trait, the method computes the mean kinship between affected in the whole pedigree along with mean kinships of randomly drawn sets of individuals. The distribution of average kinship values among the control sets is used to estimate the probability that the observed level of kinship among the cases is due to chance.

Usage

```
## S4 method for signature 'FAGenIndexResults'
plotPed(object, id=NULL, family=NULL,
                                    filename=NULL, device="plot", ...)

## S4 method for signature 'FAGenIndexResults'
plotRes(object, id=NULL, family=NULL,
                                    addLegend=TRUE, type="density", ...)

## S4 method for signature 'FAGenIndexResults'
result(object, method="BH")

## S4 method for signature 'FAGenIndexResults'
runSimulation(object, nsim=50000,
                                          perFamilyTest=FALSE,
```

```
                                                controlSetMethod="getAll",
                                               rm.singletons=TRUE, strata=NULL, ...)

## S4 replacement method for signature 'FAGenIndexResults'
trait(object) <- value
```

Arguments

(in alphabetic order)

addLegend	For plotRes: if a legend should be added to the plot.
controlSetMethod	For runSimulation: the method (i.e. name of the function) that should be used to define the set of (eventually matched) control individuals from which the random samples are taken. Supported functions are getAll, getSexMatched and getExternalMatched. For perFamilyTest=TRUE also getGenerationMatched and getGenerationSexMatched are supported. Note: for getExternalMatched, a numeric, character or factor vector to be used for the matching has to be submitted to runSimulation as additional argument match.using.
device	For plotPed: see plotPed for more details.
family	For plotPed: the family for which the pedigree should be plotted. For plotRes: the family for which the genealogical index analysis simulation results should be shown. Only supported if perFamilyTest=TRUE.
filename	For plotPed: the file name to which the pedigree plot should be exported. See plotPed for more details.
id	For plotPed: the id of an indiviual from a family for which the pedigree should be plotted. For plotRes: the id of an individual from a family for which the genealogical index analysis simulation results should be shown. Only supported if perFamilyTest=TRUE.
method	The multiple hypothesis testing method. All methods supported by p.adjust are allowed.
nsim	Number of simulations.
perFamilyTest	For runSimulation: whether the test should be performed on the whole pedigree (default) or separately within each family. In the latter case the test evaluates the presence of clustered affected individuals within each family.
rm.singletons	For runSimulation: whether unconnected individuals in the pedigree (i.e. singletons) should be removed.
object	The FAGenIndexResults object.
strata	For runSimulation: a numeric, character of factor characterizing each individual in the pedigree. The length of this vector and the ordering has to match the pedigree. This vector allows to perform stratified random sampling. See details or examples for more information.
type	For plotRes: either "density" (the default) or "hist" specifying whether the distribution of expected values from the simulation should be visualized as a density plot or histogram.

value — For trait<-: can be a named numeric, character or factor vector. The names (at least some of them) have to match the ids in the pedigree of the object.

... — For plotPed: additional arguments to be submitted to the internal buildPed call and to plotPed.

For runSimulation: additional arguments passed to the choosen controlSetMethod function (e.g. match.using for getExternalMatched).

Details

This implementation differs from the original method from Hill as it allows, in addition to perform per family analyses, to use also stratified sampling and allows a more flexible definition of the set of matched control individuals. The controlSetMethod parameter allows to specify a method to define the matched control set (e.g. matched by sex or matched by any externally provided vector).

Stratified sampling allows to even further fine tune the selection of matched controls. Assuming that in a pedigree the group of affected consists of 5 females and 3 male individuals, passing the sex of all individuals to the function (e.g. strata=fad$sex, with fad being the FAData object containing the pedigree to be analyzed) results in random sets with the same proportion of male/female individuals (i.e. 5 females, 3 males).

Note that, if strata is specified, all individuals with a missing value in strata (also affected individuals) are excluded from the analysis.

Note that by default singletons (i.e. unconnected individuals in the pedigree) are removed from the pedigree prior the analysis. Set rm.singletons=FALSE if you do not want them to be removed.

By default, the genealogical index is calculated on the whole pedigree, but it is also possible to evaluate within-family clustering of cases by specifying perFamilyTest=TRUE. In that case, it is also possible to use the getGenerationMatched and getGenerationSexMatched functions to define the set of matched controls from which random samples will be taken.

A call to the setter methods trait<- resets any simulation results present in the sim slot, thus, the object can be re-used to perform a simulation analysis using the new trait data.

Value

Refer to the method and function description above for detailed information on the returned result object.

Objects from the Class

FAGenIndexResults objects are created calling the genealogicalIndexTest method on a FAData object.

Extends

Class FAData directly.

Slots

controlSetMethod A character specifying the name of the method used to define the set of control individuals from which random samples were taken.

nsim Number of simulations.

perFamilyTest Logical indicating whether a per-family test was performed.

sim The result of the simulation. This slot should not be accessed directly, use the `result` method to extract result information.

Methods and Functions

`plotPed` Plots a pedigree for one of the affected individuals in the simulation results. The id of the selected affected individual (specified with argument `id`) is highlighted in red. See `plotPed` for more details.

`plotRes` Plots the results from a genealogical index simulation analysis. The distribution of the mean kinship values of the randomly drawn controls are displayed as a grey density plot, the observed mean kinship value of all affected as a blue vertical line.

Returns the result from the simulation as a `data.frame` with columns: `"trait_name"`: the name of the trait. `"total_phenotyped"`: total number of individuals in the pedigree phenotyped in the analyzed trait. `"total_affected"`: total number ofindividuals in the pedigree that are affected in the analyzed trait (i.e. number of cases). `"entity_id"`: the id for the analyzed entity, being either the whole pedigree (in which case the id will be `"1"`) or the id of the family (if `perFamilyTest=TRUE`). `"entity_ctrls"`: the number of (matched) control individuals from which the random samples were drawn. `"entity_affected"`: the number of affected individuals in the entity. This number can differ from the number of affected, if `strata` was specified and some of the affected have a missing value in `strata`. `"genealogical_index"`: the genealogical index of familiality (gif), i.e. the mean kinship value between all affected in the entity (pedigree or family). To be consistent with the original implementations, the genealogical index is the mean kinship multiplied with 100000. `"pvalue"`: the p-value for the significance of the mean kinship. `"padj"`: the p-value adjusted for multiple hypothesis testing (with the method specified with argument `method`).

The returned `data.frame` is sorted by column `"pvalue"`, its rownames correspond to column `"entity_id"`.

result **Simulation** Performs the simulation analysis based on the pedigree and trait information stored in the object. Returns a `FAGenIndexResults` object with the results of the simulation.

trait<- Set the trait information. This method will reset all simulation results saved in the `sim` slot.

Note

Subsetting (using the `[` operator) is not supported.

Author(s)

Johannes Rainer

References

Hill, J. R. (1980) A survey of cancer sites by kinship in the Utah Mormon population. In Cairns J, Lyon JL, Skolnick M (eds): *Cancer Incidence in Defined Populations. Banbury Report 4.* Cold Spring Harbor, NY: Cold Spring Harbor Laboratory Press, pp 299–318.

See Also

FAData, trait, probabilityTest, kinshipGroupTest, kinshipSumTest, familialIncidenceRateTest, fsirTest, plotPed

Examples

```
##########################
##
##  Perform the simulation analysis
##
## Load the Minnesota Breast Cancer data set.
data(minnbreast)

## Subset to some families and generate a pedigree data.frame
mbsub <- minnbreast[minnbreast$famid == 4 | minnbreast$famid == 14 |
                  minnbreast$famid == 6 | minnbreast$famid == 8, ]
PedDf <- mbsub[, c("famid", "id", "fatherid", "motherid", "sex")]
colnames(PedDf) <- c("family", "id", "father", "mother", "sex")

## Generate the FAData.
fad <- FAData(pedigree=PedDf)

## Specify the trait.
tcancer <- mbsub$cancer
names(tcancer) <- mbsub$id

## Perform the test with default settings, i.e. use all individuals
## in the pedigree as control set from which random samples are drawn
## and perform the analysis on the whole pedigree.
gi <- genealogicalIndexTest(fad, trait=tcancer, traitName="cancer",
                            nsim=1000,)
## Just show some information
gi

## Show the results
result(gi)

## Plot the observed mean kinship and the distribution of the mean kinship of
## random samples.
plotRes(gi)

## Plot the pedigree for one of the families. All individuals
## used as matched control set are highlighted in red.
plotPed(gi, family="8")

## Repeat the analysis using the sex as strata. This will result in stratified
## random sampling with the number of female and male individuals selected in
## each permutation corresponding to the numbers below
table(gi$sex[affectedIndividuals(gi)])
giStrata <- runSimulation(gi, nsim=1000, strata=gi$sex)
result(giStrata)
```

```
## Alternatively, we can use "getSexMatched" as the function to define the set
## of control individuals. Just, in the present case both male and females
## individuals will be selected since also there are male and female individuals
## among the affected cases.
giPerFam <- runSimulation(gi, nsim=1000, controlSetMethod="getSexMatched",
                          perFamilyTest=TRUE)
result(giPerFam)

## For those families in which there are only female cases, random samples
## were drawn among only female individuals (within the same family). These
## are highlighted in red in the pedigree plot:
plotPed(giPerFam, family="14", cex=0.5)

## Plot the simulation result for this family:
plotRes(giPerFam, family="14")
```

FAIncidenceRateResults-class
Familial Incidence Rate

Description

The FAIncidenceRateResults object contains the results from a familial incidence rate calculation employing in addition Monte Carlo simulations to assess significance levels for the familial incidence rate of each individual. The familial incidence rate (FIR, also referred to as FR in Kerber (1995)) is an estimate for the risk per gene-time for each individuals for a certain disease given the disease experience in the cohort. The measure considers the kinship of each individual with any affected individual in the pedigree and the time at risk for each.

Note that in contrast to e.g. FAKinSumResults a familial incidence rate and corresponding p-value are calculated and available for all individuals in the pedigree, not only for affected individuals.

Usage

```
## S4 method for signature 'FAIncidenceRateResults'
familialIncidenceRate(object, trait=NULL,
                                                    timeAtRisk=NULL,
                                                    ...)

## S4 method for signature 'FAIncidenceRateResults'
plotPed(object, id=NULL, family=NULL,
                                          filename=NULL, device="plot",
                                          only.phenotyped=FALSE, ...)

## S4 method for signature 'FAIncidenceRateResults'
plotRes(object, id=NULL, family=NULL,
                                          addLegend=TRUE, type="density", ...)
```

```
## S4 method for signature 'FAIncidenceRateResults'
result(object, method="BH")

## S4 method for signature 'FAIncidenceRateResults'
runSimulation(object, nsim=50000,
                                                timeAtRisk=NULL,
                                                strata=NULL, ...)

## S4 method for signature 'FAIncidenceRateResults'
timeAtRisk(object)

## S4 replacement method for signature 'FAIncidenceRateResults'
timeAtRisk(object) <- value

## S4 replacement method for signature 'FAIncidenceRateResults'
trait(object) <- value
```

Arguments

(in alphabetic order)

addLegend	For plotRes: if a legend should be added to the plot.
device	For plotPed: see plotPed for more details.
family	For plotPed: the family for which the pedigree should be plotted. For plotRes: not supported.
filename	For plotPed: the file name to which the pedigree plot should be exported. See plotPed for more details.
id	For plotPed and plotRes: the id of the indiviual for which the pedigree or the simulation result should be plotted. Note: id can be a numeric or a character. Numeric ids will be internally converted to character.
method	The multiple hypothesis testing method. All methods supported by p.adjust are allowed.
nsim	Number of simulations.
object	The FAIncidenceRateResults object.
only.phenotyped	For plotPed: Wheter only phenotyped individuals, i.e. individuals with a non-NA value in column affected (the trait information). If TRUE, the function removes all non-phenotyped individuals, keeping only those that are required for the pedigree to be complete.
strata	For runSimulation: a numeric, character of factor characterizing each individual in the pedigree. The length of this vector and the ordering has to match the pedigree. This vector allows to perform stratified random sampling. See details on the PedigreeAnalysis help page or examples for more information.

timeAtRisk	For runSimulation: a numeric vector specifying the time at risk for each individual in the given trait. See also estimateTimeAtRisk, an utility function to estimate time at risk. For familialIncidenceRate: not used.
trait	For familialIncidenceRate: not used.
type	For plotRes: at present only "density" is supported.
value	For trait<-: can be a named numeric, character or factor vector. The names (at least some of them) have to match the ids in the pedigree of the object. For timeAtRisk: a numeric vector with the time at risk for each individual.
...	For plotPed: additional arguments to be submitted to the internal buildPed call and to plotPed. For runSimulation: additional arguments prune and lowMem. See below for details.

Details

Monte Carlo simulation and empirical p-value estimation: the background distribution to calculate the p-value for a familial incidence rate (FIR) is determined by randomly sampling N affected individuals (N being the number of affected) and calculating the *expected* FIR for all individuals in each simulation iteration. The p-value for an individual represents thus the number of times an expected FIR for that individual from the simulation was found to be larger than or equal to the observed FIR divided by the number of iterations.

Calling the runSimulation method on a FAIncidenceRateResults object is the same as calling the familialIncidenceRateTest on a FAData object. In the first case the simulation is performed using the trait information data stored internally in the object, while in the latter case the trait information have to be submitted to the function call.

By providing argument strata, the stratified random sampling is performed. See example below and the details section in PedigreeAnalysis for more details.

The familial incidence rate can also be directly calculated, without simulation, using the familialIncidenceRate method of a FAData object.

A call to the setter methods trait<- resets any simulation results present in the sim slot, thus, the object can be re-used to perform a simulation analysis using the new trait data.

By default the Monte Carlo p-value estimation in the runSimulation method is quite memory demanding. For very large pedigrees the optional argument lowMem=TRUE might be passed to the method which results in faster and less memory demanding calculations. This will however disable the plotRes method on the resulting FAIncidenceRateResults as the distribution of familial incidence rates from the simulation runs is no longer reported.

Note: the FIR for singletons and individuals that do not share kinship with at least one other phenotyped individual that has also a valid value in argument timeAtRisk (and eventually strata) will be NA.

Value

Refer to the method and function description above for detailed information on the returned result object.

Objects from the Class

FAIncidenceRateResults objects are created by the familialIncidenceRateTest method on a FAData object.

Extends

Class FAData directly.

Slots

nsim Number of simulations.

sim The result of the simulation. This slot should not be accessed directly, use the result method to extract result information.

timeAtRisk Numeric vector with the time at risk for each individual. Use the accessor method timeAtRisk or use object$tar to extract this data.

