Package ‘GENLIB’

March 20, 2015

Type Package
Title Genealogical Data Analysis
Version 1.0.3
Date 2015-03-18
Description Genealogical data analysis including descriptive statistics (e.g., kinship and inbreeding coefficients) and gene-dropping simulations.
License GPL (>= 2)
LazyLoad yes
Depends R (>= 3.1.0), Rcpp (>= 0.9.10)
Imports kinship2, methods, bootstrap, Matrix, lattice, quadprog, foreach, parallel, doParallel
LinkingTo Rcpp
Author Louis Houde [aut],
Jean-Francois Lefebvre [aut],
Valery Roy-Lagace [aut],
Sebastien Lemieux [aut],
Michael J. Fromberger [ctb],
Marie-Helene Roy-Gagnon [cre]
Maintainer Marie-Helene Roy-Gagnon <mroygagn@uottawa.ca>
NeedsCompilation yes
X-CRAN-Comment Version 1.0 was removed for copyright violation.
Repository CRAN
Date/Publication 2015-03-20 18:02:50

R topics documented:

GenlibR-package .......................................................... 3
Classes of basic handling of genealogy data .......................... 3
Classes of GLgroupe handling ........................................... 4
gen.branching ............................................................. 6
gen.children .............................................................. 7
R topics documented:

- gen.climbPAR .......................................................... 8
- gen.completeness ...................................................... 8
- gen.completenessVar .................................................. 9
- gen.depth ............................................................... 11
- gen.f ................................................................. 11
- gen.fCI ................................................................. 13
- gen.find.Min.Distance.MRCA ......................................... 14
- gen.findDistance ..................................................... 15
- gen.findFounders ..................................................... 16
- gen.findMRCA ........................................................ 17
- gen.founder ............................................................ 18
- gen.gc ................................................................. 19
- gen.genealogy ........................................................ 20
- gen.genout ............................................................. 21
- gen.getAncestorsPAR .................................................. 22
- gen.getFoundersPAR ................................................... 23
- gen.graph .............................................................. 23
- gen.half.founder ..................................................... 25
- gen.implex ............................................................. 26
- gen.implexVar ......................................................... 27
- gen.lineages ........................................................... 28
- gen.max ................................................................. 29
- gen.mean ............................................................... 30
- gen.meangendepth .................................................... 31
- gen.meangendepthVar .................................................. 32
- gen.min ................................................................. 33
- gen.nochildren ....................................................... 34
- gen.noind ............................................................. 35
- gen.nomen ............................................................. 35
- gen.nowomen .......................................................... 36
- gen.occ ............................................................... 37
- gen.parent ............................................................ 38
- gen.phi ................................................................. 39
- gen.phiCI ............................................................... 40
- gen.phiMean ........................................................... 41
- gen.phiOver ............................................................ 42
- gen.pro ................................................................. 43
- gen.rec ............................................................... 44
- gen.sibship ............................................................ 45
- gen.simuProb ........................................................... 46
- gen.simuSample ....................................................... 47
- gen.simuSampleFreq .................................................... 48
- gen.simuSet ............................................................ 49
- genea140 ............................................................... 51
- geneaJi ................................................................. 51
- GLgen-class ............................................................ 52
- GLgroupe-class ......................................................... 52
- GLmultiList-class ...................................................... 53
Classes of basic handling of genealogy data

Description

Genealogical data analysis including descriptive statistics (e.g., kinship and inbreeding coefficients) and gene-dropping simulations.

Details

Package: GenlibR
Type: Package
Version: 1.0
Date: 2012-04-04
License: GPL (>=2)
LazyLoad: yes

Author(s)

Louis Houde, Jean-Francois Lefebvre, Valery Roy-Lagace, Sebastien Lemieux

Objects from the Class

Objects can be created by calls of the form

new("GLmultiVector", 'matrix', depth = 'integer')
new("GLmultiArray4", 'Array', depth = 'integer')
new("GLmultiMatrix", 'Array', depth = 'integer')
new("GLmultiNumber", 'numeric', depth = 'integer', .Names = 'character')

Description

Objects created to carry information mostly between invisible functions.
Classes of GLgroupe handling

Slots

GLmultiVector
Object of class "matrix"

.Data: Object of class "integer"

GLmultiArray4, GLmultiMatrix
.Data: Object of class "array"

depth: Object of class "integer"

GLmultiNumber
.Data: Object of class "numeric"

depth: Object of class "integer"

.Names: Object of class "character"

Extends


Methods

[< signature(x = "GLmultiVector", i = "ANY", j = "ANY", value = "ANY")]

[ signature(x = "GLmultiVector", i = "ANY", j = "ANY", drop = "ANY")]

depth signature(x = "GLmultiVector")

Dim signature(object = "GLmultiVector")

Examples

showClass("GLmultiVector")
showClass("GLmultiArray4")
showClass("GLmultiMatrix")
showClass("GLmultiNumber")

Classes of GLgroupe handling

Classes: "GLMultiPhiGroup"  "GLMultiPhiGroupSingle"
          "GLMultiFGroup"  "GLMultiFGroupSingle"
          "GLCGMatrixGroupSingle"

Description

Objects used to carry information mostly between invisible functions.
Classes of GLgroupe handling

Objects from the Class

Objects can be created by calls of the form

new("GLmultiPhiGroup", MatriceCG, groupe = 'GLgroupe', grindice = 'list')
new("GLmultiPhiGroupSingle", MatriceCG, groupe = 'GLgroupe', grindice = 'list')
new("GLmultiFGroup", MatriceCG, groupe = 'GLgroupe', grindice = 'list')
new("GLmultiFGroupSingle", MatriceCG, groupe = 'GLgroupe', grindice = 'list')
new("GLCGMatrixGroupSingle", MatriceCG, groupe = 'GLgroupe', grindice = 'list')

Slots

GLCGMatrixGroupSingle, GLmultiFGroupSingle, GLmultiPhiGroupSingle

Object of class "matrix"

Data: Object of class "GLgroupe"

grindice: Object of class "list" containing the indices of the probands of 'groupe'

GLmultiPhiGroup:

.Data: Object of class "GLmultiMatrix"
groupe: Object of class "GLgroupe"
grindice: Object of class "list" containing the indices of the probands of 'groupe'

GLmultiFGroup:

.Data: Object of class "GLmultiVector"
groupe: Object of class "GLgroupe"
grindice: Object of class "list" containing the indices of the probands of 'groupe'

Extends


Methods

[<- signature(x = "GLCGMatrixGroupSingle", i = "ANY", j = "ANY", value = "ANY"):
  ...

[ signature(x = "GLCGMatrixGroupSingle", i = "ANY", j = "ANY", drop = "ANY"):
  ...

  groupe signature(x = "GLCGMatrixGroupSingle"):

Examples

showClass("GLCGMatrixGroupSingle")
showClass("GLmultiFGroup")
showClass("GLmultiFGroupSingle")
showClass("GLmultiPhiGroupSingle")
showClass("GLmultiPhiGroup")
**Description**

Creates an object of class GLgen from an existing GLgen object by selecting specified individuals.

**Usage**

```r
gen.branching(gen, pro = 0, ancestors = gen.founder(gen), bflag = 0)
```

**Arguments**

- `gen`: An object of class GLgen obtained with `gen.genealogy`, `gen.lineages` or `gen.branching`. Required.
- `pro`: Vector of proband id numbers to be included. Default is 0, which will select all individuals without children.
- `ancestors`: Vector of ancestors id numbers to be included. Default includes all founders in the original GLgen object.
- `bflag`: If set to 0 (default and recommended), probands and founders are filtered to quicken the calculations.