Methods and Functions

object$name Access the familial incidence rate using object$fir, the (raw) p-value from the simulation using object$pvalue and the time at risk for each individual using object$tar or object$timeAtRisk with object being the FAIncidenceRateResults object.

familialIncidenceRate Returns the familial incidence rate values calculated by a call to the runSimulation method or familialIncidenceRate method on a FAData object. In contrast to that latter method, which directly calculates the values, this method returns the values from a calculation stored inside the FAIncidenceRateResults object.

The method returns a named numeric vector with the familial incidence rates for all individuals in the pedigree, the names being the ID of the individuals. Singletons as well as individuals that, after removing not phenotyped individuals or individuals without time at risk, do not share kinship with any other individual in the pedigree have a value of NA.

plotPed Plots a pedigree for one of the affected individuals in the simulation results. The id of the selected affected individual (specified with argument id) is highlighted in red. The familial incidence rate value for each individual is drawn below the individual's id. See plotPed for more details.

plotRes Plots the distribution of expected familial incidence rates calculated for the selected individuals from Monte Carlo simulations along with the actually observed familial incidence rate.

result Returns the result from the simulation as a data.frame with columns:

"trait_name": the name of the trait.

"total_phenotyped": total number of phenotyped individuals in the trait.

"total_affected": total number of affected individuals in the trait.

"total_tested": the number of individuals in the pedigree considered for the simulation. This corresponds to all individuals with valid, non-NA, values in trait, timeAtRisk and eventually strata.

"id": the id of the individual.

"family": the family id.

"fir": the familial incidence rate. Note that this will be

NA for all non-phenotyped individuals and singletons in the pedigree as well as for individuals that do not share kinship with at least one other phenotyped individual with valid time at risk (or valid value in parameter strata).

"pvalue": the p-value for the significance of the familial incidence rate assessed by Monte Carlo simulations.

"padj": the p-value adjusted for multiple hypothesis testing (with the method specified with argument method).

The returned data.frame is sorted by column "pvalue", its row names correspond to column "id".

runSimulation Performs the simulation analysis based on the pedigree and trait information stored in the object as well as the time at risk provided with argument timeAtRisk. Returns a FAIncidenceRateResults object with the results from the simulation.

trait<- Set the trait information. This method will reset all simulation results saved in the sim slot.

Note

Subsetting (using the [operator) is not supported.

Author(s)

Johannes Rainer

References

Kerber, R.A. (1995) Method for calculating risk associated with family history of a disease. *Genet Epidemiol*, pp 291–301.

See Also

FAData, kinship, trait, probabilityTest, kinshipGroupTest, kinshipSumTest, genealogicalIndexTest, familialIncidenceRateTest, fsirTest, plotPed, estimateTimeAtRisk

Examples

```
##########################
##
##  Perform the simulation analysis
##
## Load the test data.
data(minnbreast)

## Subset to some families and generate a pedigree data.frame.
mbsub <- minnbreast[minnbreast$famid == 4 | minnbreast$famid == 5 |
                    minnbreast$famid == 6 | minnbreast$famid == 7 |
                    minnbreast$famid == 411, ]
PedDf <- mbsub[, c("famid", "id", "fatherid", "motherid", "sex")]
colnames(PedDf) <- c("family", "id", "father", "mother", "sex")

## Generate the FAData.
fad <- FAData(pedigree=PedDf)
```

```
## Specify the trait.
tcancer <- mbsub$cancer
names(tcancer) <- mbsub$id

## Spefify the "time at risk"; we are using column "endage"
tar <- mbsub$endage

## Perform the simulation test:
far <- familialIncidenceRateTest(fad, trait=tcancer, traitName="cancer",
                                 timeAtRisk=tar, nsim=1000)
head(result(far))

## We can easily extract the actual FIR values:
head(far$fir)
## Or
head(familialIncidenceRate(far))

## Access the p-value directly.
head(far$pvalue)

## Access the time at risk
head(timeAtRisk(far))
head(far$tar)
head(far$timeAtRisk)

## Plot the pedigree for a family with significant FIRs.
## The numbers below the IDs of the individuals represent the actual
## FIR values.
plotPed(far, family=result(far)$family[1])

## Plot also the result from the simulation run.
plotRes(far, id=result(far)$id[1])
```

`FAKinGroupResults-class`

Kinship group test

Description

The `FAKinGroupResults` object contains the results from the *kinship test*. This test performs a familial aggregation analysis on a subset of individuals within a family. Two actual tests are conducted, a *ratio test* that evaluates whether the number of affected individuals within the group is higher than expected by chance, and a *kinship test* that compares the largest kinship value between affected in the group to the one between randomly sampled individuals.

For more details see `kinshipGroupTest`.

Usage

```
## S4 method for signature 'FAKinGroupResults'
affectedKinshipGroups(object)

## S4 method for signature 'FAKinGroupResults'
buildPed(object, id=NULL, max.generations.up=3,
                                    max.generations.down=16, prune=FALSE)


## S4 method for signature 'FAKinGroupResults'
plotPed(object, id=NULL, family=NULL,
                                   filename=NULL, device="plot", ...)

## S4 method for signature 'FAKinGroupResults'
plotRes(object, id=NULL, family=NULL,
                                   addLegend=TRUE, type="density", ...)

## S4 method for signature 'FAKinGroupResults'
result(object, method="BH")

## S4 method for signature 'FAKinGroupResults'
runSimulation(object, nsim=50000, strata=NULL)

## S4 method for signature 'FAKinGroupResults'
shareKinship(object, id=NULL, rmKinship=0)

## S4 replacement method for signature 'FAKinGroupResults'
trait(object) <- value
```

Arguments

(in alphabetic order)

addLegend	For plotRes: if a legend should be added to the plot.
device	For plotPed: see plotPed for more details.
family	For plotPed: not supported.
filename	For plotPed: the file name to which the pedigree plot should be exported. See plotPed for more details.
id	For buildPed, plotPed, plotRes and shareKinship: the id of the kinship group (i.e. one of the ids in column "group_id" of the result table result(object)). Note: id can be a numeric or a character. Numeric ids will be internally converted to character.
max.generations.down	For buildPed: the maximal number of generations to look for children.
max.generations.up	For buildPed: the maximal number of generations to look for ancestors.

method	The multiple hypothesis testing method. All methods supported by p.adjust are allowed.
nsim	Number of simulations.
object	The FAKinGroupResults object.
prune	For buildPed: whether the full pedigree should be returned (prune=FALSE) or the pedigree should be reduced to a smaller pedigree containing only individuals in the kinship group (prune=TRUE); see details for more information. Note: the plotPed method does also support this parameter.
rmKinship	For shareKinship: Restrict reporting pairs to those that have kinship values >rmKinship. See shareKinship below for more details.
strata	For runSimulation: a numeric, character of factor characterizing each individual in the pedigree. The length of this vector and the ordering has to match the pedigree. This vector allows to perform stratified random sampling. See details on the PedigreeAnalysis help page or examples for more information.
type	For plotRes: either "density" (the default) or "hist" specifying whether the distribution of expected values from the simulation should be visualized as a density plot or histogram.
value	For trait<-: can be a named numeric, character or factor vector. The names (at least some of them) have to match the ids in the pedigree of the object.
...	For plotPed: additional arguments to be submitted to the internal buildPed call and to plotPed.

Details

Calling the runSimulation method on a FAKinGroupResults object is the same as calling the kinshipGroupTest on a FAData object. In the first case the simulation is performed using the trait information data stored internally in the object, while in the latter case the trait information have to be submitted to the function call.

A call to the setter methods trait<- resets any simulation results present in the sim slot, thus, the object can be re-used to perform a simulation analysis using the new trait data.

The buildPed method returns by default the full pedigree (all ancestors and all children) up to the maximal number of generations. By setting prune=TRUE the method restricts the pedigree to all individuals with a kinship >= the minimal kinship between the individual (with the id equal to the group id) and any other affected individual in its pedigree.

The plotPed method allows to plot the pedigree for a kinship group. This pedigree consists of the full pedigree (all ancestors and children) of all individuals in the kinship group. Similar to the buildPed method the pedigree can be restricted to the kinship group (and eventual missing parents etc) by setting prune=TRUE.

Value

Refer to the method and function description above for detailed information on the returned result object.

Objects from the Class

FAKinGroupResults objects are created by the kinshipGroupTest method for FAData objects.

Extends

Class FAData directly.

Slots

nsim Number of simulations.

sim The list containing the result of the simulation. Element "pvalueKinship" contains the p-values from the kinship test, "pvalueRatio" the p-values from the ratio test, "expDensity" and "expHist" density and hist objects representing the background distribution from the Monte Carlo simulation. This slot should not be directly accessed, use the result method to extract result information.

affectedKinshipGroups A list of lists, each element representing one kinship group, $aff the ids of all affected persons in that group sharing kinship with the indivudual (being the id of the group), $phe ids of all phenotyped individuals in the pedigree with kinship to the individual up to a kinship being smaller or equal to the smallest kinship of the indivudal with any other affected in that group. $kinfreq: a table with the frequency (counts) of kinship values (smaller 0.5). $meankin: the mean kinship in that group.

Methods and Functions

affectedKinshipGroups Get groups of affected individuals in the pedigree along with all phenotyped individuals with kinship larger or equal to the smallest kinship between affected individuals in the group. Returns a list of lists with elements aff, phe, kinfreq and meankin for each list item: aff: character vector with the ids of all affected in the group, phe: character vector with the ids of all phenotyped in the group, kinfreq: a table with the frequency (counts) of kinship values (self-self kinships removed; the names of the table correnspond to the kinship values ordered increasingly), and meankin: the mean kinship value in the group. The names of the list correspond to the id of the affected individual for which the affected kinship group was determined.

buildPed Builds the pedigree for the submitted id (which represents the id of the group, i.e. one of names(affectedKinshipGroups(object)), respectively ids in column "group_id" of result(object)). Building a pedigree by submitting the family id is not supported. See details below for more information. The resulting pedigree is returned as a data.frame.

plotPed Plots a pedigree for one of the affected kinship groups in the simulation results. The ids of all individuals of the affected kinship group are highlighted in the plot in red. See plotPed for more details on the plotting and details below for additional settings.

plotRes Plots the distribution of counts of randomly sampled affected counts within the kinship group along with the actually observed kinship sum.

result Returns the result from the simulation as a data.frame with columns: "trait_name": the name of the trait. "total_phenotyped": the total number of phenotyped individuals in the trait. "total_affected": the total number of affected individuals in the trait. "phenotyped": the number of phenotyped individuals in all analyzed kinship groups. "affected": the number of affected individuals in all analyzed kinship groups. "group_id": the id for the kinship group (represents the id of one of the affected individuals in the group). "family": the family id of the affected/group. "group_phenotyped": the number of phenotyped individuals in the current kinship group. "group_affected": the number of affected individuals in the current kinship group. "ratio_pvalue": the p-value from the ratio test. "ratio_padj": the p-value

from the ratio test adjusted for multiple hypothesis testing using the method specified with argument `method`. `"mean_kinship"`: the mean kinship value between all individuals in the current kinship group. `"kinship_pvalue"`: the p-value from the kinship test. `"kinship_padj"`: the p-value from the kinship test adjusted for multiple hypothesis testing using the method specified with argument `method`.

The `data.frame` is sorted by column `"ratio_pvalue"`, its rownames correspond to the `"group_id"`.

runSimulation Performs the simulation analysis based on the pedigree and trait information stored in the object. Optionally allows to perform stratified sampling. Returns a `FAKinGroupResults` with the results of the simulation.

shareKinship Returns a character vector with ids of all individuals that share kinship with any of the individuals in the kinship group identified by the argument `id`. If `rmKinship` is specified, only individuals with a kinship >`rmKinship` to the group defined by `id` will be reported. This esentially restricts only the inclusion of individuals outside of the group. Everyone inside the group will be reported independently of the threshold defined by `rmKinship`. This feature has mainly been implemented for reasons of API compatibility with the remaining versions of `shareKinship`.

trait<- Set the trait information. This method will reset all simulation results saved in the `sim` slot.

Note

Subsetting (using the `[` operator) is not supported.

Author(s)

Johannes Rainer, Daniel Taliun

See Also

FAData, kinship, trait, probabilityTest, kinshipSumTest, genealogicalIndexTest, familialIncidenceRateTest, fsirTest, buildPed, plotPed, switchPlotfun

Examples

```
##########################
##
##  Perform the simulation analysis
##
## Load the test data.
data(minnbreast)

## Subset to some families and generate the pedigree data.frame
mbsub <- minnbreast[minnbreast$famid == 165 | minnbreast$famid == 432, ]
PedDf <- mbsub[, c("famid", "id", "fatherid", "motherid", "sex")]
colnames(PedDf) <- c("family", "id", "father", "mother", "sex")

## Generate the FAData.
fad <- FAData(pedigree=PedDf)

## Specify the trait.
tcancer <- mbsub$cancer
```

```
names(tcancer) <- mbsub$id

## Perform the kinship group test.
far <- kinshipGroupTest(fad, trait=tcancer, traitName="cancer", nsim=1000)
res <- result(far)
head(res)

## Plot the pedigree for the most significant kinship group
plotPed(far, id=res[1, "group_id"])

## The full pedigree for this affected individual and its kinship group is
## large:
nrow(buildPed(far, id=res[1, "group_id"]))

## We can however restrict it to a reduced pedigree containing only the
## kinship group and all individuals with a kinship >= the smallest kinship
## between the individual and any other affected individual in the pedigree:
nrow(buildPed(far, id=res[1, "group_id"], prune=TRUE))

## By specifying prune=TRUE we can restrict the pedigree plot to these
## individuals
plotPed(far, id=res[1, "group_id"], prune=TRUE)

## Get the ids of all individuals sharing kinship with any of the inddividuals
## in that kinship group:
shareKinship(far, id=res[1, "group_id"])

## Plot the simulation analysis result for the ratio test.
plotRes(far, id=res[1, "group_id"], type="hist")
```

`FAKinSumResults-class` *Kinship sum test*

Description

The `FAKinSumResults` object contains the results from a kinship cluster test which evaluates familial aggregation based on the sum of kinship values between affected cases. This test highlights individuals that exhibit a higher than chance relationship to other affected individuals, therefore highlighting individuals within families aggregating the phenotype. To achieve this, for each affected individual the sum of kinship values to all other affected cases is computed. In a Monte Carlo simulation this is repeated with the same number of cases and the resulting background distribution is used to compute p-values for the kinship sums obtained from the observed cases.