**Value**

returns a GLgen object

**Note**

Note that if there are no links between some probands and ancestors, these ids will not be included.

**See Also**

`gen.genealogy`, `gen.graph`, `gen.lineages`

**Examples**

```r
data(geneajiI)
genJi<-gen.genealogy(geneajiI)
genJi_part<-gen.branching(genJi, pro=c(2,28))

## Not run: Plots of original genealogy and of the branched version
layout(matrix(1:2,c(1,2),byrow=TRUE))
gen.graph(genJi)
mtext("Original", line=2, cex=1.2)
gen.graph(genJi_part)
mtext("Branched tree\nfor individuals 2 and 28", line=1, cex=1.2)
```
gen.children

*Get id numbers of children*

**Description**

Returns the id numbers of the children of specified individuals.

**Usage**

```r
gen.children( gen, individuals, ...)
```

**Arguments**

- `gen`: An object of class GLgen obtained with `gen.genealogy`, `gen.lineages` or `gen.branching`. Required.
- `individuals`: Vector of individual id numbers. Required.
- `...`: Option to pass additional arguments automatically between methods. Internal use only.

**Value**

returns a vector of integer

**See Also**

`gen.genealogy`, `gen.pro`, `gen.founder`, `gen.sibship`, `gen.parent`

**Examples**

```r
data(geneaJi)
genJi<-gen.genealogy(geneaJi)
gen.children(genJi, individuals=14)

data(genea140)
gen140<-gen.genealogy(genea140)
children51052<-gen.children(gen140, 51052)
gen.graph(gen140, pro=c(children51052))
```
### gen.climbPAR

**Description**

`gen.climbPAR` is an internal function used by `gen.findMRCA`.

**Usage**

```r
gen.climbPAR(gen, individuals, founder)
```

**Arguments**

- `gen`: The genealogy to consider.
- `individuals`: Proband(s) to consider.
- `founder`: The founder by which the distance is calculated.

**Value**

Returns a list containing the founder ID and the distance.

**See Also**

`gen.findMRCA`, `gen.getAncestorsPAR`

**Examples**

```r
data(geneaji)
genJi<-gen.genealogy(geneaji)
gen.fimdMRCA(genJi, individuals=c(1,29), NbProcess = 1)
```

---

### gen.completeness

**Description**

Returns the completeness of the genealogical data for the specified probands.

**Usage**

```r
gen.completeness( gen, pro="0", genNo=-1, type="MEAN", ...)
```
Arguments

- **gen**: An object of class GLgen obtained with `gen.genealogy`, `gen.lineages` or `gen.branching`. Required.

- **pro**: Vector of proband id numbers to be included. Default is 0, which will select all individuals without children.

- **genNo**: Vector of generation numbers at which completeness should be calculated. Default is -1, which calculates completeness at each generation.

- **type**: If `type="MEAN"` (default), mean completeness over all specified probands is calculated. If `type="IND"`, completeness is calculated for each specified proband.

- **...**: Option to pass additional arguments automatically between methods. Internal use only.

Value

A data frame with each generation at which the completeness is calculated as rows and one column when `type` is "MEAN". When `type` is "IND", the number of columns equals the number of probands specified.

References


See Also

- `gen.genealogy`
- `gen.rec`
- `gen.occ`
- `gen.implex`
- `gen.meangendepth`

Examples

```r
data(geneaji)
genJi<-gen.genealogy(geneaji)
gen.completeness(genJi, type="IND")
## Not run: For the 4th generation
 genie.completeness(genJi, type="IND", genNo=4)

data(genea140)
gen140<-gen.genealogy(genea140)
gen.completeness(gen140)
```

---

**gen.completenessVar**  
**Variance of completeness index**

Description

Returns the variance of the completeness index across probands.
gen.completenessVar

Usage

gen.completenessVar( gen, pro = "0", genNo = -1, ...)

Arguments

gen  An object of class GLgen obtained with gen.genealogy, gen.lineages or gen.branching. Required.

pro  Vector of proband id numbers to be included. Default is 0, which will select all individuals without children.

genNo  Vector of generation numbers at which completeness should be calculated. Default is -1, which calculates completeness at each generation.

...  Option to pass additional arguments automatically between methods. Internal use only.

Value

A data frame with one column and each generation at which the variance is calculated as rows.

References


See Also

gen.gc

Examples

data(geneaJi)
genJi<-gen.genealogy(geneaJi)
gen.completeness(genJi, type="IND")
gen.completenessVar(genJi)
## Not run: For the 4th generation
ngen.completenessVar(genJi, genNo=4)

data(genea140)
gen140<-gen.genealogy(genea140)
gen.completeness(gen140)
gen.completenessVar(gen140)
### gen.depth

**Genealogy depth**

**Description**

Returns the number of generations in the genealogy from a GLgen object.

**Usage**

```r
gen.depth(gen)
```

**Arguments**

- `gen`: An object of class GLgen obtained with `gen.genealogy`, `gen.lineages` or `gen.branching`. Required.

**Value**

Returns a vector of integer

**See Also**

- `gen.genealogy`  
- `gen.min`  
- `gen.mean`  
- `gen.min`  
- `gen.meangendepth`

**Examples**

```r
data(geneajiI)
genjiI<-gen.genealogy(geneajiI)
gen.depth(genjiI)

data(genea1TPI)
gen1TPI<-gen.genealogy(genea1TPI)
gen.depth(gen1TPI)
```

---

### gen.f

**Inbreeding coefficient**

**Description**

Returns the inbreeding coefficients of the specified probands

**Usage**

```r
gen.f( gen, pro, depthmin= (gen.depth(gen)-1), depthmax= (gen.depth(gen)-1))
```
Arguments

- **gen**: An object of class GLgen obtained with `gen.genealogy`, `gen.lineages` or `gen.branching`. Required.
- **pro**: Vector of proband id numbers to be included.
- **depthmin**: Minimum genealogical depth to consider in the calculation. Default is the whole genealogy.
- **depthmax**: Maximum genealogical depth to consider in the calculation. Default is the whole genealogy.

Value

A vector or GLmultiVector object depending on the number of generations treated. GLmultiVector is an array of vectors, one for each depth. Array of size `('depthMax'-'depthMin')`, vectors of size `length(pro)`. A vector of size `length(pro)` is returned when `('depthMax'-'depthMin')` equals 1.

References


See Also

- `gen.genealogy`
- `gen.phi`

Examples

```r
data(geneajiI)
genJi<-gen.genealogy(geneajiI)
f_values<-gen.f(genJi)
f_values
f_allgen<-gen.f(genJi, depthmin=1)
f_allgen<-unclass(f_allgen)

plot(1:7,f_allgen[1,],type="b",xlab="Generation",ylab="Inbreeding values",ylim=c(0,0.25),pch=0)
points(1:7,f_allgen[3,], type="b", lty=12, pch=1)
legend("topright", legend=c("Individual 1", "Individual 29"),lty=c(1,12), pch=c(0,1))

data(genea1TPI)
gen1TPI<-gen.genealogy(genea1TPI)
inbreeding_val<-gen.f(gen1TPI)
boxplot(inbreeding_val, horizontal=TRUE, xlab="Inbreeding values")
```
gen.fCI

Average inbreeding coefficient confidence interval

Description

Returns the confidence interval of the average inbreeding among specified individuals.