Usage

```
## S4 method for signature 'FAKinSumResults'
plotPed(object, id=NULL, family=NULL,
                                    filename=NULL, device="plot",
```

```
                                    only.phenotyped=FALSE, ...)

## S4 method for signature 'FAKinSumResults'
plotRes(object, id=NULL, family=NULL,
                                    addLegend=TRUE, type="density", ...)

## S4 method for signature 'FAKinSumResults'
result(object, method="BH", cutoff=0.05, rmKinship=0)

## S4 method for signature 'FAKinSumResults'
runSimulation(object, nsim=50000,
strata=NULL, ...)

## S4 replacement method for signature 'FAKinSumResults'
trait(object) <- value
```

Arguments

(in alphabetic order)

addLegend	For plotRes: if a legend should be added to the plot.
cutoff	For result: P-value cutoff for clustering of kinship-related affected individuals based on their padj value. Individuals with p-value lower than this cutoff will be included in the grouping. Each group will have at least one individual that satisfies this threshold, as this is the one that started that group. Others are included due to kinship to this particular individual. This is especially useful for large pedigrees with inaccurate or missing family assignment.
device	For plotPed: see plotPed for more details.
family	For plotPed: not supported.
filename	For plotPed: the file name to which the pedigree plot should be exported. See plotPed for more details.
id	For plotPed and plotRes: the id of the indiviual (i.e. affected individual in the result data.frame) for which the pedigree or the simulation result should be plotted. Note: id can be a numeric or a character. Numeric ids will be internally converted to character.
method	The multiple hypothesis testing method. All methods supported by p.adjust are allowed.
nsim	Number of simulations.
object	The FAKinSumResults object.
only.phenotyped	For plotPed: Wheter only phenotyped individuals, i.e. individuals with a non-NA value in column affected (the trait information). If TRUE, the function removes all non-phenotyped individuals, keeping only those that are required for the pedigree to be complete.
rmKinship	For result: When assigning kinship groups, skip pairs of cases with kinship <= rmKinship.

strata	For runSimulation: a numeric, character of factor characterizing each individual in the pedigree. The length of this vector and the ordering has to match the pedigree. This vector allows to perform stratified random sampling. See details on the PedigreeAnalysis help page or examples for more information.
type	For plotRes: either "density" (the default) or "hist" specifying whether the distribution of expected values from the simulation should be visualized as a density plot or histogram.
value	For trait<-: can be a named numeric, character or factor vector. The names (at least some of them) have to match the ids in the pedigree of the object.
...	For plotPed: additional arguments to be submitted to the internal buildPed call and to plotPed.

Details

Calling the runSimulation method on a FAKinSumResults object is the same as calling the kinshipSumTest on a FAData object. In the first case the simulation is performed using the trait information data stored internally in the object, while in the latter case the trait information have to be submitted to the function call.

A call to the setter methods trait<- resets any simulation results present in the sim slot, thus, the object can be re-used to perform a simulation analysis using the new trait data.

The expected frequency (column "freq") in the result data.frame is NA if the corresponding sum of kinship coefficients reported in column "kinship_sum" was never sampled in the simulation. Still, a p-value can be reported.

The plotPed function does not support to draw pedigrees for individuals for which no simulation test has been performed. To draw a pedigree for any individual (with or without trait information, being affected or not in the trait) refer to the plotPed method for FAData objects.

Value

Refer to the method and function description above for detailed information on the returned result object.

Objects from the Class

FAKinSumResults objects are created by the kinshipSumTest method on a FAData object.

Extends

Class FAData directly.

Slots

nsim Number of simulations.

sim The result of the simulation. This slot should not be accessed directly, use the result method to extract result information.

Methods and Functions

plotPed Plots a pedigree for one of the affected individuals in the simulation results. The id of the selected affected individual (specified with argument `id`) is highlighted in red. See `plotPed` for more details.

plotRes Plots the distribution of kinship sums between random sets of samples individuals from the Monte Carlo simulation along with the actually observed kinship sum for the affected individual specified with parameter `id`. For `id` only affected individuals for which the analysis has been performed are allowed. The ids of these individuals are listed in column `"affected_id"` of the `data.frame` returned by `result`.

result Returns the result from the simulation as a `data.frame` with columns: `"trait_name"`: the name of the trait. `"total_phenotyped"`: total number of phenotyped individuals in the trait. `"total_affected"`: total number of affected individuals in the trait. `"affected_id"`: the id of the affected individual for whom the test has been performed. `"family"`: the family id of the affected. `"ksgrp"`: Numeric identifier that specifies a group of affected individuals related by kinship. Group assignment starts with the top ranking individual (by `padj`), `NA` is assigned to those that did not pass the threshold `cutoff` supplied to `result`. If parameter `rmKinship` is passed, assignment is restricted to kinship values `>rmKinship` between the top ranking individual that founded this group and the rest. Kinship-related individuals that have a lower kinship value will be left unassigned, therefore they may end up in a separate group. `"kinship_sum"`: the sum of kinship values. `"freq"`: the expected frequency of the kinship sum from the simulation. `"pvalue"`: the p-value for the significance of the kinship sum. `"padj"`: the p-value adjusted for multiple hypothesis testing (with the method specified with argument `method`).

The returned `data.frame` is sorted by column `"pvalue"`, its row names correspond to column `"affected_id"`.

runSimulation Performs the simulation analysis based on the pedigree and trait information stored in the object. Returns a `FAKinSumResults` object with the results of the simulation.

trait<- Set the trait information. This method will reset all simulation results saved in the `sim` slot.

Note

Subsetting (using the `[` operator) is not supported.

Author(s)

Johannes Rainer, Christian Weichenberger

See Also

`FAData`, `kinship`, `trait`, `probabilityTest`, `kinshipGroupTest`, `kinshipSumTest`, `genealogicalIndexTest`, `familialIncidenceRateTest`, `fsirTest`, `plotPed`

Examples

```
##########################
##
##  Perform the simulation analysis
##
```

```
## Load the test data.
data(minnbreast)

## Subset to some families and generate a pedigree data.frame.
mbsub <- minnbreast[minnbreast$famid == 4 | minnbreast$famid == 5 |
                  minnbreast$famid == 6 | minnbreast$famid == 7, ]
PedDf <- mbsub[, c("famid", "id", "fatherid", "motherid", "sex")]
colnames(PedDf) <- c("family", "id", "father", "mother", "sex")

## Generate the FAData.
fad <- FAData(pedigree=PedDf)

## Specify the trait.
tcancer <- mbsub$cancer
names(tcancer) <- mbsub$id

## Perform the test:
far <- kinshipSumTest(fad, trait=tcancer, traitName="cancer",
                      nsim=1000)
head(result(far))

## Plot the pedigree for one of affected individuals. The id of the affected
## individual is highlighted in red.
plotPed(far, id=result(far)$affected_id[1])

## Replace the trait, this will delete all simulation results
## on the existing FAKinSumResults object
tpreg <- mbsub$everpreg
names(tpreg) <- mbsub$id
trait(far) <- tpreg

## The analysis can be performed using the runSimulation method.
far <- runSimulation(far, nsim=1000)
head(result(far))

## Plot the pedigree of one of the affected; note that "affected" now
## indicates whether the individual was ever pregnant.
plotPed(far, id="9")

## Plot also the result from the simulation run.
plotRes(far, id="9")
```

FAProbResults-class *DEPRECATED: Probability test*

Description

The FAProbResults object contains the results from the *probability test*. The probability test is only a convience method that calls the gap package's method pfc.sim to compute probabilities

of familial clustering of phenotypes [Yu and Zelterman (2002)]. One drawback of that method is that it is limited to families with at most 22 individuals. Thus, pedigrees need to be split with specialized software such as Jenti [Falchi and Fuchsberger ea. (2008)], which within large families define cliques that can then be used as input to this algorithm.

DEPRECATION WARNING:

Due to problems of the gap package on MS Windows systems, this test will be removed in the next Bioconductor release (3.8).

Usage

```
## S4 method for signature 'FAProbResults'
buildPed(object, id=NULL, max.generations.up=3,
                                  max.generations.down=16, prune=FALSE)

## S4 method for signature 'FAProbResults'
cliqueAndTrait(object, na.rm=FALSE)

## S4 method for signature 'FAProbResults'
cliques(object, na.rm=FALSE)

## S4 replacement method for signature 'FAProbResults'
cliques(object) <- value

## S4 method for signature 'FAProbResults'
plotPed(object, id=NULL, family=NULL,
                                filename=NULL, device="plot", ...)

## S4 method for signature 'FAProbResults'
result(object, method="BH")

## S4 method for signature 'FAProbResults'
runSimulation(object, nsim=50000)

## S4 method for signature 'FAProbResults'
shareKinship(object, id=NULL)

## S4 replacement method for signature 'FAProbResults'
trait(object) <- value

## S4 method for signature 'FAProbResults'
traitByClique(object)
```

Arguments

(in alphabetic order)

device	For plotPed: see plotPed for more details.

family	For plotPed: not supported.
filename	For plotPed: the file name to which the pedigree plot should be exported. See plotPed for more details.
id	For buildPed, plotPed, shareKinship: the id (character or numerif) of the clique (i.e. one of the ids in column "group_id" of the result table result(object)). Note: id can be a numeric or a character. Numeric ids will be internally converted to character.
max.generations.down	For buildPed: the maximal number of generations to look for children.
max.generations.up	For buildPed: the maximal number of generations to look for ancestors.
method	The multiple hypothesis testing method. All methods supported by p.adjust are allowed.
na.rm	Whether NA elements should be returned or not.
nsim	Number of simulations.
object	The FAProbResults object.
prune	For buildPed: whether the full pedigree should be returned (prune=FALSE) or the pedigree should be reduced to the individuals in the corresponding clique (prune=TRUE). Note: the plotPed method does also support this parameter.
value	For cliques<-: can be a named numeric, character or factor vector. The names (at least some of them) have to match the ids in the pedigree of the object.
...	For plotPed: additional arguments to be submitted to the internal buildPed call and to plotPed.

Details

Calling the runSimulation method on a FAProbResults object is the same as calling the probabilityTest on a FAData object. In the first case the simulation is performed using the clique and trait information data stored internally in the object, while in the latter case the clique and trait information have to be submitted to the function call.

A call to the setter methods trait<- or cliques<- resets any simulation results present in the sim slot, thus, the object can be re-used to perform a simulation analysis using the new trait or clique data.

Value

Refer to the method and function description above for detailed information on the returned result object.

Objects from the Class

FAProbResults objects are created by the probabilityTest method of FAData objects.

Extends

Class FAData directly.

Slots

nsim Number of simulations.

sim The result of the simulation. This slot should not be directly accessed, use the `result` method to extract result information.

.cliques A factor with the assignment of individuals to cliques. This slot should not be accessed directly, but the `cliques` and `cliques<-` methods should be used instead that ensure that the data is matched to the information in the pedigree.

Methods and Functions

cliqueAndTrait Get a `data.frame` with the clique ID and the value from the trait for each individual. If `na.rm=TRUE` all rows (individuals) with either a missing clique ID or trait value are removed.

buildPed Builds the pedigree for the submitted id (which represents the id of the group, i.e. the clique (e.g. column `"group_id"` of `result(object)`). By default the method builds the full pedigree for all individuals in the clique but the argument `prune` allows to reduce it to the individuals of the clique. Building a pedigree by submitting the family id is not supported. The pedigree is returned as a `data.frame`.

cliques Returns a `factor` vector representing the clique/group assignment of the individuals. By default, the ordering is the same as pedigree, setting argument `na.rm=TRUE` removes all `NA` values, thus the ordering and length might differ (e.g. if some individuals are not part of any clique).

cliques<- Setter for the cliques vector. Can be a named numeric vector, character vector or factor with the names matching the ids of the individuals in the pedigree. The method internally matches and re-orders the cliques vector to match the ordering of the ids in the pedigree. In addition, the function resets eventually stored simulation results in the `sim` slot.

`plotPed` Plots a pedigree for one of the cliques in the simulation results. The ids of all individuals of the selected clique are highlighted in red. See `plotPed` for more details on the plotting and details below for additional settings.

result Returns the result from the simulation as a `data.frame` with columns: `trait_name`: the name of the trait. `total_phenotyped`: the total number of phenotyped individuals in the trait. `total_affected`: the total number of affected individuals in the trait. `phenotyped`: the number of phenotyped individuals in all group (i.e. cliques specified by the `clique` parameter). `affected`: the number of affected individuals in all group (i.e. cliques specified by the `clique` parameter). `group_id`: the id of the group (clique). `family`: the id of the family in which the clique was defined. `group_phenotyped`: the number of phenotyped individuals in the current group. `group_affected`: the number of affected individuals in the current group. `pvalue`: the p-value from the Monte Carlo simulation. `padj`: the p-value adjusted for multiple hypothesis testing using the method specified with argument `method`.

runSimulation Performs the simulation analysis based on the pedigree, trait and clique information stored in the object. Returns a `FAProbResults` object with the results of the simulation.

shareKinship Returns a character vector with the ids of all individuals that share kinship with any of the individuals in the clique identified by the argument `id`.

trait<- Set the trait information. This method will reset all simulation results saved in the `sim` slot.

traitByClique Summarize the trait information by clique. Returns a `matrix` with the size of the clique (i.e. individuals in the clique with available trait information) and count of individuals in the clique with a trait value other than zero (column `affected_size`). The clique ids are used as rownames of the `matrix`.

Note

Subsetting (using the `[` operator) is not supported.

Author(s)

Johannes Rainer, Daniel Taliun

References

Yu C & Zelterman D (2002) Statistical inference for familial disease clusters. *Biometrics*, pp 481-491

Falchi M & Fuchsberger C (2008) Jenti: an efficient tool for mining complex inbred genealogies. *Bioinformatics*, pp 724-726

See Also

`FAData`, `buildPed`, `plotPed`, `trait`, `probabilityTest`, `kinshipGroupTest`, `kinshipSumTest`, `genealogicalIndexTest`, `familialIncidenceRateTest`, `fsirTest`

FAStdIncidenceRateResults-class

Familial Standardized Incidence Rate

Description

The `FAStdIncidenceRateResults` object contains the results from a familial standardized incidence rate (FSIR) calculation employing in addition Monte Carlo simulations to assess significance levels for the individuals' FSIRs.

The FSIR weights the disease status of relatives based on their degree of relatedness with the proband (Kerber, 1995). Formally, the FSIR is defined as the standardized incidence ratio (SIR) or standardized morality ratio in epidemiology, i.e. as the ratio between the observed and expected number of cases, only that both are in addition also weighted by the degree of relatedness (i.e. kinship value) between individuals in the pedigree.