Usage

```
gen.fCI(vectF, prob=c(0.025, 0.05, 0.95, 0.975), b="5000", print.it="F")
```

Arguments

- **vectF**: A vector of inbreeding coefficients obtained for example with `gen.f`. Required.
- **prob**: Confidence limits probabilities. Default is `prob=c(0.025, 0.05, 0.95, 0.975)`.
- **b**: Number of simulations used to calculate the confidence interval. Default is `b="5000"`.
- **print.it**: If `print.it="F"` (default), progression in the number of simulations performed is not displayed.

Value

A vector or a GLmultiVector object depending on the type of `vectF` entered. If `vectF` is a GLmultiVector, an array of vectors (GLmultiVector), one for each depth, is returned. Array of size (`'depthMax'-'depthMin'`), vectors of size `length(probs)`). If `vectF` is a GLmultiVector, a vector of size `length(probs)` is returned.

See Also

`gen.f`

Examples

```
data(genealogy140)
gen140 <- gen.genealogy(genealogy140)
gen10 <- gen.branching(gen140, pro=gen.pro(gen140)[c(1:10)])
fval <- gen.f(gen10)
gen.fCI(fval)
```
**gen.find.Min.Distance.MRCA**

Finds the minimum distances between pairs of individuals given a set of ancestors.

Description

Returns the shortest distances (number of meioses) between pairs of probands given the matrix of MRCAs output by the gen.findMRCA function.

Usage

```r
gen.find.Min.Distance.MRCA(genMatrix, individuals="ALL", ancestors="ALL")
```

Arguments

- `genMatrix`: Matrix of most recent common ancestors, MRCAs, obtained with gen.findMRCA. Required.
- `individuals`: Vector of proband id numbers to include. All are included by default.
- `ancestors`: Vector of MRCA id numbers to include. All are included by default.

Value

returns a matrix

See Also

- `gen.genealogy`
- `gen.founder`
- `gen.findMRCA`
- `gen.findDistance`
- `gen.findFounders`

Examples

```r
data(genea1j)
geni<-gen.genealogy(genea1j)
matMRCA<-gen.findMRCA(geni, individuals=c(1,29), NbProcess = 1)
gen.find.Min.Distance.MRCA(matMRCA)

## Not run: Increasing NbProcess will decrease execution time

data(genea1o)
geni0<-gen.genealogy(genea1o)
matMRCA0<-gen.findMRCA(geni0, individuals=c(409033,408728,408828), NbProcess = 1)
gen.find.Min.Distance.MRCA(matMRCA, individuals=c(409033,408728))
```
**gen.findDistance**

*Minimum genetic distance between two individuals*

**Description**

Returns the minimum distance (number of meioses) between the specified individuals through one specified ancestor.

**Usage**

```
gen.findDistance(gen, individuals, ancestor)
```

**Arguments**

- **gen**
  
  An object of class GLgen obtained with `gen.genealogy`, `gen.lineages` or `gen.branching`. Required.

- **individuals**
  
  A vector of individual id numbers between which to calculate the distance. Required.

- **ancestor**
  
  A common ancestor to the specified individuals. Required.

**Value**

returns a numeric value

**See Also**

- `gen.genealogy`
- `gen.founder`
- `gen.findMRCA`
- `gen.findFounders`
- `gen.find.Min.Distance.MRCA`

**Examples**

```r
data(geneaJi)
genJi <- gen.genealogy(geneaJi)
gen.findDistance(genJi, individuals=c(1,29), ancestor=17)
gen.findDistance(genJi, individuals=c(1,29), ancestor=26)

data(genea140)
gen140 <- gen.genealogy(genea140)
gen.findDistance(gen140, individuals=c(409033,408728), ancestor=38714)
gen.findDistance(gen140, individuals=c(408728,408828), ancestor=95080)
```
gen.findFounders  Find common founder ancestors

Description

Returns all the ancestors that are founders for specified individuals in the genealogy.

Usage

gen.findFounders(gen, individuals, NbProcess=detectCores()-1)

Arguments

gen An object of class GLgen obtained with gen.genealogy, gen.lineages or gen.branching. Required.
individuals Vector of individual id numbers for which to find the founders. Required.
NbProcess Number of processes to use when running this function. Default=detectCores()-1

Value

A vector of integers

Note

Uses slave processes to make the search faster. Those slave processes launch the function gen.getFoundersMPI.

See Also

gen.genealogy gen.founder gen.findMRCA gen.findDistance gen.find.Min.Distance.MRCA

Examples

data(geneaJ1)
genJ1<-gen.genealogy(geneaJ1)
gen.founder(genJ1)
gen.findFounders(genJ1, individuals=c(1,29), NbProcess = 1)

data(genea140)
gen140<-gen.genealogy(genea140)
gen2<-gen.branching(gen140,pro=c(409033,408728))
## Not run: 615 founders in genealogy of #409033 and #408728
all_founders<-gen.founder(gen2)
length(all_founders)

all_commonFounders<-gen.findFounders(gen140, individuals=c(409033,408728), NbProcess = 1)
length(all_commonFounders) ## Not run: 127 founders common to #409033 and #408728
Finding most recent common ancestors, MRCAs

Description

Returns MRCAs of pairs of specified individuals and the distance (number of meioses) between individuals through the MRCAs.

Usage

```
gen.findMRCA(gen, individuals, NbProcess=detectCores()-1)
```

Arguments

- **gen**: An object of class GLgen obtained with `gen.genealogy`, `gen.lineages` or `gen.branching`. Required.
- **individuals**: Vector of individual id numbers for which to find the MRCAs. Required.
- **NbProcess**: Number of processes to use when running this function. Default = `detectCores()-1`.

Value

returns a matrix

Note

This function uses the parallel programming functions of `foreach`, `snow` and `doSNOW` for launching several processes.

See Also

- `gen.genealogy`, `gen.founder`, `gen.findFounders`, `gen.findDistance`, `gen.find.Min.Distance.MRCA`

Examples

```
data(geneaji)
genji<-gen.genealogy(geneaji)
gen.findMRCA(genji, individuals=c(1,29), NbProcess = 1)
```

## Not run: For a more complex example:
```
data(genea140)
gen140<-gen.genealogy(genea140)
all_commonFounders<-gen.findFounders(gen140, individuals=c(409033,408728), NbProcess = 1)
```

## Not run: 127 founders common to #409033 and #408728
```
length(all_commonFounders)
```

## Not run: 22 most recent common ancestors for #409033 and #408728
```
MRCA_2ind<-gen.findMRCA(gen140, individuals=c(409033,408728), NbProcess = 1)
```
Description

Returns the id numbers of the founders. Founders are defined as the individuals without parents in the genealogy (i.e., mother id=0, father id=0).

Usage

gen.founder( gen, ...)

Arguments

gen : An object of class GLgen obtained with gen.genealogy, gen.lineages or gen.branching. Required.

... : Option to pass additional arguments automatically between methods. Internal use only.