Usage

```
## S4 method for signature 'FAStdIncidenceRateResults'
fsir(object, trait=NULL, lambda=NULL,
                                          timeInStrata=NULL,
                                          ...)
```

```
## S4 method for signature 'FAStdIncidenceRateResults'
lambda(object, ...)

## S4 method for signature 'FAStdIncidenceRateResults'
plotPed(object, id=NULL, family=NULL,
                                              filename=NULL, device="plot",
                                              only.phenotyped=FALSE, ...)

## S4 method for signature 'FAStdIncidenceRateResults'
plotRes(object, id=NULL, family=NULL,
                                          addLegend=TRUE, type="density", ...)

## S4 method for signature 'FAStdIncidenceRateResults'
result(object, method="BH")

## S4 method for signature 'FAStdIncidenceRateResults'
resultForId(object, id=NULL)

## S4 method for signature 'FAStdIncidenceRateResults'
runSimulation(object, nsim=50000,
                                                    lambda=NULL,
                                                    timeInStrata=NULL,
                                                    strata=NULL,
                                                    ...)

## S4 method for signature 'FAStdIncidenceRateResults'
timeInStrata(object)

## S4 replacement method for signature 'FAStdIncidenceRateResults'
trait(object) <- value
```

Arguments

(in alphabetic order)

addLegend	For plotRes: if a legend should be added to the plot.
device	For plotPed: see plotPed for more details.
family	For plotPed: the family for which the pedigree should be plotted. For plotRes: not supported.
filename	For plotPed: the file name to which the pedigree plot should be exported. See plotPed for more details.
id	For plotPed and plotRes: the id of the indiviual for which the pedigree or the simulation result should be plotted. Note: id can be a numeric or a character. Numeric ids will be internally converted to character. For resultForId: the ID of the individual from which the result should be returned.

`lambda` Numeric vector with the incidence rates per stratum from the population. The length of this vector has to match the number of columns of argument `timeInStrata`.

For `fsir`: not used.

`method` The multiple hypothesis testing method. All methods supported by `p.adjust` are allowed.

`nsim` Number of simulations.

`object` The `FAStdIncidenceRateResults` object.

`only.phenotyped`

For `plotPed`: Wheter only phenotyped individuals, i.e. individuals with a non-NA value in column `affected` (the trait information). If `TRUE`, the function removes all non-phenotyped individuals, keeping only those that are required for the pedigree to be complete.

`strata` For `runSimulation`: a numeric, character of factor characterizing each individual in the pedigree. The length of this vector and the ordering has to match the pedigree. This vector allows to perform stratified random sampling. See details on the `PedigreeAnalysis` help page or examples for more information.

`timeInStrata` For `runSimulation`: a numeric matrix specifying the time at risk for each individual in each strata. Columns represent the strata, rows the individuals, each cell the time at risk for the individual in the respective strata. See example below. The `factor2matrix` could be useful in generating such a table.

For `fsir`: not used.

`trait` For `fsir`: not used.

`type` For `plotRes`: at present only `"density"` is supported.

`value` For `trait<-`: can be a named numeric, character or factor vector. The names (at least some of them) have to match the ids in the pedigree of the object.

`...` For `plotPed`: additional arguments to be submitted to the internal `buildPed` call and to `plotPed`.

For `runSimulation`: additional arguments `prune` and `lowMem`. See below for details.

Details

Monte Carlo simulation and empirical p-value estimation: see details of `FAIncidenceRateResults` as the concept and calculation is essentially identical.

Calling the `runSimulation` method on a `FAStdIncidenceRateResults` object is the same as calling the `fsirTest` on a `FAData` object. In the first case the simulation is performed using the trait information data stored internally in the object, while in the latter case the trait information have to be submitted to the function call.

By providing argument `strata`, the stratified random sampling is performed. See example below and the details section in `PedigreeAnalysis` for more details.

The FSIR can also be directly calculated, without simulation, using the `fsir` method of a `FAData` object.

A call to the setter methods `trait<-` resets any simulation results present in the `sim` slot, thus, the object can be re-used to perform a simulation analysis using the new trait data.

By default the Monte Carlo p-value estimation in the `runSimulation` method is quite memory demanding. For very large pedigrees the optional argument `lowMem=TRUE` might be passed to the method which results in faster and less memory demanding calculations. This will however disable the `plotRes` method on the resulting `FAStdIncidenceRateResults` as the distribution of FSIR from the simulation runs is no longer reported.

Note: the FIR for singletons and individuals that do not share kinship with at least one other phenotyped individual that has also a valid value in argument `timeAtRisk` (and eventually `strata`) will be `NA`.

Value

Refer to the method and function description above for detailed information on the returned result object.

Objects from the Class

`FAStdIncidenceRateResults` objects are created by the `fsirTest` method on a `FAData` object.

Extends

Class `FAData` directly.

Slots

nsim Number of simulations.

sim The result of the simulation. This slot should not be accessed directly, use the `result` method to extract result information.

timeInStrata Numeric matrix specifying the time at risk in each strata (represented by the columns) of each individual (rows). Use the accessor method `timeInStrata` to extract this data.

lambda Numeric vector with the incidence rates per stratum from the population.

Methods and Functions

object$name Access various results and data stored in the object. `name` can be `fsir`, `pvalue`, `lambda` or `timeInStrata` to access the FSIR, the (raw) p-value from the simulation analysis, the lambda or the time in strata.

fsir Returns the FSIR values calculated by a call to the `runSimulation` method or `fsir` method on a `FAData` object. In contrast to that latter method, which directly calculates the values, this method returns the values from a calculation stored inside the `FAStdIncidenceRateResults` object.

The method returns a named numeric vector with the familial incidence rates, the names being the ID of the individuals. Singletons as well as individuals that, after removing not phenotyped individuals or individuals without time at risk, do not share kinship with any other individual in the pedigree have a value of `NA`.

plotPed Plots a pedigree for one of the affected individuals in the simulation results. The id of the selected affected individual (specified with argument `id`) is highlighted in red. The FSIR value for each individual is drawn below the individual's id. See `plotPed` for more details.

plotRes Plots the distribution of expected FSIR calculated for the selected individuals from Monte Carlo simulations along with the actually observed FSIR.

result Returns the result from the simulation as a `data.frame` with columns:

`"trait_name"`: the name of the trait.

`"total_phenotyped"`: total number of phenotyped individuals in the trait.

`"total_affected"`: total number of affected individuals in the trait.

`"total_tested"`: the number of individuals in the pedigree considered for the simulation. This corresponds to all individuals with valid, non-NA, values in trait, timeAtRisk and eventually strata.

`"id"`: the id of the individual.

`"family"`: the family id.

`"fsir"`: the familial standardized incidence rate. Note that this will be

`NA` for all non-phenotyped individuals and singletons in the pedigree as well as for individuals that do not share kinship with at least one other phenotyped individual with valid time at risk (or valid value in parameter `strata`).

`"pvalue"`: the p-value for the significance of the familial standardized incidence rate assessed by Monte Carlo simulations.

`"padj"`: the p-value adjusted for multiple hypothesis testing (with the method specified with argument `method`).

The returned `data.frame` is sorted by column `"pvalue"`, its row names correspond to column `"id"`.

resultForId Extracts results information for a given individual. The method returns a `list` with elements `"id"`, `"fsir"`, `"pvalue"`, `"timeInStrata"` and `"lambda"` with the ID of the individual, the FSIR and corresponding p-value estimated by the simulation, the row from the `timeInStrata` matrix of the individual and the `lambda` (incidence rates from the population).

runSimulation Performs the simulation analysis based on the pedigree and trait information stored in the object as well as the time at risk provided with argument `timeAtRisk`. Returns a `FAStdIncidenceRateResults` object with the results from the simulation.

trait<- Set the trait information. This method will reset all simulation results saved in the `sim` slot.

Note

Subsetting (using the `[` operator) is not supported.

Author(s)

Johannes Rainer

References

Kerber, R.A. (1995) Method for calculating risk associated with family history of a disease. *Genet Epidemiol*, pp 291–301.

See Also

`FAData`, `kinship`, `trait`, `probabilityTest`, `kinshipGroupTest`, `kinshipSumTest`, `genealogicalIndexTest`, `familialIncidenceRateTest`, `fsirTest`, `plotPed`, `estimateTimeAtRisk`

Examples

```
#########################
##
##  Perform the simulation analysis
##
## Load the test data.
data(minnbreast)

## Subset to some families and generate a pedigree data.frame.
mbsub <- minnbreast[minnbreast$famid == 4 | minnbreast$famid == 5 |
                    minnbreast$famid == 6 | minnbreast$famid == 7 |
                    minnbreast$famid == 411, ]
PedDf <- mbsub[, c("famid", "id", "fatherid", "motherid", "sex")]
colnames(PedDf) <- c("family", "id", "father", "mother", "sex")

## Generate the FAData.
fad <- FAData(pedigree=PedDf)

## Specify the trait.
tcancer <- mbsub$cancer
names(tcancer) <- mbsub$id

## Next we have to specify lambda and the timeInStrata matrix.
## For lambda we use information from Cancer Research UK:
## New breast cancer cases in females per year and 100000: 155.3
## New breast cancer cases in males per year and 100000:     1.1
## New prostate cancer cases in females per year and 100000: 0
## New prostate cancer cases in males per year and 100000: 134.3
lbda <- c(M=(1.1+134.4)/100000, F=155.3/100000)

## Next we need the time at risk of each individual in each strata.
## For strata we use male and females, for the time at risk we use
## column "endage":
stratMat <- factor2matrix(fad$sex)
## Next we have to multiply that with the endage, since each person
## spent that time "at risk" to get cancer.
stratMat <- stratMat * mbsub$endage

## Running the simulation
fsirs <- fsirTest(fad, trait=tcancer, traitName="cancer", lambda=lbda,
                  timeInStrata=stratMat, nsim=500)

## Showing some of the results
head(result(fsirs))

## Extract the FSIR
head(fsirs$fsir)

## We can also directly access the p-values from the simulation,
## these are however the raw, unadjusted p-values.
head(fsirs$pvalue)
```

```
## Get the time in strata for each individual
head(fsirs$timeInStrata)

## Plot the pedigree for a family with significant FSIRs.
## The numbers below the IDs of the individuals represent the actual
## FSIR values.
plotPed(fsirs, family=result(fsirs)$family[1])

## Plot also the result from the simulation run.
plotRes(fsirs, id=result(fsirs)$id[1])

## Extract the data and result for an individual
resultForId(fsirs, id="16424")
```

---

getAll *Define sets of control individuals for one or more given individuals*

Description

These functions allow to define sets of (eventually matched) control individuals for one or more given individuals using pedigree data. By default, controls from the same family than the specified individual(s) are returned. They are used e.g. in the genealogicalIndexTest method.

Usage

```
## S4 method for signature 'data.frame'
getAll(object, id=NULL, ...)

## S4 method for signature 'data.frame'
getExternalMatched(object, id=NULL, match.using, ...)

## S4 method for signature 'data.frame'
getGenerationMatched(object, id=NULL,
                                          include.anc=0, include.off=0, ...)

## S4 method for signature 'data.frame'
getGenerationSexMatched(object, id=NULL,
                                            include.anc=0, include.off=0, ...)

## S4 method for signature 'data.frame'
getSexMatched(object, id=NULL, ...)
```

Arguments

(in alphabetic order)

id	A character vector with ids of individuals for whom matched control individuals (from the same family) should be defined.
include.anc	For getGenerationMatched and getGenerationSexMatched: number of ancestor generations of individuals id in which control individuals should be defined, in addition to the actual generation of the individuals id. By default the functions define control individuals only in the actual generation of the individuals in id.
include.off	For getGenerationMatched and getGenerationSexMatched: number of offspring generations of individuals id in which control individuals should be defined, in addition to the actual generation of the individuals id. By default the functions define control individuals only in the actual generation of the individuals in id.
match.using	For getExternalMatched: a named vector for matching of controls. The names of the vector have to correspond to the ids of the individuals in the pedigree (but can have a different ordering).
object	Either a data.frame, pedigree or a pedigreeList object specifying the pedigree. If a data.frame is submitted, the columns id, family, father, mother and sex are required.
...	Additional arguments; not used at present.

Value

Refer to the method and function description above for detailed information on the returned result object.

Get matched controls

getAll Simply returns the ids of all individuals in the family (i.e. individuals with the same family id in the pedigree) of the specified ids in id. Returns a list with the ids (of type character) of the controls. The names of the list correspond to the family id.

getExternalMatched Returns the ids of matched individuals from the same family for the specified ids in id. The match.using vector is for the matching, i.e. the function first extracts the values for the individuals in id from match.using and returns the ids of all individuals whose value in match.using matches the value of the individuals specified by id. Individuals with a missing value in match.using are excluded. Returns a list with the ids (of type character) of the controls. The names of the list correspond to the family id.

getGenerationMatched Returns the ids of individuals matched by the generation of the individuals in id. The function returns the ids of all individuals from the same generation(s) than the individuals in id. The arguments include.anc and include.off can be used to increase the range of generation from which individuals are selected. Returns a list with the ids (of type character) of the controls. The names of the list correspond to the family id.

getGenerationSexMatched Same as getGenerationMatched, but matches in addition individuals by sex (see getSexMatched). Returns a list with the ids (of type character) of the controls. The names of the list correspond to the family id.

getSexMatched Returns ids of individuals from the same family matching the sex of the individuals in id. Individuals with sex being NA are excluded. The result is essentially identical to the

getAll if id contains ids of male and female individuals. Returns a list with the ids (of type character) of the controls. The names of the list correspond to the family id.

Author(s)

Johannes Rainer.

See Also

pedigree, FAData, FAProbResults, FAKinGroupResults, FAKinSumResults, FAGenIndexResults, genealogicalIndexTest

Examples

```
##########################
##
##  Defining a small pedigree
##
## load the Minnesota Breast Cancer record and subset to the
## first families.
data(minnbreast)
mbsub <- minnbreast[minnbreast$famid==4 | minnbreast$famid==5, ]
mbped <- mbsub[, c("famid", "id", "fatherid", "motherid", "sex")]
## renaming column names
colnames(mbped) <- c("family", "id", "father", "mother", "sex")

## Get ids of individuals from the same generation than "4"
getGenerationMatched(mbped, id="4")

## Get the ids of gemale individuals from the same generation than
## "4" and "22"
getGenerationSexMatched(mbped, id=c("4", "22"))

## Generate a FAData
fad <- FAData(mbped)

## Plot the pedigree so we can evaluate the results of the function
plotPed(fad, family="4")

## We're using getExternalMatched on the sex of the individuals
## in the pedigree, thus the results are identical to the
## getSexMatched function.
## Extracting the sex using $sex returns a named vector just
## as we need for getExternalMatched.
head(fad$sex)
getExternalMatched(fad, id="4", match.using=fad$sex)

getSexMatched(fad, id="4")
```

`kinshipPairs` *Extract pairs of individuals matching certain kinship criteria*

Description

The `kinshipPairs` function allows to extract pairs of individuals matching a user-defined kinship *condition* (e.g. individuals with a kinship larger than 0.0625). Such sets of paired individuals (along with paired unrelated values) would enable a *familial resemblance* analysis on quantitative traits (Ziegler 2010) (see examples below for details).