Value

A vector of integer

See Also

gen.genealogy gen.pro gen.half.founder gen.parent

Examples

data(geneaji)
genJi<-gen.genealogy(geneaji)
## Not run: There are 6 founders
gen.founder(genJi)

data(geneal40)
gen140<-gen.genealogy(geneal40)
founder140<-gen.founder(gen140)
## Not run: There are 7399 founders
length(founder140)
The function `genNgc` computes the genetic contribution of ancestors to the gene pool of specific probands. It takes the following arguments:

- `gen`: An object of class `GLgen` obtained with `gen.genealogy`, `gen.lineages` or `gen.branching`. Required.
- `pro`: Vector of proband id numbers to be included. Default is 0, which will select all individuals without children.
- `ancestors`: Vector of ancestors id numbers to be included. Default is 0, which will select all individuals without parents.
- `vctProb`: Vector of transmission probabilities. Default is `vctProb=c(0.5,0.5,0.5,0.5)`.
- `typeCG`: IF `typeCG="IND"` (default), the genetic contribution from each ancestor is calculated for each proband. If `typeCG="MEAN"`, the average (over all probands) genetic contributions of each ancestor is returned. If `typeCG="PRODUCT"`, the product (over all probands) of genetic contributions is returned for each ancestor. If `typeCG="TOTAL"`, the sum (over all probands) of genetic contributions is returned for each ancestor. If `typeCG="CUMUL"`, ancestors are ranked in decreasing order of total contribution and cumulative contribution is returned.

The function returns a matrix with rows corresponding to probands and columns corresponding to ancestors when `typeCG="IND"`. For the other `typeCG` values, rows are ancestors and there is one column containing mean, product, total or cumulative values.

References:

See Also
`gen.genealogy gen.rec gen.occ gen.implex gen.meangendepth gen.completeness`
Examples

```r
data(geneaji)
genJi<-gen.genealogy(geneaji)
gc<-gen.gc(genJi, pro=c(1,29), ancestors=c(3,6,10,12,14,16,18,20,26,28))
gc
gc_cum<-gen.gc(genJi, pro=c(1,29), ancestors=c(3,6,10,12,14,16,18,20,26,28), type="CUMUL")
gc_cum
data(genea140)
gen140<-gen.genealogy(genea140)
gc<-gen.gc(gen140, pro=c(454422,676521,677273,717537,717634,717709,868572),
ancestors=c(18305,18528,31114,18325))
gc
## Not run: Mother–daughter transmission only with probability=0.5
gc_MoLi<-gen.gc(gen140, pro=c(217891,302718,409282,802424,409682,443151),
ancestors=c(18321,218231,296200,39066,18679,442607),vctProb=c(0,0,0.5,0))
gc_MoLi
```

---

### Description

Creates an object of class GLgen that contains the ascending genealogies derived from input data in pedigree format.

### Usage

```r
gen.genealogy(ped, autoComplete=FALSE, ...)
```

### Arguments

- **ped**
  A matrix or data frame with the following columns: individual id, father id, mother id, sex. Required. Individual id numbers must be numeric and unique. If an individual does not have a father and/or mother, the father and/or mother id numbers must be set to 0. Sex must be equal to M or 1 for males and F or 2 for females. The sex column is optional for this function but necessary for some other functions using GLgen objects.

- **autoComplete**
  If TRUE, any non-zero father and mother id numbers not appearing in the individual id column, will be added in the individual column as having no father or mother (both set to 0). Default to FALSE.

- **...**
  Option to pass additional arguments automatically between methods. Internal use only.

### Value

An object of class GLgen.
See Also

`gen.graph gen.genout`

Examples

```r
ind<-c(1,2,3,11:21,101:108,201:202)
father<-c(11,15,15,102,0,102,0,103,103,103,105,105,107,0,202,0,202,202,0,202,0,0,0)
mother<-c(12,14,14,101,0,101,0,104,104,106,106,108,108,0,201,0,201,201,0,201,0,0,0)
sex<-c(1,2,2,1,2,1,2,1,2,2,1,2,1,2,1,2,1,2,2,1)
gen.df<-data.frame(ind, father, mother, sex)
gen.genealogy(gen.df)

data(geneaji)
geneaji[1:5,]
genJi<-gen.genealogy(geneaji)
## Not run: Print basic information about the genealogy
genJi
```

---

**Description**

Creates data frame in pedigree format from an object of class GLgen

**Usage**

```r
gen.genout( gen, sorted = "F")
```

**Arguments**

- `gen` An object of class GLgen obtained with `gen.genealogy`, `gen.lineages` or `gen.branching`. Required.
- `sorted` If `sorted="F"` (default), individual id number are not sorted in output data. Id numbers for parents will be placed before their children ids. If `sorted="T"`, individual id numbers will be sorted.

**Value**

returns a data.frame containing the following: `ind father mother sex`

**See Also**

`gen.genealogy gen.graph gen.branching gen.lineages`
Examples

```
data(geneaji)
## Not run: original data is a data.frame
geneaji[1:12,]

genJi<-gen.genealogy(geneaji)
## Not run: as a genealogy object
genJi

## Not run: Genealogy as a data.frame
genJi_df<-gen.genout(genJi)
genJi_df[1:12,]

## Not run: Maternal lineage
genJi_Mali<-gen.lineages(geneaji, maternal = TRUE)
## Not run: Maternal lineage as a data.frame
genJi_Mali_df<-gen.genout(genJi_Mali)
genJi_Mali_df
```

---

gen.getAncestorsPAR  

*Internal function used by gen.findMRCA*

Description

`gen.getAncestorsPAR` is an internal function used by `gen.findMRCA`.

Usage

```
gen.getAncestorsPAR(gen, pro)
```

Arguments

- `gen` The genealogy to consider.
- `pro` probands to consider

Value

Vector of the founders identity.

See Also

```
gen.findMRCA gen.climbPAR
```

Examples

```
data(geneaji)
genJi<-gen.genealogy(geneaji)
gen.findMRCA(genJi, individuals=c(1,29), NbProcess = 1)
```
Description

gen.getFoundersPAR is an internal function used by gen.findFounders.

Usage

```
gen.getFoundersPAR(gen, pro)
```

Arguments

- `gen` The genealogy to consider.
- `pro` probands to consider

Value

A vector of the founders identity.

See Also

```
gen.findFounders gen.findDistance
```

Examples

```
data(geneaji)
genj1<-gen.genealogy(geneaji)
gen.founder(genj1, individuals=c(1,29))
gen.findFounders(genj1, individuals=c(1,29), NbProcess = 1)
```

Description

Function that plots pedigrees of GLgen objects.

Usage

```
gen.graph( gen, pro=gen.pro(gen), ancestors=gen.founder(gen),
          indVarAffected=gen.genout(gen)$ind, varAffected=gen.genout(gen)$ind, cex="1",
          col="0", symbolsize="1", width="1", packed="F", align="T", ...)
```
Arguments

- **gen**: An object of class GLgen obtained with `gen.genealogy`, `gen.lineages` or `gen.branching`. Required.
- **pro**: Vector of proband id numbers to be included. Default is 'gen.pro(gen)', which will select all individuals without children.
- **ancestors**: Vector of ancestors id numbers to be included. Default is 'gen.founder(gen)', which will select all individuals without parents.
- **indVarAffected**: Vector of individuals id numbers used for labeling. Default is 'gen.genout(gen)$ind', which is all the individuals of the genealogy.
- **varAffected**: Vector of individuals labels. Default is 'gen.genout(gen)$ind', meaning that the individuals labels are their ids.
- **cex**: Controls text size (same as kinship2::plot.pedigree). Default is 1.
- **col**: Color for each id (same as kinship2::plot.pedigree). Default is 0, which assigns the same color to everyone.
- **symbolsize**: Controls symbol size (same as kinship2::plot.pedigree). Default is 1.
- **width**: For a packed pedigree, the minimum width allowed in the realignment of pedigrees (same as kinship2::plot.pedigree). Default is 1.
- **packed**: If T, uniform distance between all individuals at a given level (same as kinship2::plot.pedigree). Default is FALSE.
- **align**: Controls the extra effort spent trying to align children underneath parents, but without making the pedigree too wide (same as kinship2::plot.pedigree). Default is TRUE.
- **...**: Other arguments that can be passed to the kinship2::plot.pedigree.

Value

Returns the same invisible list as that returned by kinship2::plot.pedigree, which contains the following: 
- **plist**: list containing the information about the pedigree (n, id, pos, fam, spouse) x x-axis position y y-axis position **boxw** box width **boxh** box height **call** the call made to plot.pedigree() (kinship2 package)

See Also

`gen.genealogy` `gen.genout` `gen.branching`

Examples

```r
ind<-c(1,2,3,11:21,101:108,201:202)
father<-c(11,15,15,102,0,103,103,0,105,105,107,0,202,0,202,202,0,202,0,0,0,0)
mother<-c(12,14,14,101,0,101,0,104,0,106,106,108,0,201,0,201,0,201,0,0,0,0)
sex<-c(1,2,2,1,2,2,1,2,2,2,1,2,1,2,1,2,1,2,1,2,2,2,1)
gen.df<-data.frame(ind, father, mother, sex)
genEx<-gen.genealogy(gen.df)
gen.graph(genEx)
data(geneaji)
```
**gen.half.founder**

Get half-founder id numbers

**Description**

Returns the id numbers of the half-founders. Half-founders are defined as the individuals with only one founder parent in the genealogy.

**Usage**

```
gen.half.founder( gen, ...)```

**Arguments**

- `gen`: An object of class `GLgen` obtained with `gen.genealogy`, `gen.lineages` or `gen.branching`. Required.
- `...`: Option to pass additional arguments automatically between methods. Internal use only.

**Value**

returns a vector of integer

**See Also**

`gen.genealogy`, `gen.pro`, `gen.founder`, `gen.parent`

**Examples**

```
data(geneaji)
genJi<-gen.genealogy(geneaji)
genJi.graph(genJi)

# Not run: There are 2 half-founders
gen.half.founder(genJi)```
## Description

Returns the genealogical implex index (a measure of pedigree collapsing) for the specified probands.

## Usage

```r
gen.implex( gen, pro = "0", genNo = -1, type = "MEAN", onlyNewAnc = "F", ...)
```

## Arguments

- `gen` An object of class GLgen obtained with `gen.genealogy`, `gen.lineages` or `gen.branching`. Required.
- `pro` Vector of proband id numbers to be included. Default is 0, which will select all individuals without children.
- `genNo` Vector of generation numbers at which the implex should be calculated. Default is -1, which calculates the implex at each generation.
- `type` If `type="MEAN"` (default), implex index values are averaged over all specified probands. If `type="IND"`, the implex index is calculated for all specified proband together. If `type="ALL"`, the implex index is calculated for each specified proband.
- `onlyNewAnc` If `onlyNewAnc="F"` (default), all ancestors will be considered. If `onlyNewAnc="T"`, only new ancestors will be counted (i.e., an ancestor is not counted again if it has already been counted in another generation).

## Value

A data frame with each generation at which the completeness is calculated as rows and one column when `type` is "MEAN". When `type` is "IND", the number of columns equals the number of probands specified.

## References


## See Also

- `gen.genealogy`  
- `gen.occ`  
- `gen.rec`  
- `gen.meangendepth`  
- `gen.completeness`  
- `gen.gc`
Examples

```r
data(geneaji)
genJi<-gen.genealogy(geneaji)
gen.impex(genJi)
gen.impex(genJi, type="IND")
## Not run: For the 5th generation
gen.impex(genJi, type="IND", genNo=5)
```

```r
data(geneal0)
gen10<-gen.genealogy(geneal0)
gen.impex(gen10)
gen.impex(gen10, pro=c(454422, 676521, 677273, 717537, 717634, 717709, 868572))
gen.impex(gen10, pro=c(454422, 676521, 677273, 717537, 717634, 717709, 868572), type="IND")
```

---

**gen.impexVar**

**Variance of genealogical implex**

**Description**

Returns the variance of the implex index across probands.

**Usage**

```r
gen.impexVar( gen, pro = "0", onlyNewAnc = "F", genNo = -1, ...)
```

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gen</td>
<td>An object of class GLgen obtained with gen.genealogy, gen.lineages or gen.branching. Required.</td>
</tr>
<tr>
<td>pro</td>
<td>Vector of proband id numbers to be included. Default is 0, which will select all individuals without children.</td>
</tr>
<tr>
<td>onlyNewAnc</td>
<td>If onlyNewAnc=&quot;F&quot; (default), all ancestors will be considered. If onlyNewAnc=&quot;T&quot;, only new ancestors will be counted (i.e., an ancestor is not counted again if it has already been counted in another generation).</td>
</tr>
<tr>
<td>genNo</td>
<td>Vector of generation numbers at which the implex should be calculated. Default is -1, which calculates the implex at each generation.</td>
</tr>
<tr>
<td>...</td>
<td>Additional arguments to be passed to methods.</td>
</tr>
</tbody>
</table>

**Value**

A data frame with one column and each generation at which the variance is calculated as rows.

**References**

See Also

gcn. gc

Examples

data(geneaji)
genJi<-gen.genealogy(geneaji)
genimplex(genJi, type="IND")
genimplexVar(genJi)
## Not run: For the 5th generation
genimplexVar(genJi, genNo=5)

data(genea10)
gen10<-gen.genealogy(genea10)
genimplex(gen10, pro=c(454422, 676521, 677273, 717537, 717634, 717709, 868572), type="IND")
genimplexVar(gen10, pro=c(454422, 676521, 677273, 717537, 717634, 717709, 868572), type="IND")

---

**gen.lineages**

Create object of class GLgen for maternal or paternal lineages

Description

Creates an object of class GLgen that contains maternal or paternal lineages selected from input data in pedigree format

Usage

gen.lineages(ped, pro = "0", maternal = "T", ...)

Arguments

- **ped**: A matrix or data frame with the following columns: individual id, father id, mother id, sex. Required. Individual id numbers must be numeric and unique. If an individual does not have a father and/or mother, the father and/or mother id numbers must be set to 0. All non-zero father and mother id numbers must also appear in the individual id column. Sex must be equal to M or 1 for males and F or 2 for females. The sex column is optional for this function but necessary for some other functions using GL-gen objects.
- **pro**: Vector of individual id numbers for which lineages should be included. Optional.
- **maternal**: If mat="T" (default), maternal lineages are selected. mat="F" returns paternal lineages.
- **...**: Option to pass additional arguments automatically between methods. Internal use only.
```r
data(geneajiI)
genJi <- gen.genealogy(geneajiI)
genJi_MaLi<-gen.lineages(geneajiI, maternal = TRUE)
genJi_FaLi<-gen.lineages(geneajiI, maternal = FALSE)

## Not run: Plots of original genealogy and maternal and paternal lineages
layout(matrix(1:3,c(1,3),byrow=TRUE), widths =c(3,1,1), heights = 1)
gen.graph(genJi)
mtext("Original", line=2)
gen.graph(genJi_MaLi)
mtext("Maternal\nlineages", line=1)
gen.graph(genJi_FaLi)
mtext("Paternal\nlineages", line=1)
```

## gen.max

### Maximum number of generations

**Description**

Returns the maximum number of generations between all probands and the individuals specified. Probands are defined as the individuals without children in the genealogy.

**Usage**

```r
gen.max( gen, individuals)
```

**Arguments**

- `gen` An object of class GLgen obtained with `gen.genealogy`, `gen.lineages` or `gen.branching`. Required.
- `individuals` A vector of ids specifying the individuals to include in the calculation. Required.