By default, `kinshipPairs` returns all pairs of individuals for which the `condition` on the kinship matrix matches (e.g. all pairs of individuals with a kinship coefficient larger than or equal to 0.25). Individuals can thus be reported multiple times (see examples below). Parameter `duplicates` can be used to define a strategy to avoid such duplicated IDs. Supported are:

- `duplicates = "keep"`: the default, return all values.
- `duplicates = "first"`: report only the first pair of individuals for each individual ID.
- `duplicates = "last"`: report only the last pair of individuals for each individual ID.
- `duplicates = "random"`: randomly select one pair of individuals for each individual ID.

For any setting different than `duplicates = "keep"` each individual will only be listed **once** in the resulting matrix.

Usage

```
kinshipPairs(
  x,
  condition = function(x) x >= 0.25,
  duplicates = c("keep", "first", "last", "random"),
  id = NULL,
  family = NULL
)
```

Arguments

`x`	A `FAData` object (or object inheriting from that).
`condition`	A `function` defining how individuals should be selected based on the object's kinship matrix. The default is to select all individuals with a kinship `>= 0.25`. Note that the diagonal of the kinship matrix (i.e. the kinship of individuals with itself) is always skipped, so no additional criteria is needed to avoid self-pairs.
`duplicates`	`character(1)` defining how to deal with duplicated IDs in the result returned by the function. See function description and examples below for more details. Defaults to `duplicates = "keep"` returning all pairs of IDs matching `condition`.
`id`	optional identifiers of subsets of individuals on which the pairs should be defined. Defaults to `id = NULL` hence the full data set is considered.
`family`	optional family identifiers if pairs should only defined for selected families. Defaults to `family = NULL` hence the full data set is considered.

Value

A two column `matrix` with the IDs (colnames/rownames of the kinship matrix or as defined in `x$id`) of the pairs. If `duplicates` is either `"first"`, `"last"` or `"random"` each ID is only returned once (i.e. no ID is reported more than one time).

Author(s)

Johannes Rainer

References

Ziegler A., Koenig I. R. (2010). Familiality, Heristability, and Segregation Analysis. In A Statistical Approach to Genetic Epidemiology: With Access to E-Learning Platform by Friedrich Pahlke, Second Edition. doi:10.1002/9783527633654.ch6.

See Also

PedigreeUtils for other pedigree utility functions.

Examples

```
##########################
##
##  Create a new FAData object
##
## Load the Minnesota Breast Cancer record and subset to the
## first families.
data(minnbreast)
mbsub <- minnbreast[minnbreast$famid %in% 1:20, ]
mbped <- mbsub[, c("famid", "id", "fatherid", "motherid", "sex")]
## Renaming column names
colnames(mbped) <- c("family", "id", "father", "mother", "sex")
## Defining the optional argument age.
Age <- mbsub$endage
names(Age) <- mbsub$id
## Create the object
fad <- FAData(pedigree=mbped, age=Age)

## Getting all pairs of individuals with a kinship coefficient >= 0.25
## keeping all duplicates
rel_pairs <- kinshipPairs(fad)
head(rel_pairs)
## As we see, we have multiple times the individual 1 etc.

## For an actual correlation analysis it would be better to drop duplicates.
## Below we randomly select individual pairs if they occurr multiple times
rel_pairs <- kinshipPairs(fad, duplicates = "random")
head(rel_pairs)

## In addition we extract pairs of individuals that are much less related.
## For this examples we consider all individuals with a kinship
```

```
## coefficient < 0.03125 (second cousin) to be *unrelated*.
unrel_pairs <- kinshipPairs(fad, duplicates = "random",
    condition = function(z) z < 0.03125)
head(unrel_pairs)

## For a familial resemblance analysis we can now calculate the correlation
## coefficient of a quantitative trait between pairs of related individuals
## and compare that with the correlation coefficient calculated on unrelated
## individuals. For our toy example we use the participant's age, since we
## don't have any other quantitative values available.
cor_rel <- cor(age(fad)[rel_pairs[, 1]], age(fad)[rel_pairs[, 2]],
    use = "pairwise.complete.obs")

cor_unrel <- cor(age(fad)[unrel_pairs[, 1]], age(fad)[unrel_pairs[, 2]],
    use = "pairwise.complete.obs")
cor_rel
cor_unrel

## We don't see a clear difference in the correlation, thus, the age (as
## expected) has no familial component.
```

PedigreeAnalysis *Pedigree analysis and familial aggregation methods*

Description

Various functions to perform pedigree analyses and to investigate familial clustering of e.g. cancer cases.

Usage

```
binomialTest(object, trait, traitName, global = FALSE, prob = NULL,
    alternative = c("greater", "less", "two.sided"))

estimateTimeAtRisk(startDate=NULL, startDateFormat="%Y-%m-%d",
                   endDate=NULL, endDateFormat="%Y-%m-%d",
                   incidenceDate=NULL, incidenceDateFormat="%Y-%m-%d",
                   deathDate=NULL, deathDateFormat="%Y-%m-%d",
                   allowNegative=FALSE, affected=NULL,
                   incidenceSubtract=0.5)

factor2matrix(x)

## S4 method for signature 'FAData'
familialIncidenceRate(object, trait=NULL,
                                        timeAtRisk=NULL)

## S4 method for signature 'FAData'
familialIncidenceRateTest(object, trait=NULL,
```

```
                                          nsim=50000, traitName=NULL,
                                          timeAtRisk=NULL,
                                          strata=NULL, ...)

## S4 method for signature 'FAData'
fsir(object, trait=NULL, lambda=NULL, timeInStrata=NULL)

## S4 method for signature 'FAData'
fsirTest(object, trait=NULL, nsim=50000, traitName=NULL,
                            lambda=NULL, timeInStrata=NULL,
                            strata=NULL, ...)

## S4 method for signature 'FAData'
genealogicalIndexTest(object, trait, nsim=50000,
                                       traitName, perFamilyTest=FALSE,
                                       controlSetMethod="getAll",
                                       rm.singletons=TRUE, strata=NULL, ...)

## S4 method for signature 'FAData'
kinshipGroupTest(object, trait, nsim=50000,
                                    traitName, strata=NULL, ...)

## S4 method for signature 'FAData'
kinshipSumTest(object, trait, nsim=50000,
                                  traitName, strata=NULL, ...)

## S4 method for signature 'FAData'
probabilityTest(object, trait, cliques,
                                   nsim=50000, traitName,
                                   ...)

sliceAge(x, slices=c(0, 40, Inf))
```

Arguments

(in alphabetic order)

affected	For estimateTimeAtRisk: optional parameter specifying which of the individuals are affected. This is useful if only endDate is specified, but not the incidenceDate. See method description for further details.
allowNegative	For estimateTimeAtRisk: if FALSE any negative time periods are set to 0.
alternative	For binomialTest: the alternative hypothesis. See binom.test for more details. Defaults to "greater", i.e. tests whether in a family a larger number of affected is present than expected by chance (given a global probability).
cliques	A named numeric or characted vector or factor with the names corresponding to ids of the individuals in the pedigree. The ids will be internally matched and sub-set to the ids available in the pedigree.

controlSetMethod
: For genealogicalIndexTest: the method (i.e. name of the function) that should be used to define the set of (eventually matched) control individuals from which the random samples are taken. Supported functions are getAll, getSexMatched and getExternalMatched. For perFamilyTest=TRUE also getGenerationMatched and getGenerationSexMatched are supported. Note: for getExternalMatched, a numeric, character or factor vector to be used for the matching has to be submitted as additional argument match.using.

deathDate
: For estimateTimeAtRisk: the date of death.

deathDateFormat
: For estimateTimeAtRisk: the format in which the dates are submitted. See as.Date for more information.

endDate
: For estimateTimeAtRisk: the end date, which can be the end date for the study or, if deathDate and incidenceDate are not specified, the earliest time point of: date of incidence, death or end of study.

endDateFormat
: For estimateTimeAtRisk: the format in which the dates are submitted. See as.Date for more information.

global
: For binomialTest: whether the binomial test should be applied to the whole pedigree, or family-wise (default). If global = TRUE the population probability has to be provided with parameter prob.

incidenceDate
: For estimateTimeAtRisk: the date of the incidence for an individual, i.e. the date when the status was changed from un-affected to affected in the to be analyzed trait.

incidenceDateFormat
: For estimateTimeAtRisk: the format in which the dates are submitted. See as.Date for more information.

incidenceSubtract
: For estimateTimeAtRisk: the amount of time (of the time unit of the time at risk) that should be subtracted from the calculated time at risk for affected individuals. See method description below for more details.

lambda
: Numeric vector with the incidence rates per stratum from the population. The length of this vector has to match the number of columns of argument timeInStrata.

nsim
: The number of simulations.

object
: The FAData object.

perFamilyTest
: For genealogicalIndexTest: whether the test should be performed on the whole pedigree (default) or separately within each family. In the latter case the test evaluates the presence of clustered affected individuals within each family.

prob
: For binomialTest: the hypothesized probability of success (being affected) from/for the whole population.

rm.singletons
: For genealogicalIndexTest: whether unconnected individuals in the pedigree (singletons) should be removed from the pedigree prior to the analysis.

slices
: For sliceAge: a numeric vector defining the age-slices. Similar to argument vec for findInterval. Defines the minimum and maximum age for the age slices, i.e. first number corresponds to the lower boundary of the first age slice, the second number to the upper boundary of the first and lower boundary of the second age slice and so on.

startDate	For estimateTimeAtRisk: the date of the start of the study. Can also be the birth date.
startDateFormat	For estimateTimeAtRisk: the format in which the dates are submitted. See as.Date for more information.
strata	For genealogicalIndexTest, kinshipGroupTest and kinshipSumTest: a numeric, character or factor characterizing each individual in the pedigree. The length of this vector and the ordering has to match the pedigree. This vector allows to perform stratified random sampling. See details for more information.
timeAtRisk	A numeric vector specifying the *time at risk* for each individual. The definition for this variable is taken from Kerber (1995). See description of the method below for more information. timeAtRisk has to have the same number of elements than there are individuals in the pedigree and it is assumed that the ordering of the vector matches the order of the individuals in the pedigree.
timeInStrata	For fsir and fsirTest: a numeric matrix specifying the time at risk for each individual in each strata. Columns represent the strata, rows the individuals, each cell the time at risk for the individual in the respective strata.
trait	A named numeric vector (values 0, 1 and NA) or logical vector (values FALSE, TRUE and NA) with the names matching the ids of the individuals in the pedigree. The method internally matches and re-orders the trait vector to match the ordering of the ids in the pedigree. If trait is not specified, the trait information stored within the FAData object is used.
traitName	The name of the trait (optional).
x	For sliceAge: a numeric vector representing the age of individuals. For factor2matrix: a factor that should be converted into a matrix.
...	For genealogicalIndexTest: additional arguments passed to the choosen controlSetMethod function (e.g. match.using for getExternalMatched). For familialIncidenceRateTest: use lowMem=TRUE for very large pedigrees. This will use a faster and less memory demanding p-value estimation.

Details

Stratified sampling: some of the familial aggregation methods allow to use stratified sampling for the Monte Carlo simulations. In stratified sampling, the same number of random samples will be selected within each class/stratum then there are among the affected. As example, if 5 female and 2 male individuals are affected in the analysed trait and sex stratified sampling is performed, in each permuatation the same number of random samples in each group (i.e. 5 females and 2 males) are selected.

A note on singletons: for all per-individual measures, unconnected individuals within the pedigree are automatically excluded from the calculations as no kinship based statistic can be estimated for them since they do, by definition, not share kinship with any other individual in the pedigree.

Value

Refer to the method and function description above for detailed information on the returned result object.

Familial aggregation methods

binomialTest Evaluate whether the number of affected in a trait are higher than expected by chance using a simple binomial test. In contrast to most other methods presented here, this does not use the kinship between affected individuals, but simply performs a binomial test for each family considering the numbers of affected within the family, the size of the family and the global probability of being affected. The latter is by default calculated on the data set (ratio between the total number of affected in the pedigree and the total number of phenotyped individuals), can however also be specified with the `prob` argument.

The test is performed using the `binom.test`.

The function returns a `FABinTestResults` object.

familialIncidenceRate Calculate the familial incidence rate (FIR, or FR) as defined in [Kerber 1995], formula (3). The FIR is an estimate for the risk per gene-time for each individual for a certain disease (trait) given the disease experience in the cohort. The measure considers the kinship of each individual with any affected individual in the pedigree and the time at risk for each individual.

Internally, the function first excludes individuals from the test which have a missing value (`NA`) either in the argument `trait` or in the argument `timeAtRisk`. Next, the thus reduced pedigree, is further cleaned by removing all resulting singletons (i.e. individuals that do not share kinship with any other individual in the above reduced data set).

The method returns a vector with the FIR value for each individual. Individuals that were excluded from the test as described above habe an FIR of `NA`.

familialIncidenceRateTest Calculates the familial incidence rate for each individual and in addition assesses the significance of these based on Monte Carlo simulations. See `FAIncidenceRateResults` for more details.

The method returns a `FAIncidenceRateResults` object.

fsir Calculate the familial standardized incidence rate (FSIR) as defined in [Kerber, 1995], formula (4). The FSIR weights the disease status of relatives based on their degree of relatedness with the proband [Kerber, 1995]. Formally, the FSIR is defined as the standardized incidence ratio (SIR) or standardized morality ratio in epidemiology, i.e. as the ratio between the observed and expected number of cases, only that both are in addition also weighted by the degree of relatedness (i.e. kinship value) between individuals in the pedigree.

Similar to `familialIncidenceRate`, the function excludes individuals with missing values in any of the arguments `trait`, `timeInStrata` (and optionally `strata`) and all individuals that do not share any kinship with any other individual in the pedigree after removing the above individuals.

The method returns a vector with the FSIR value for each individual. Individuals excluded as above describe have a FSIR value of `NA`.

fsirTest Calculates the familial standardized incidence rate (FSIR) for each individual and in addition assesses the significance of these based on Monte Carlo simulations. See `FAStdIncidenceRateResults` for more details.

The method returns a `FAStdIncidenceRateResults` object.

genealogicalIndexTest Performs the *genealogical index* analysis from [Hill 1980] (also known as the *genealogical index of familiality* or `genetic index of familiality`) to identify familial clustering of traits (e.g. cancers etc).

This test calculates the mean kinship among affected individuals in a pedigree along with mean kinships of equal sized random control sets drawn form the pedigree. The distribution of average kinship values among these random sets is used to estimate the probability that the observed mean kinship between the affected individuals is due to chance. The `controlSetMethod` argument allows to specify the method to define sets of matched control individuals in a pedigree or family.

Note that by default singletons (i.e. unconnected individuals in the pedigree) are removed from the pedigree prior the analysis. Set `rm.singletons=FALSE` if you do not want them to be removed.

The method can also be performed separately for each family within the larger pedigree (`perFamilyTest=TRUE` to evaluate the presence of clustered affected within each family). In this case it is also possible to use `controlSetMethod="getGenerationMatched"` or `controlSetMethod="getGenerat` which allows to draw random control samples from the same generation(s) than the affected are.

Stratified random sampling can be performed with the `strata` argument. See details for more information.

The function returns a `FAGenIndexResults` object.

kinshipGroupTest Performs a familial aggregation test on a subset of a family. The idea behind this test is to narrow down the set of controls for each affected individual by considering only individuals that are as closely related as the most distant affected individual. This strategy incorporates more the family structure of the cases and is meant to be an alternative to the `kinshipSumTest` method.

Initially, for an affected individual *i* a group *C(i)* is created that contains all individuals that share kinship as far as the most distantly related affected individual. This cluster can be interpreted as a circle that is centered at individual *i* with radius equal to the most distantly related case. Therefore, the cluster defines a narrowed, individual-specific set of individuals in which the phenotype is assumed to have been passed on. Groups consisting of the same set of affected individuals are reduced to a single group (i.e. the group with the smallest total number of individuals).

Based on this definition of groups *C(i)*, we compute two statistics by performing Monte Carlo simulations (which optionally allow to perform stratified random sampling). During each simulation step affected cases are randomly sampled from the population.

1. The ratio test counts per group *C(i)* the number of times we observe a higher number of affected individuals in the simulation than in the observed case. Dividing this number by the number of simulation steps yields immediately the p-value that describes the event to observe by chance a higher number of affected individuals than in the given case.

2. The kinship test addresses the degree of relatedness within the simulated set by a counting method where we count the number of times in a simulation step there is a pair of affected individuals that are more closely related than in the observed group *C(i)*. In case the closest degree of relatedness is equal in both the simulation step and the observed case, we look at the number of pairs found in both and count it if this number is higher in the simulation step. Again, dividing this count by the number of simulation steps readily yields a p-value.

See also the method `runSimulation` for `FAKinGroupResults`.

The function returns a `FAKinGroupResults` object.

kinshipSumTest Performs a test for familial aggregation based on the sum of kinship values between affected cases. This test highlights individuals that exhibit a higher than chance relationship to other affected individuals, therefore highlighting individuals within families aggre-

gating the phenotype. To achieve this, for each affected individual the sum of kinship values to all other affected cases is computed. In a Monte Carlo simulation this is repeated with the same number of cases (and optionally stratified with the `strata` argument), and the resulting background distribution is used to compute p-values for the kinship sums obtained from the observed cases. See also the method `runSimulation` for `FAKinSumResults`.

The function returns a `FAKinSumResults` object.

probabilityTest DEPRECATED: this test will be removed in Bioconductor version 3.8 due to problems and incompatibilities of the gap package on MS Windows systems.

This is only a convience method that calls the gap package's method `pfc.sim` to compute probabilities of familial clustering of phenotypes [Yu and Zelterman (2002)]. One drawback of that method is that it is limited to families with at most 22 individuals. Thus, pedigrees need to be split with specialized software such as Jenti [Falchi and Fuchsberger ea. (2008)], which within large families define cliques that can then be used as input to this algorithm.

See also method `runSimulation` for `FAProbResults`.

The function returns a `FAProbResults` object.

Utility functions

factor2matrix Converts a factor into a matrix with columns corresponding to the levels and values (cell row i, column j) being either 0 or 1 depending on whether the ith factor was of the level j. See examples below for in or `FAStdIncidenceRateResults`.

estimateTimeAtRisk Function to calculate the time at risk based on the start date of the study or the birth date of an individual (`startDate`) and the study's end date (`endDate`), the date of an incidence (e.g. date of diagnosis of a cancer `incidenceDate`) or the death of the individual (`deathDate`). The time at risk for each individual is calculated as the minimal time period between `startDate` and any of `endDate`, `incidenceDate` or `deathDate`. Thus it is also possible to provide just the `endDate` along with the `startDate`, in which case the `endDate` should be the earliest time point of: end date of the study, incidence date or date of death.

For affected individuals (those for which either an incidence date is provided or the value in the optional argument `affected` is `TRUE` or bigger than 0), by default half of the time unit is subtracted. For example, a individual that has an incidence after 2 days is 1.5 days at risk. The proportion of the time unit to subtract can be specified with the argument `incidenceSubtract`.

The function returns a numeric vector with the time at risk in days.

sliceAge Generates a matrix with columns corresponding to age slices/strata defined by argument `slices` and rows to individuals. Each cell in a row represents the time spent by the individual in the age slice/strata. See example below.

Author(s)

Johannes Rainer, Daniel Taliun, Christian Weichenberger.

References

Rainer J, Talliun D, D'Elia Y, Domingues FS and Weichenberger CX (2016) FamAgg: an R package to evaluate familial aggregation of traits in large pedigrees. *Bioinformatics*.

Hill, J.R. (1980) A survey of cancer sites by kinship in the Utah Mormon population. In Cairns J, Lyon JL, Skolnick M (eds): *Cancer Incidence in Defined Populations. Banbury Report 4.* Cold Spring Harbor, NY: Cold Spring Harbor Laboratory Press, pp 299–318.

Kerber, R.A. (1995) Method for calculating risk associated with family history of a disease. *Genet Epidemiol*, pp 291–301.

Yu, C. and Zelterman, D. (2002) Statistical inference for familial disease clusters. *Biometrics*, pp 481–491

Falchi, M. and Fuchsberger, C. (2008) Jenti: an efficient tool for mining complex inbred genealogies. *Bioinformatics*, pp 724–726

See Also

pedigree, FAData, FAProbResults, FAKinGroupResults, FAKinSumResults, FAIncidenceRateResults

Examples

```
##########################
##
##  Defining a small pedigree
##
## load the Minnesota Breast Cancer record and subset to the
## first families.
data(minnbreast)
mbsub <- minnbreast[minnbreast$famid==4 | minnbreast$famid==5 |
                    minnbreast$famid==14 | minnbreast$famid==8, ]
mbped <- mbsub[, c("famid", "id", "fatherid", "motherid", "sex")]
## renaming column names
colnames(mbped) <- c("family", "id", "father", "mother", "sex")
## create the FAData object
fad <- FAData(pedigree=mbped)

## We specify the cancer trait.
tcancer <- mbsub$cancer
names(tcancer) <- mbsub$id

##########################
##
## Familial Incidence Rate
##
## Calculate the FR for each individual given the affected status of
## each individual in trait cancer and the time at risk for each
## participant. We use column "endage" in the minnbreast data.frame
## that specifies the age at the last follow-up or incident cancer as a
##rather impresice estimate for time at risk.
fr <- familialIncidenceRate(fad, trait=tcancer, timeAtRisk=mbsub$endage)

## Plot the distribution of familial rates
plot(density(fr, na.rm=TRUE))

## Perform in addition Monte Carlo simulations to assess the significance
## for the familial incidence rates.
frRes <- familialIncidenceRateTest(fad, trait=tcancer,
                                   timeAtRisk=mbsub$endage,
                                   nsim=500)
head(result(frRes))
```

```
##########################
##
## Familial Standardized Incidence Rate:
## Please see examples of FAStdIncidenceRateResults.



##########################
##
##  Perform familial aggregation analyses using the genealogical index
##
gi <- genealogicalIndexTest(fad, trait=tcancer, traitName="cancer",
                            nsim=500)
result(gi)
## A significant clustering of cancer cases was identified in the
## analyzed pedigree.

## Plotting the observed mean kinship and the distribution of mean kinship
## from the random sampling.
plotRes(gi)


##########################
##
##  Perform familial aggregation analysis using the kinship sum test
##
kcr <- kinshipSumTest(fad, trait=tcancer, traitName="cancer",
                      nsim=500)
kcr
head(result(kcr))


##########################
##
##  Perform familial aggregation analysis using the kinship group test,
##  stratifying by sex
##
kr <- kinshipGroupTest(fad, trait=tcancer, traitName="cancer",
                       nsim=500, strata=fad$sex)
kr
head(result(kr))



##########################
##
##  Estimate the time at risk given
##
## Define some birth dates and incidence dates and end date of study
bdates <- c("2012-04-17", "2014-05-29", "1999-12-31", "2002-10-10")
idates <- c(NA, NA, "2007-07-13", "2013-12-23")
```

```
edates <- rep("2015-09-15", 4)

## Estimate the time at risk. The time period is returned in days.
riskDays <- estimateTimeAtRisk(startDate=bdates, incidenceDate=idates, endDate=edates)
riskDays

##########################
##
##  Define the time spent in an age stratum given the indivduals'
##  age at incidence or end of study.
head(mbsub$endage)
## We "slice" the age in specified intervals/slices
stratAge <- sliceAge(mbsub$endage, slices=c(0, 40, 60, Inf))
head(stratAge)

## The first column lists the number of years spent in the first age
## stratum (0 < age <= 40) and the second in the second stratum
## (40 < age <= Inf)

## We could also stratify the disk days from above in per year strata.
sliceAge(riskDays/365, slices=c(0, 2.5, 5, 10, 20))


##########################
##
##  Simple example for factor2matrix: generate a matrix for factor $sex
head(factor2matrix(fad$sex))
```

PedigreeUtils *Basic pedigree utilities*

Description

Utility functions to access, modify or subset pedigrees. Most of these functions can be applied to simple `data.frame` in pedigree format or `pedigree` or `pedigreeList` objects defined in the `kinship2` package.

Usage

```
## S4 method for signature 'missing'
cliques(object, ...)

connectedSubgraph(graph, nodes, mode="all", all.nodes=TRUE, ifnotfound)

## S4 method for signature 'FAData'
```

```
countGenerations(object, id=NULL, direction="down", ...)

## S4 method for signature 'FAData'
estimateGenerations(object, family=NULL, ...)

## S4 method for signature 'FAData'
findFounders(object, family=NULL, id = NULL, ...)

## S4 method for signature 'FAData'
generationsFrom(object, id=NULL, ...)

## S4 method for signature 'FAData'
getAncestors(object, id=NULL, max.generations=3, ...)

## S4 method for signature 'FAData'
getChildren(object, id=NULL, max.generations=16, ...)

## S4 method for signature 'FAData'
getCommonAncestor(object, id, method="min.dist")

## S4 method for signature 'FAData'
getFounders(object, ...)

## S4 method for signature 'FAData'
getMissingMate(object, id=NULL, ...)

## S4 method for signature 'FAData'
getSiblings(object, id=NULL, ...)

## S4 method for signature 'FAData'
getSingletons(object, ...)

ped2graph(ped)

## S4 method for signature 'FAData'
removeSingletons(object, ...)

## S4 method for signature 'data.frame'
removeSingletons(object, ...)

subPedigree(ped, id=NULL, all=TRUE)

## S4 method for signature 'FAData'
shareKinship(object, id, rmKinship=0)
```

Arguments

(in alphabetic order)

all	For subPedigree: if all individuals have to be present in the sub-pedigree.
all.nodes	For connectedSubgraph: if all nodes have to be present in the resulting graph, or only those that are connected with each other.
direction	For countGenerations: whether the number of ancestor ("up") generations or offspring ("down") generation should be counted.
family	A character or numeric representing the family id. For doFindFounders: the id of the family in the pedigree for which the founders should be identified. Uses the first family in the pedigree if not specified. For estimateGenerations: optional id of the family if generation numbers should only be calculated for one family. Otherwiseif, the generations are estimated for all families (separately) in the object.
graph	An igraph graph object.
id	A character or numeric vector length 1 or longer specifying the id(s) of the individual(s). For generationsFrom and findFounders only a single id should be submitted.
ifnotfound	For connectedSubgraph: if not defined, the function throws an error if no subgraph can be specified. If defined, its value is returned if no subgraph was found.
max.generations	For getAncestors and getChildren: the maximal number of ancestor or offspring generations that should be returned.
method	For getCommonAncestor: the method by which the closest common ancestor sould be identified. Either "min.dist" (ancestor with the minimal distance to any of the individuals) or "smallest.mean.dist" (ancestor with the smallest mean distance to any of the individuals).
mode	For connectedSubgraph: either "all", "in", "out" specifying how distances and paths between individual nodes should be determined. See help for function shortest_paths in package igraph for more details.
nodes	For connectedSubgraph: A character vector of node (vertex) names for which the subgraph should be defined.
object	For cliques: passed to the cliques function from the igraph package. For all other methods: either a FAData object (or any object inheriting from it), a data.frame, pedigree or pedigreeList objects (the latter being defined in the kinship2 package).
ped	Either a data.frame or a pedigree object specifying the pedigree. If a data.frame is submitted, the columns id, family, father, mother and sex are required.
rmKinship	For shareKinship: threshold for inclusion in being reported. Pairs of individuals with a kinship less or equal than this value will be omitted. This can be used to remove very distant relatives.
...	For cliques: additional arguments passed to the cliques function from the igraph package.

Value

Refer to the method and function description above for detailed information on the returned result object.

Basic pedigree utilities

countGenerations Count the generations up- or down the pedigree for the specified individual(s), i.e. determine the number of ancestor or offspring generations defined in the pedigree for the specified individual(s). Returns a named numeric vector, names corresponding to the individual's id, with the number of generations for each specified individual.

findFounders Identifies the founder couple with the largest number of offspring generations in the pedigree. The provided pedigree object/data.frame can contain pedigrees of multiple families, thus, to identify the founder pair for a family its ID can be provided with the `family` parameter. Alternatively, the ID of an individual can be specified, in which case the founder pair of the (full) pedigree of the specified individual is identified. If two or more couples have the same, largest number of offspring generations, the first couple is selected. Returns a character vector of length 2 with the ids of the founder individuals.

getFounders Returns the ids of all founders in the pedigree. A founder is an individual from which neither father nor mother is known in the pedigree.

getSingletons Returns the ids of all singletons, i.e. individuals in the pedigree that are not connected to any other individual (have no parents in the pedigree and no children).

getAncestors Identify and return the ids of ancestor generations (up to `max.generations`) for the specified individual(s).

getChildren Identify and return the ids of offspring generations (up to `max.generations`) for the specified individual(s).

getCommonAncestor Finds the closest common ancestor between specified individuals (2 or more ids are required). Returns a character vector with the ids of the ancestors or `NA` if no common ancestor was found.

getMissingMate The function evaluates if in the sub-pedigree defined by the specified ids one or more mates (spouse) are missing and if so it returns their ids.

getSiblings Get siblings for the specified id(s). Returns their ids as character, or numeric vector.

removeSingletons Removes all unconnected individuals (i.e. singletons) from the pedigree. Returns a `data.frame` with the pedigree cleaned from all singletons. Note that, due to internal sanitizing, columns `"father"` and `"mother"` in the resulting `data.frame` have a `NA` for individuals for which the father or mother is not known in the pedigree.

subPedigree Finds the smallest pedigree containing all specified individuals. Depending on the input, a `data.frame`, `pedigree` or `pedigreeList`.