**Value**

returns a vector of integer

**See Also**

`gen.genealogy`, `gen.mean`, `gen.min`, `gen.depth`, `gen.meangendepth`
Examples

data(geneaji)
  genJi<-gen.genealogy(geneaji)
  gen.min(genJi,c(17,26))
  gen.mean(genJi,c(17,26))
  gen.max(genJi,c(17,26))

data(genea1TPI)
  gen1TPI<-gen.genealogy(genea1TPI)
  gen.min(gen1TPI,c(18S11L18TSPII))
  gen.mean(gen1TPI,c(18S11L18TSPII))
  gen.max(gen1TPI,c(18S11L18TSPII))

---

### gen.mean

**Mean number of generations**

**Description**

Returns the average number of generations between all probands and the individuals specified. Probands are defined as the individuals without children in the genealogy.

**Usage**

```r
  gen.mean( gen, individuals)
```

**Arguments**

- `gen` An object of class GLgen obtained with `gen.genealogy`, `gen.lineages` or `gen.branching`. Required.
- `individuals` A vector of ids specifying the individuals to include in the calculation. Required.

**Value**

returns a numeric value

**See Also**

- `gen.genealogy`  
- `gen.min`  
- `gen.max`  
- `gen.depth`  
- `gen.meangendepth`

**Examples**

data(geneaji)
  genJi<-gen.genealogy(geneaji)
  gen.min(genJi,c(17,26))
  gen.mean(genJi,c(17,26))
  gen.max(genJi,c(17,26))
**Description**

Returns the expected genealogical depth.

**Usage**

```r
gen.meangendepth( gen, pro = "0", type = "MEAN", ...)
```

**Arguments**

- `gen` An object of class GLgen obtained with gen.genealogy, gen.lineages or gen.branching. Required.
- `pro` Vector of proband id numbers to be included. Default is 0, which will select all individuals without children.
- `type` If type="MEAN" (default), mean genealogical depth over all specified probands is calculated. If type="IND", mean genealogical depth is calculated for each specified proband.
- `...` Option to pass additional arguments automatically between methods. Internal use only.

**Value**

A data frame with only one numeric value when type is "MEAN". When type is "IND", the number of rows equals the number of probands specified.

**References**


**See Also**

- `gen.genealogy` `gen.occ` `genimplex` `gen.rec` `gen.completeness` `gen.gc`
**Examples**

```r
data(geneaji)
genJi<-gen.genealogy(geneaji)
gen.meangendepth(genJi)
gen.meangendepth(genJi, type="IND")

data(genea140)
gen140<-gen.genealogy(genea140)
gen.meangendepth(gen140)
gen.meangendepth(gen140, pro=c(454422, 676521, 677273, 717537, 717634, 717709, 868572))
gen.meangendepth(gen140, pro=c(454422, 676521, 677273, 717537, 717634, 717709, 868572), type="IND")
```

---

**Description**

Returns the variance of the genealogical depth

**Usage**

```r
gen.meangendepthVar( gen, pro = "0", type = "MEAN", ...)
```

**Arguments**

- `gen` An object of class GLgen obtained with gen.genealogy, gen.lineages or gen.branching. Required.
- `pro` Vector of proband id numbers to be included. Default is 0, which will select all individuals without children.
- `type` If type="MEAN" (default), the average of genealogical depth variances (over all probands) is returned. If type="IND", the variance of the genealogical depth is calculated for each specified proband.
- `...` Option to pass additional arguments automatically between methods. Internal use only.

**Value**

A data frame with only one numeric value when type is "MEAN". When type is "IND", the number of rows equals the number of probands specified.

**References**

**Description**

Returns the minimum number of generations between all probands and the individuals specified. Probands are defined as the individuals without children in the genealogy.

**Usage**

```r
gen.min( gen, individuals)
```

**Arguments**

- `gen` An object of class GLgen obtained with `gen.genealogy`, `gen.lineages` or `gen.branching`. Required.
- `individuals` A vector of ids specifying the individuals to include in the calculation. Required.

**Value**

returns a vector of integer

**See Also**

`gen.genealogy`, `gen.mean`, `gen.max`, `gen.depth`, `gen.meangendepth`
Examples
data(geneaji)
genJi<-gen.genealogy(geneaji)
gen.min(genJi,c(17,26))
gen.mean(genJi,c(17,26))
gen.max(genJi,c(17,26))

data(genea1TPI)
gen1TPI<-gen.genealogy(genea1TPI)
gen.min(gen1TPI,c(18S11L18TSPII))
gen.mean(gen1TPI,c(18S11L18TSPII))
gen.max(gen1TPI,c(18S11L18TSPII))

genochildren
Number of children

Description
Returns the number of children for specified individuals

Usage
genochildren( gen, individuals)

Arguments
  gen An object of class GLgen obtained with gen.genealogy, gen.lineages or gen.branching. Required.
  individuals A vector of ids of the individuals on each of whom the number of children is to be returned. Required.

Value
A vector containing the number of children for each individual specified.

See Also
gen.genealogy gen.nowomen gen.noind gen.nomen gen.children

Examples
data(geneaji)
genJi<-gen.genealogy(geneaji)
gen.children(genJi,14)
gen.nochildren(genJi,individuals=c(1,12,14,20))
data(genea1TPI)
gen.noind

Descrition
Returns the number of individuals included in the genealogy

Usage

```r
gen.noind( gen)
```

Arguments

- **gen**
  
  An object of class GLgen obtained with gen.genealogy, gen.lineages or gen.branching.

  Required.

Value

- returns a vector of integer

See Also

- gen.genealogy
- gen.nowomen
- gen.nochildren
- gen.nomen

Examples

```r
data(geneajI)
genJI<-gen.genealogy(geneajI)
gen.noind(genJi)
```

gen.momen

<table>
<thead>
<tr>
<th>Number of men</th>
</tr>
</thead>
</table>

Description

Returns the number of men included in the genealogy

Usage

```r
gen.momen( gen)
```
Arguments

gen An object of class GLgen obtained with gen.genealogy, gen.lineages or gen.branching. Required.

Value

returns a vector of integer

See Also

gen.genealogy gen.nowomen gen.nochildren gen.noid

Examples

data(geneajiI)
genJi<gen.genealogy(geneajiI)
gen.nomen(genJi)

table

<table>
<thead>
<tr>
<th>gen.nowomen</th>
<th>Number of women</th>
</tr>
</thead>
</table>

Description

Returns the number of women included in the genealogy

Usage

gen.nowomen( gen)

Arguments

gen An object of class GLgen obtained with gen.genealogy, gen.lineages or gen.branching. Required.

Value

returns a vector of integer

See Also

gen.genealogy gen.nomen gen.nochildren gen.noid

Examples

data(geneajiI)
genJi<gen.genealogy(geneajiI)
gen.nowomen(genJi)
Description

Returns the number of times that the specified ancestors are present in the genalogies of the specified probands.

Usage

gen.occ( gen, pro = "0", ancestors = "0", typeOcc = "IND", ...)

Arguments

gen  An object of class GLgen obtained with gen.genealogy, gen.lineages or gen.branching. Required.
pro  Vector of proband id numbers to be included. Default is 0, which will select all individuals without children.
ancestors  Vector of ancestors id numbers to be included. Default is 0, which will select all individuals without parents.
typeOcc  If typeOcc="IND" (default), the number of occurrences for each proband will be returned. If typeOcc="TOTAL", the sum of all occurrences over all probands will be returned.
...  Option to pass additional arguments automatically between methods. Internal use only.

Value

A matrix with number of lines equal to the number of ancestors included and the number of columns equal to the number of probands included if typeOcc="BRUT" or only one column if typeOcc="TOTAL"

See Also

gen.genealogy gen.rec gen.implex gen.meangendepth gen.gc

Examples

data(geneaJi)
genJi<-gen.genealogy(geneaJi)
## Not run: Number of occurrences of ancestors in the specified proband's genealogy
gen.occ(genJi, pro=c(1,29), ancestors=c(3,6,10,12,14,16,18,20,26,28))

data(genea140)
gen140<-gen.genealogy(genea140)
gen.occ(gen140, pro=c(454422,676521,677273,717537,717634,717709,868572),
ancestors=c(18305,18528,31114,18325))
Description

Returns the id numbers of the parents of specified individuals.

Usage

gen.parent( gen, individuals, output = "FaMo", ...)

Arguments

- gen: An object of class GLgen obtained with gen.genealogy, gen.lineages or gen.branching. Required.
- individuals: Vector of individual id numbers. Required.
- output: If output="FaMo" (default) then both mothers and fathers are included. "Mo" outputs mothers only and "Fa", fathers only.
- ...: Option to pass additional arguments automatically between methods. Internal use only.

Value

returns a list containing the following: Fathers Mothers

See Also

gen.genealogy gen.pro gen.founder gen.children gen.sibship

Examples

data(geneajiI)
genJi<-gen.genealogy(geneajiI)
gen.parent(genJi, individuals=c(3,21,29))
**Description**

Returns the kinship coefficients between pairs of individuals

**Usage**

```r
gen.phi(gen, pro, depthmin=(gen.depth(gen)-1), depthmax=(gen.depth(gen)-1), MT=F)
```

**Arguments**

- `gen` An object of class GLgen obtained with gen.genealogy, gen.lineages or gen.branching. Required.
- `pro` Vector of proband id numbers to be included. Required.
- `depthmin` Minimum genealogical depth to consider in the calculation. Default is the whole genealogy.
- `depthmax` Maximum genealogical depth to consider in the calculation. Default is the whole genealogy.
- `MT` Allows parallel computing when set to TRUE. Default is MT=F.

**Value**

A matrix or a GLmultiMatrix object depending on the number of generations treated. GLmultiMatrix is an array of matrices, one for each depth. Array of size (`depthMax`-'depthMin') and matrices of size `length(pro)` * `length(pro)`. The matrix object, also of size `length(pro)` * `length(pro)`, is returned when (`depthMax`-'depthMin') equals 1.

**References**


**See Also**

- `gen.genealogy`
- `gen.f`

**Examples**

```r
data(geneaji)
genJi<-gen.genealogy(geneaji)
kinship<-gen.phi(genJi)
kinship
```
kinship_allgenerations <- gen.phi(genji, depthmin = 1)
kinship_allgenerations
## Not run: 7th generations back in time is equivalent to considering all generations
kinship_allgenerations <- unclass(kinship_allgenerations)
kinship_allgenerations[, , 7] = kinship

kinship_allgenerations[1, 2, ]
## Not run: Plot of kinship varying according to number of generations considered
plot(1:7, kinship_allgenerations[1, 2, ], type = "b", xlab = "Generation", ylab = "Kinship value",
ylim = c(0, 0.6), pch = 8)
points(1:7, kinship_allgenerations[1, 3, ], type = "b", lty = 12, pch = 1)
legend("topright", legend = c("Individuals 1 and 2", "Individuals 2 and 29"), lty = c(1, 12), pch = c(0, 1))

---

### gen.phiCI

**Average kinship confidence interval**

**Description**

Returns the confidence interval of the average kinship among pairs of specified individuals

**Usage**

```r
gen.phiCI(phiMatrix, prob = c(0.025, 0.05, 0.95, 0.975), b = 5000, print.it = F)
```

**Arguments**

- `phiMatrix`: A square matrix of kinship coefficients obtained for example with `gen.phi`, or an array of square matrices (GLmultiMatrix object) of kinship coefficients obtained with `gen.phi`. Required.
- `prob`: Confidence limits probabilities. Default is `probs = c(0.025, 0.05, 0.95, 0.975)`. 
- `b`: Number of simulations used to calculate the confidence interval. Default is `b = "5000"`. 
- `print.it`: If `print.it = "F"` (default), progression in the number of simulations performed is not displayed.

**Value**

A vector or a GL.multiVector object depending on the type of `phiMatrix` entered. If `phiMatrix` is a GL.multiMatrix, an array of vectors (GL.multiVector), one for each depth, is returned. Array of size (`depthMax` - `depthMin`), vectors of size `length(probs)`. If `phiMatrix` is a matrix, a vector of size `length(probs)` is returned.

**See Also**

- `gen.genealogy`, `gen.phi`, `gen.phiOver`, `gen.phiMean`
**Examples**

```r
data(genea140)
gen140<-gen.genealogy(genea140)
ge07<-gen.branching(gen140, pro=gen.pro(gen140)[c(1:7)])
phiMat<-gen.phi(g07)
gen.phiCI(phiMat)
```

---

<table>
<thead>
<tr>
<th>gen.phiMean</th>
<th>Average kinship</th>
</tr>
</thead>
</table>

**Description**

Returns the average kinship among pairs of specified individuals.

**Usage**

```r
gen.phiMean( phiMatrix)
```

**Arguments**

- `phiMatrix`: A square matrix of kinship coefficients obtained for example with `gen.phi`. Required.

**Value**

returns a numeric value

**See Also**

`gen.genealogy gen.phi gen.phiOver gen.phiCI`

**Examples**

```r
data(geneaJi)
genJi<-gen.genealogy(geneaJi)
kinship<-gen.phi(genJi)
gen.phiMean(kinship)
```

```r
data(genea140)
gen140<-gen.genealogy(genea140)
phi6subjects<-gen.phi(gen140, pro=c(454422, 676521, 677273, 717537, 717634, 717709, 868572))
gen.phiMean(phi6subjects)
```
gen.phiOver

### Description

Returns the pairs of individuals with kinship coefficient values greater than specified threshold.

### Usage

```r
gen.phiOver( phiMatrix, threshold)
```

### Arguments

- `phiMatrix`: A square matrix of kinship coefficients obtained for example with `gen.phi`. Required.
- `threshold`: Threshold of kinship values to return.

### Value

A data frame containing the probands and their kinship.

### See Also

- `gen.genealogy`
- `gen.phi`
- `gen.phiMean`
- `gen.phiCI`

### Examples

```r
data(geneaJi)
genJi<-gen.genealogy(geneaJi)
kinship<-gen.phi(genJi)
gen.phiOver(kinship, 0.1)

data(genea140)
gen140<-gen.genealogy(genea140)
phi9subj<-gen.phi(gen140, pro=c(408758, 408950, 409082, 409111, 676521, 717537, 717634, 717709, 868572))
gen.phiOver(phi9subj, 0.025)
```
Description

Returns the id numbers of the probands. Probands are defined as the individuals without children in the genealogy.

Usage

gen.pro( gen, ...)

Arguments

gen An object of class GLgen obtained with gen.genealogy, gen.lineages or gen.branching. Required.

... Option to pass additionnal arguments automatically between methods. Internal use only.