Advanced pedigree methods

estimateGenerations Estimates generation levels/numbers for each, or only one, family in the object. Generation numbers are always relative to the founder couple (defined by `findFounders`). Returns (always) a named list of generation numbers. The names of the list represent the family id, the names of the numeric vector of generations the id of the individuals in the family.

generationsFrom Determine generations starting from the specified individual. Siblings including their mates and all other in the same generation () are assigned generation 0, ancestor generations (all their parents etc) negative generation numbers, decreasing with ancestor level and their offspring positive numbers, increasing with each generation. Generations are only estimated within the family of the individual, also, if the pedigree consists of un-connected sub-pedigree, generation numbers will only be calculated for the sub-pedigree containing the

specified individual. The function returns a named numeric vector of generation numbers, the names corresponding to the ids of the individuals in the specified individual's family. Not connected individuals in the family get a NA generation number.

shareKinship Finds all related individuals (individuals sharing kinship > rmKinship with the individual) for the specified individual(s) in the pedigree and returns their ids as a character vector.

Graph theory related functions

cliques Wrapper method passing all arguments to the cliques function from the igraph package.

connectedSubgraph Finds the (eventually smallest) connected subgraph of all specified nodes. Returns an igraph object representing the subgraph of the specified nodes.

ped2graph Transforms the pedigree into a (directed) graph with the direction of the edges being always from parent to child. An igraph object.

Author(s)

Johannes Rainer.

See Also

pedigree, FAData, FAProbResults, FAKinGroupResults, FAKinSumResults, PedigreeAnalysis kinshipPairs

Examples

```
##########################
##
##  Defining a small pedigree
##
## load the Minnesota Breast Cancer record and subset to the
## first families.
data(minnbreast)
mbsub <- minnbreast[minnbreast$famid==4 | minnbreast$famid==5, ]
mbped <- mbsub[, c("famid", "id", "fatherid", "motherid", "sex")]
## renaming column names
colnames(mbped) <- c("family", "id", "father", "mother", "sex")

## plot the pedigree for family 4 to get an overview.
switchPlotfun(method="ks2paint")
fam4 <- mbped[mbped$family==4, ]
doPlotPed(individual=fam4$id, father=fam4$father, mother=fam4$mother,
          gender=fam4$sex, device="plot")

## find the closest common ancestor between individuals 23, 3 and 8
getCommonAncestor(fam4, id=c(23, 3, 8))

## create the smallest sub-pedigree for individuals 21, 22 and 25
subPedigree(fam4, id=c(21, 22, 25))
## plot that
fam4sub <- subPedigree(fam4, id=c(21, 22, 25))
```

```
doPlotPed(individual=fam4sub$id, father=fam4sub$father, mother=fam4sub$mother,
          gender=fam4sub$sex, device="plot")

########################
##
##  Basic pedigree utils
##
## Note: the same methods can be applied to a data.frame representing
## a pedigree, or a FAData, pedigree or pedigreeList object.

## Find the founder couple for family 4
findFounders(fam4, family=4)

## Alternatively, find the founders for the pedigree in which ibdividual 20 is a
## member
findFounders(fam4, id = 20)

## Return all founders in the pedigree.
getFounders(fam4)

## Get all founders without children (i.e. singletons).
getSingletons(fam4)

## Clean the pedigree from all singletons
fam4noS <- removeSingletons(fam4)
nrow(fam4)
nrow(fam4noS)

## Count the offspring generations for individual "4"
countGenerations(fam4, id="4")

## Get the ids of all ancestors for that individual
getAncestors(fam4, id="4")

## Get the ids of the children of this individual
getChildren(fam4, id="4", max.generations=1)

## Get the ids of the complete offspring for this individuals
getChildren(fam4, id="4")

## Create a FAData object from the pedigree data.frame
fad <- FAData(fam4)
## get the list of all ids sharing kinship with individuals
## 5 and 9
shareKinship(fad, id=c("5", "9"))

## Count the numbers of generations of ancestors for individual 12
countGenerations(fad, id="12", direction="up")

## Count the numbers of offspring generations for individuals 2 and 29
countGenerations(fad, id=c("2", "29"))

## Get all brothers/sisters for individual 9
```

```
getSiblings(fad, id="9")

## Determine generation levels starting from individual "9"
generationsFrom(fad, id="9")

## Estimate generations relative to the founder couple for each
## family in the submitted object, a data.frame in the example below
estimateGenerations(mbped)


#########################
##
##  Graph utilities
##
## Convert the pedigree into a graph
pgraph <- ped2graph(fam4)
plot(pgraph)

## Make a subgraph containing nodes 10, 22, 12 and 14
sgraph <- connectedSubgraph(pgraph, c("10", "22", "12", "14"))
plot(sgraph)
```

---

plotPed *Pedigree plotting functions*

Description

Plot a pedigree for a family or an individual.

Usage

```
doPlotPed(family=NULL, individual=NULL, father=NULL, mother=NULL, gender=NULL,
        affected=NULL, is.deceased=NULL, is.sab.or.top=NULL, is.proband=NULL,
        is.adopted=NULL, are.twins=NULL, are.consanguineous=NULL,
        text.inside.symbol=NULL, text.beside.symbol=NULL,
        text1.below.symbol=NULL, text2.below.symbol=NULL,
        text3.below.symbol=NULL, text4.below.symbol=NULL,
        filename=NULL, device="plot", res=600, ...)

switchPlotfun(method, check=TRUE)
```

Arguments

family	(Optional) character or numeric vector specifying the family id.
individual	(Required) character or numeric vector with the ids of the individuals.
father	(Required) character or numeric vector with the id of the father for each individual.

`mother` (Required) character or numeric vector with the id of the mother for each individual.

`gender` (Required) character, factor or numeric vector specifying the gender, with `1` or any string starting with `"m"` coding for male and `2` or any string starting with `"f"` for male; `NA` codes for unknown.

`affected` (Optional) numeric or logical vector specifying if the individual is affected, `0` or `FALSE` for not affected, `1` or `TRUE` for affected, `NA` for not phenotyped.

`is.deceased` (Optional) numeric of logical vector specifying whether the individual is deceased.

`is.sab.or.top` (Optional) numeric or logical vector specifying if the individual is the result of a spontaneous abortion or termination of pregnancy.

`is.proband` (Optional) numeric or logical vector specifying whether the individual is declared as proband (i.e. the first affected family member coming to medical attention).

`is.adopted` (Optional) numeric or logical vector specifying if the individual has been adopted.

`are.twins` (Optional) character vector spefifying twins in the family. Individuals sharing the same string are recognized as twins. The string has to start either with `"m_"` or `"d_"` for monozygotic or dizygotic twins, followed by the unique identifier for the twins.

`are.consanguineous`
(Optional) character vector specifying cosanguineous couples.

`text.inside.symbol`
(Optional) character vector with text to place inside symbols.

`text.beside.symbol`
(Optional) character vector with text to place beside symbols.

`text1.below.symbol`
(Optional) character vector with text to place below symbols.

`text2.below.symbol`
(Optional) character vector with text to place below symbols.

`text3.below.symbol`
(Optional) character vector with text to place below symbols.

`text4.below.symbol`
(Optional) character vector with text to place below symbols.

`filename` (Optional) file name for the plot. If not specified the result is plotted to a temporary file.

`device` The format of the output file. Can be `"ps"`, `"pdf"`, `"svg"`, `"png"` or `"txt"` if `HaploPainter` is used to create the plot, or `"pdf"`, `"png"` or `"plot"` if `kinship2` is used for plotting. If the `HaploPainter` backend is not installed, it is still possible to produce `HaploPainter` input files using `devive = "txt"` for later invocation of `HaploPainter`: this is achieved by calling `switchPlotfun("haplopaint", check = FALSE)`, which will not check for the presence of a `HaploPainter` executable. Note: if `"plot"` is specified the plot is displayed instead of exported to a file.

`res` (Optional) the resolution of the image if saved to a bitmap device.

method	The method which should be used for plotting, either "ks2paint" (uses kinship2 for plotting) or "haplopaint" (uses HaploPaint). If not specified, the functions switches between the methods.
check	A logical indicating whether the plotting backends (currently applied only to HaploPaint) is installed and working. Defaults to TRUE, such that it is guaranteed that a call to doPlotPed will at least technically succeed. The test is omitted by setting this argument to FALSE.
...	For plotPed: additional arguments submitted to the plotting function doPlotPed.

Details

All arguments passed to the doPlotPed function have to have the same length (with the exception of arguments filename, device and res) and their order has to match the order of the individuals.

The arguments of the doPlotPed function represent the input parameters supported by HaploPainter; for more information about HaploPainter refer to http://haplopainter.sourceforge.net/.

By default, doPlotPed uses the kinship2 package for plotting, but can also be configured by the switchPlotFun to use HaploPainter instead. HaploPainter is a perl script/tool for pedigree plotting bundled in the package that requires however some dependencies that might not be present on every system. Thus, the package checks on startup whether all requirements for HaploPainter are available. This check can be skipped by using check=FALSE when calling switchPlotFun. While using this argument is generally not recommended, it is of use when only writing HaploPainter input files, which does not make use of the HaploPainter plotting backend.

If HaploPainter is used, the plot can only be exported to a pdf or png device, while, if kinship2 is used, the plot can also be directly plotted and displayed (if device="plot" is specified).

HaploPainter plotting supports also device = "txt" in which case the pedigree data are exported (in the HaploPainter file format) as a tabulator delimited file - no plot is created, the name of the file is returned. This can even be done without a HaploPainter executable by calling switchPlotFun("haplopaint", check=FALSE).

Also, the arguments of this function match the arguments for HaploPainter and not all settings can be directly matched to settings in kinship2 plotting. The list below lists all arguments specific to HaploPainter and how, if at all, they are displayed or used in kinship2 plotting:

is.sab.or.top Not supported yet.

is.proband The id of individuals which are marked as probands are highlighted in red.

is.adopted Not supported yet.

are.twins Not supported yet.

are.consanguineous Not supported yet.

text2.below.symbol The text will be plotted on the top left corner of the symbol of the respective individual.

text3.below.symbol The text will be plotted on the top right corner of the symbol of the respective individual.

Value

switchMethod A character string representing the plotting function to be used.

doPlotPed The name of the file to which the plot was exported.

Author(s)

Johannes Rainer.

See Also

plot.pedigree, plotPed,FAData-method,

Examples

```
## load the Minnesota Breast Cancer record and subset to the
## first families.
data(minnbreast)
family <- minnbreast[minnbreast$famid==4, ]

## draw a pedigree and export it to a pdf file; the file name is
## returned by the function.
doPlotPed(family=family$famid, individual=family$id, father=family$fatherid,
          mother=family$motherid, gender=family$sex, device="pdf")

## switch to the plotting functions of the kinship2 package
switchPlotfun("ks2paint")

## plot the same pedigree, but display it
doPlotPed(family=family$famid, individual=family$id, father=family$fatherid,
          mother=family$motherid, gender=family$sex, device="plot")

## Finally, generate an input file that can be used for interactive or
## scripted HaploPainter pedigree drawing.
switchPlotfun("haplopaint", check=FALSE)
doPlotPed(family=family$famid, individual=family$id, father=family$fatherid,
          mother=family$motherid, gender=family$sex, device="txt",
          filename="haplopainter.tsv")
```

Index

∗ **classes**
FABinTestResults-class, 2
FAData-class, 5
FAGenIndexResults-class, 14
FAIncidenceRateResults-class, 19
FAKinGroupResults-class, 24
FAKinSumResults-class, 29
FAProbResults-class, 33
FAStdIncidenceRateResults-class, 37
getAll, 43
PedigreeAnalysis, 48
PedigreeUtils, 57
∗ **plot**
plotPed, 63
[,FABinTestResults,ANY,ANY,ANY-method *(*FABinTestResults-class*)*, 2
[,FABinTestResults,ANY-method *(*FABinTestResults-class*)*, 2
[,FAData,ANY,ANY,ANY-method *(*FAData-class*)*, 5
[,FAData,ANY-method *(*FAData-class*)*, 5
[,FAGenIndexResults,ANY,ANY,ANY-method *(*FAGenIndexResults-class*)*, 14
[,FAGenIndexResults,ANY-method *(*FAGenIndexResults-class*)*, 14
[,FAIncidenceRateResults,ANY,ANY,ANY-method *(*FAIncidenceRateResults-class*)*, 19
[,FAIncidenceRateResults,ANY-method *(*FAIncidenceRateResults-class*)*, 19
[,FAKinGroupResults,ANY,ANY,ANY-method *(*FAKinGroupResults-class*)*, 24
[,FAKinGroupResults,ANY-method *(*FAKinGroupResults-class*)*, 24
[,FAKinSumResults,ANY,ANY,ANY-method *(*FAKinSumResults-class*)*, 29
[,FAKinSumResults,ANY-method *(*FAKinSumResults-class*)*, 29
[,FAProbResults,ANY,ANY,ANY-method *(*FAProbResults-class*)*, 33
[,FAProbResults,ANY-method *(*FAProbResults-class*)*, 33
[,FAStdIncidenceRateResults,ANY,ANY,ANY-method *(*FAStdIncidenceRateResults-class*)*, 37
[,FAStdIncidenceRateResults,ANY-method *(*FAStdIncidenceRateResults-class*)*, 37
$ *(*FAData-class*)*, 5
$,FAData-method *(*FAData-class*)*, 5
$,FAIncidenceRateResults-method *(*FAIncidenceRateResults-class*)*, 19
$,FAStdIncidenceRateResults-method *(*FAStdIncidenceRateResults-class*)*, 37

affectedIndividuals *(*FAData-class*)*, 5
affectedIndividuals,FAData-method *(*FAData-class*)*, 5
affectedKinshipGroups *(*FAKinGroupResults-class*)*, 24
affectedKinshipGroups,FAKinGroupResults-method *(*FAKinGroupResults-class*)*, 24
age *(*FAData-class*)*, 5
age,FAData-method *(*FAData-class*)*, 5
age<- *(*FAData-class*)*, 5
age<-,FAData-method *(*FAData-class*)*, 5
as.Date, *50, 51*

binom.test, *3*, *49*, *52*
binomialTest, *2, 3*
binomialTest *(*PedigreeAnalysis*)*, 48
buildPed, *28*, *37*
buildPed *(*FAData-class*)*, 5
buildPed,FAData-method *(*FAData-class*)*, 5

buildPed,FAKinGroupResults-method
(FAKinGroupResults-class), 24
buildPed,FAProbResults-method
(FAProbResults-class), 33

cliqueAndTrait (FAProbResults-class), 33
cliqueAndTrait,FAProbResults-method
(FAProbResults-class), 33
cliques (FAProbResults-class), 33
cliques,FAProbResults-method
(FAProbResults-class), 33
cliques,missing-method (PedigreeUtils),
57
cliques<- (FAProbResults-class), 33
cliques<-,FAProbResults-method
(FAProbResults-class), 33
connectedSubgraph (PedigreeUtils), 57
countGenerations (PedigreeUtils), 57
countGenerations,data.frame-method
(PedigreeUtils), 57
countGenerations,FAData-method
(PedigreeUtils), 57
countGenerations,pedigree-method
(PedigreeUtils), 57
countGenerations,pedigreeList-method
(PedigreeUtils), 57

doPlotPed, *8*, *11*
doPlotPed (plotPed), 63

estimateGenerations (PedigreeUtils), 57
estimateGenerations,data.frame-method
(PedigreeUtils), 57
estimateGenerations,FAData-method
(PedigreeUtils), 57
estimateGenerations,pedigree-method
(PedigreeUtils), 57
estimateGenerations,pedigreeList-method
(PedigreeUtils), 57
estimateTimeAtRisk, *21*, *23*, *41*
estimateTimeAtRisk (PedigreeAnalysis),
48
export (FAData-class), 5
export,FAData-method (FAData-class), 5

FABinTestResults, *52*
FABinTestResults
(FABinTestResults-class), 2
FABinTestResults-class, 2

factor2matrix, *39*
factor2matrix (PedigreeAnalysis), 48
FAData, *3*, *4*, *16*, *18*, *21–23*, *26–28*, *31*, *32*, *35*,
37, *39–41*, *45*, *55*, *59*, *61*
FAData (FAData-class), 5
FAData-class, 5
FAGenIndexResults, *11*, *45*, *53*
FAGenIndexResults
(FAGenIndexResults-class), 14
FAGenIndexResults-class, 14
FAIncidenceRateResults, *39*, *52*, *55*
FAIncidenceRateResults
(FAIncidenceRateResults-class),
19
FAIncidenceRateResults-class, 19
FAKinGroupResults, *11*, *45*, *53*, *55*, *61*
FAKinGroupResults
(FAKinGroupResults-class), 24
FAKinGroupResults-class, 24
FAKinSumResults, *11*, *19*, *45*, *54*, *55*, *61*
FAKinSumResults
(FAKinSumResults-class), 29
FAKinSumResults-class, 29
familialIncidenceRate, *21*, *22*
familialIncidenceRate
(PedigreeAnalysis), 48
familialIncidenceRate,FAData-method
(PedigreeAnalysis), 48
familialIncidenceRate,FAIncidenceRateResults-method
(FAIncidenceRateResults-class),
19
familialIncidenceRateTest, *4*, *18*, *21–23*,
28, *32*, *37*, *41*
familialIncidenceRateTest
(PedigreeAnalysis), 48
familialIncidenceRateTest,FAData-method
(PedigreeAnalysis), 48
family (FAData-class), 5
family,FAData-method (FAData-class), 5
FAProbResults, *11*, *45*, *54*, *55*, *61*
FAProbResults (FAProbResults-class), 33
FAProbResults-class, 33
FAStdIncidenceRateResults, *52*
FAStdIncidenceRateResults
(FAStdIncidenceRateResults-class),
37
FAStdIncidenceRateResults-class, 37
findFounders, *60*

findFounders (PedigreeUtils), 57
findFounders,data.frame-method (PedigreeUtils), 57
findFounders,FAData-method (PedigreeUtils), 57
findFounders,pedigree-method (PedigreeUtils), 57
findFounders,pedigreeList-method (PedigreeUtils), 57
findInterval, *50*
fsir, *39, 40*
fsir (PedigreeAnalysis), 48
fsir,FAData-method (PedigreeAnalysis), 48
fsir,FAStdIncidenceRateResults-method (FAStdIncidenceRateResults-class), 37
fsirTest, *4*, *18*, *23*, *28*, *32*, *37*, *39–41*
fsirTest (PedigreeAnalysis), 48
fsirTest,FAData-method (PedigreeAnalysis), 48

genealogicalIndexTest, *4*, *16*, *23*, *28*, *32*, *37*, *41*, *43*, *45*
genealogicalIndexTest (PedigreeAnalysis), 48
genealogicalIndexTest,FAData-method (PedigreeAnalysis), 48
generationsFrom (PedigreeUtils), 57
generationsFrom,data.frame-method (PedigreeUtils), 57
generationsFrom,FAData-method (PedigreeUtils), 57
generationsFrom,pedigree-method (PedigreeUtils), 57
generationsFrom,pedigreeList-method (PedigreeUtils), 57
getAll, *11*, *15*, 43, *50*
getAll,data.frame-method (getAll), 43
getAll,FAData-method (getAll), 43
getAll,pedigree-method (getAll), 43
getAll,pedigreeList-method (getAll), 43
getAncestors (PedigreeUtils), 57
getAncestors,data.frame-method (PedigreeUtils), 57
getAncestors,FAData-method (PedigreeUtils), 57
getAncestors,pedigree-method (PedigreeUtils), 57
getAncestors,pedigreeList-method (PedigreeUtils), 57
getChildren (PedigreeUtils), 57
getChildren,data.frame-method (PedigreeUtils), 57
getChildren,FAData-method (PedigreeUtils), 57
getChildren,pedigree-method (PedigreeUtils), 57
getChildren,pedigreeList-method (PedigreeUtils), 57
getCommonAncestor (PedigreeUtils), 57
getCommonAncestor,data.frame-method (PedigreeUtils), 57
getCommonAncestor,FAData-method (PedigreeUtils), 57
getCommonAncestor,pedigree-method (PedigreeUtils), 57
getCommonAncestor,pedigreeList-method (PedigreeUtils), 57
getExternalMatched, *15, 16*, *50, 51*
getExternalMatched (getAll), 43
getExternalMatched,data.frame-method (getAll), 43
getExternalMatched,FAData-method (getAll), 43
getExternalMatched,pedigree-method (getAll), 43
getExternalMatched,pedigreeList-method (getAll), 43
getFounders (PedigreeUtils), 57
getFounders,data.frame-method (PedigreeUtils), 57
getFounders,FAData-method (PedigreeUtils), 57
getGenerationMatched, *15, 16*, *50*
getGenerationMatched (getAll), 43
getGenerationMatched,data.frame-method (getAll), 43
getGenerationMatched,FAData-method (getAll), 43
getGenerationMatched,pedigree-method (getAll), 43
getGenerationMatched,pedigreeList-method (getAll), 43
getGenerationSexMatched, *15, 16*, *50*
getGenerationSexMatched (getAll), 43
getGenerationSexMatched,data.frame-method

(getAll), 43
getGenerationSexMatched,FAData-method (getAll), 43
getGenerationSexMatched,pedigree-method (getAll), 43
getGenerationSexMatched,pedigreeList-method (getAll), 43
getMissingMate (PedigreeUtils), 57
getMissingMate,data.frame-method (PedigreeUtils), 57
getMissingMate,FAData-method (PedigreeUtils), 57
getMissingMate,pedigree-method (PedigreeUtils), 57
getMissingMate,pedigreeList-method (PedigreeUtils), 57
getSexMatched, *15*, *50*
getSexMatched (getAll), 43
getSexMatched,data.frame-method (getAll), 43
getSexMatched,FAData-method (getAll), 43
getSexMatched,pedigree-method (getAll), 43
getSexMatched,pedigreeList-method (getAll), 43
getSiblings (PedigreeUtils), 57
getSiblings,data.frame-method (PedigreeUtils), 57
getSiblings,FAData-method (PedigreeUtils), 57
getSiblings,pedigree-method (PedigreeUtils), 57
getSiblings,pedigreeList-method (PedigreeUtils), 57
getSingletons (PedigreeUtils), 57
getSingletons,data.frame-method (PedigreeUtils), 57
getSingletons,FAData-method (PedigreeUtils), 57

kinship, *4*, *23*, *28*, *32*, *41*
kinship (FAData-class), 5
kinship,FAData-method (FAData-class), 5
kinshipGroupTest, *4*, *18*, *23*, *24*, *26*, *32*, *37*, *41*
kinshipGroupTest (PedigreeAnalysis), 48
kinshipGroupTest,FAData-method (PedigreeAnalysis), 48
kinshipPairs, 46, *61*
kinshipSumTest, *4*, *18*, *23*, *28*, *31*, *32*, *37*, *41*
kinshipSumTest (PedigreeAnalysis), 48
kinshipSumTest,FAData-method (PedigreeAnalysis), 48

lambda (FAStdIncidenceRateResults-class), 37
lambda,FAStdIncidenceRateResults-method (FAStdIncidenceRateResults-class), 37

p.adjust, *3*, *15*, *20*, *26*, *30*, *35*, *39*
ped2graph (PedigreeUtils), 57
pedigree, *10*, *11*, *45*, *55*, *61*
pedigree (FAData-class), 5
pedigree,FAData-method (FAData-class), 5
pedigree,missing-method (FAData-class), 5
pedigree<- (FAData-class), 5
pedigree<-,FAData-method (FAData-class), 5
PedigreeAnalysis, *5*, *11*, *20*, *21*, *26*, *31*, *39*, 48, *61*
pedigreeSize (FAData-class), 5
pedigreeSize,FAData-method (FAData-class), 5
PedigreeUtils, *5*, *11*, *47*, 57
phenotypedIndividuals (FAData-class), 5
phenotypedIndividuals,FAData-method (FAData-class), 5
plot.pedigree, *66*
plotPed, *4*, *15–18*, *20–23*, *25–28*, *30–32*, *34–41*, 63
plotPed (FAData-class), 5
plotPed,FAData-method (FAData-class), 5
plotPed,FAGenIndexResults-method (FAGenIndexResults-class), 14
plotPed,FAIncidenceRateResults-method (FAIncidenceRateResults-class), 19
plotPed,FAKinGroupResults-method (FAKinGroupResults-class), 24
plotPed,FAKinSumResults-method (FAKinSumResults-class), 29
plotPed,FAProbResults-method (FAProbResults-class), 33
plotPed,FAStdIncidenceRateResults-method (FAStdIncidenceRateResults-class),

37
plotRes (FAGenIndexResults-class), 14
plotRes,FAGenIndexResults-method (FAGenIndexResults-class), 14
plotRes,FAIncidenceRateResults-method (FAIncidenceRateResults-class), 19
plotRes,FAKinGroupResults-method (FAKinGroupResults-class), 24
plotRes,FAKinSumResults-method (FAKinSumResults-class), 29
plotRes,FAStdIncidenceRateResults-method (FAStdIncidenceRateResults-class), 37
probabilityTest, *4*, *18*, *23*, *28*, *32*, *35*, *37*, *41*
probabilityTest (PedigreeAnalysis), 48
probabilityTest,FAData-method (PedigreeAnalysis), 48

removeSingletons (PedigreeUtils), 57
removeSingletons,data.frame-method (PedigreeUtils), 57
removeSingletons,FAData-method (PedigreeUtils), 57
result (FAProbResults-class), 33
result,FABinTestResults-method (FABinTestResults-class), 2
result,FAGenIndexResults-method (FAGenIndexResults-class), 14
result,FAIncidenceRateResults-method (FAIncidenceRateResults-class), 19
result,FAKinGroupResults-method (FAKinGroupResults-class), 24
result,FAKinSumResults-method (FAKinSumResults-class), 29
result,FAProbResults-method (FAProbResults-class), 33
result,FAStdIncidenceRateResults-method (FAStdIncidenceRateResults-class), 37
resultForId (FAStdIncidenceRateResults-class), 37
resultForId,FAStdIncidenceRateResults-method (FAStdIncidenceRateResults-class), 37
runSimulation (FAProbResults-class), 33
runSimulation,FAGenIndexResults-method (FAGenIndexResults-class), 14
runSimulation,FAIncidenceRateResults-method (FAIncidenceRateResults-class), 19
runSimulation,FAKinGroupResults-method (FAKinGroupResults-class), 24
runSimulation,FAKinSumResults-method (FAKinSumResults-class), 29
runSimulation,FAProbResults-method (FAProbResults-class), 33
runSimulation,FAStdIncidenceRateResults-method (FAStdIncidenceRateResults-class), 37

shareKinship (PedigreeUtils), 57
shareKinship,FAData-method (PedigreeUtils), 57
shareKinship,FAKinGroupResults-method (FAKinGroupResults-class), 24
shareKinship,FAProbResults-method (FAProbResults-class), 33
show (FAData-class), 5
show,FAData-method (FAData-class), 5
sliceAge (PedigreeAnalysis), 48
subPedigree (PedigreeUtils), 57
switchPlotfun, *8*, *28*
switchPlotfun (plotPed), 63

timeAtRisk (FAIncidenceRateResults-class), 19
timeAtRisk,FAIncidenceRateResults-method (FAIncidenceRateResults-class), 19
timeAtRisk<- (FAIncidenceRateResults-class), 19
timeAtRisk<-,FAIncidenceRateResults-method (FAIncidenceRateResults-class), 19
timeInStrata (FAStdIncidenceRateResults-class), 37
timeInStrata,FAStdIncidenceRateResults-method (FAStdIncidenceRateResults-class), 37
trait, *4*, *18*, *23*, *28*, *32*, *37*, *41*
trait (FAData-class), 5

trait,FAData-method *(FAData-class)*, 5
trait<- *(FAData-class)*, 5
trait<-,FABinTestResults-method
(FABinTestResults-class), 2
trait<-,FAData-method *(FAData-class)*, 5
trait<-,FAGenIndexResults-method
(FAGenIndexResults-class), 14
trait<-,FAIncidenceRateResults-method
(FAIncidenceRateResults-class),
19
trait<-,FAKinGroupResults-method
(FAKinGroupResults-class), 24
trait<-,FAKinSumResults-method
(FAKinSumResults-class), 29
trait<-,FAProbResults-method
(FAProbResults-class), 33
trait<-,FAStdIncidenceRateResults-method
(FAStdIncidenceRateResults-class),
37
traitByClique *(FAProbResults-class)*, 33
traitByClique,FAProbResults-method
(FAProbResults-class), 33