Value

returns a vector of integer

See Also

gen.genealogy gen.founder gen.half.founder gen.parent gen.children

Examples

data(geneaJi)
genJi<-gen.genealogy(geneaJi)
## Not run: There are 3 probants
gen.pro(genJi)

data(genea140)
gen140<-gen.genealogy(genea140)
gen.pro(gen140)
## Not run: There are 140 probants
Ancestors coverage

Description

Returns the number of specified probands genealogically related to specified ancestors (i.e., ancestor occurs in the proband’s genealogy).

Usage

gen.rec( gen, pro = "0", ancestors = "0", ...)

Arguments

gen An object of class GLgen obtained with gen.genealogy, gen.lineages or gen.branching. Required.
pro Vector of proband id numbers to be included. Default is 0, which will select all individuals without children.
ancestors Vector of ancestors id numbers to be included. Default is 0, which will select all individuals without parents.
... Option to pass additional arguments automaticaly between methods. Internal use only.

Value

A matrix with number of lines equal to the number of ancestors specified and one column.

Note

If an ancestor is also a proband, he/she will be counted in his/her coverage.

See Also

gen.genealogy gen.occ gen.implex gen.meangendepth gen.gc

Examples

data(geneaJi)
genJi<-gen.genealogy(geneaJi)
## Not run: Number of probands which are descendants of an ancestor
gen.rec(genJi)

data(genea140)
gen140<-gen.genealogy(genea140)
gen.rec(gen140, ancestors=c(18305, 18528, 31114, 18325))
Description

Returns the id numbers of the siblings of specified individuals.

Usage

gen.sibship( gen, individuals, halfSibling = "T", ...)

Arguments

gen An object of class GLgen obtained with gen.genealogy, gen.lineages or gen.branching. Required.

individuals Vector of individual id numbers. Required.

halfSibling If halfSibling="T" (default) then ids of halfsiblings are also returned. halfSibling="F" returns only full sibling ids.

... Option to pass additional arguments automatically between methods. Internal use only.

Value

returns a vector of integer

See Also

gen.genealogy gen.pro gen.founder gen.children gen.parent

Examples

data(geneaJi)
genJi<-gen.genealogy(geneaJi)
gen.sibship(genJi, individuals=21, halfSibling=TRUE)

data(genea140)
gen140<-gen.genealogy(genea140)
sibs<-gen.sibship(gen140, individuals=10174, halfSibling=FALSE)
gen.graph(gen140, pro=c(10174, sibs))
Gen. dropping simulations - Probabilities

Description

Returns the probabilities that specified probands inherit disease alleles from ancestors.

Usage

```r
gen.simuProb(gen, pro, statePro, ancestors, stateAncestors, simulNo=5000,
  probRecomb=c(0,0), probSurvival=1.0)
```

Arguments

- **gen**: An object of class GLgen obtained with `gen.genealogy`, `gen.lineages` or `gen.branching`. Required.
- **pro**: Vector of proband id numbers to be included. Required.
- **statePro**: Required vector indicating, for each proband in pro, the probability to be calculated: 0 = no disease allele is transmitted 1 = 1 disease allele is transmitted 2 = 2 disease alleles are transmitted 3 = 1 or 2 disease alleles are transmitted
- **ancestors**: Vector of ancestors id numbers to be included. Required.
- **stateAncestors**: Required vector indicating, for each ancestor in ancestors, the genotype state: 0 = no disease allele present 1 = 1 disease allele present 2 = 2 disease alleles present
- **simulNo**: Number of simulations to perform. Default is 5000.
- **probRecomb**: Recombination probabilities for males and females. Default is no recombination.
- **probSurvival**: Survival probability for homozygotes. Default is 1.

Value

A list containing the following: the joint probability of specified statePro for all probands, the probability of specified statePro for each proband, the probability that, 0, 1, ..., and all probands inherit the specified number of disease alleles.

See Also

- `gen.genealogy`
- `gen.simuSample`
- `gen.simuSet`
- `gen.simuSampleFreq`

Examples

```r
data(geneaji)
genJi<-gen.genealogy(geneaji)
# Not run: Probability that subjects 1 and 29 get 1 and 2 alleles from ancestors 20 and 25,
# Not run: that have themselves 2 and 1.
gen.simuProb(genJi, pro=c(1,29), statePro=c(1,2), ancestors=c(20,25), stateAncestors=c(2,1),
```
### gen.simuSample

#### Description

Returns the number of alleles transmitted to specified probands from ancestors.

#### Usage

```r
gen.simuSample(gen, pro, ancestors, stateAncestors, simulNo = 5000)
```

#### Arguments

- **gen**: An object of class GLgen obtained with `gen.genealogy`, `gen.lineages` or `gen.branching`. Required.
- **pro**: Vector of proband id numbers to be included. Required.
- **ancestors**: Vector of ancestors id numbers to be included. Required.
- **stateAncestors**: Required vector indicating, for each ancestor in ancestors, the genotype state: 0 = no disease allele present, 1 = 1 disease allele present, 2 = 2 disease alleles present.
- **simulNo**: Number of simulations to perform. Default is 5000.

#### Value

A matrix with number of columns equal to the number of simulations and number of rows equal to the number of probands.

#### See Also

`gen.genealogy`, `gen.simuProb`, `gen.simuSet`, `gen.simuSampleFreq`
Examples

data(geneaji)
genji<-gene.genealogy(geneaji)
simu_1000<-gene.simuSample(genji, pro=c(1,29), ancestors=c(20,25), stateAncestors=c(2,1), simulNo=1000)
## Not run: Number of alleles received by probants
table(simu_1000)
## Not run: Number of alleles received by simulation
table(colSums(simu_1000))
## Not run: Number of alleles received by each probant
table(simu_1000[1,], simu_1000[2,])

data(geneaji)
genji<-gene.genealogy(geneaji)
simu_5000<-gene.simuSample(genji, pro=c(1,29), ancestors=c(20,25), stateAncestors=c(2,1), simulNo=5000)
## Not run: Number of alleles received by probants
table(simu_5000)
## Not run: Number of alleles received by simulation
table(colSums(simu_5000))
## Not run: Number of alleles received by each probant
table(simu_5000[1,], simu_5000[2,])

gen.simuSampleFreq  Gene dropping simulations - Frequencies

Description

Returns the number of alleles transmitted to specified probands from ancestors

Usage

gen.simuSampleFreq( gen, pro, ancestors, stateAncestors, simulNo = "5000")

Arguments

gen  An object of class GLgen obtained with gen.genealogy, gen.lineages or gen.branching. Required.
pro   Vector of proband id numbers to be included. Required.
ancestors   Vector of ancestors id numbers to be included. Required.
stateAncestors Required vector indicating, for each ancestor in ancestors, the genotype state: 0 = no disease allele present 1 = 1 disease allele present 2 = 2 disease alleles present
simulNo Number of simulations to perform. Default is 5000.
gen.simuSet

Value
A data frame containing for each row (one row per proband): **Alleles.transmitted.0 Alleles.transmitted.1 Alleles.transmitted.2**

See Also
*gen.genealogy gen.simuProb gen.simuSet gen.simuSample*

Examples
```
data(geneaji)
genji<-gen.genealogy(geneaji)
gen.simuSampleFreq(genji, pro=c(1,29), ancestors=c(20,25),stateAncestors=c(2,1), simulNo = 1000)
```

---

gen.simuSet  

*Gene dropping simulations with specified transmission probabilities*

Description
Returns the number of alleles transmitted to specified probands from ancestors considering specified transmission probabilities.

Usage
```
gen.simuSet(gen, pro, ancestors, stateAncestors, probMatrix=matrix(c(
c(1,0.5,0,0.5,0.25,0,0,0,0,1,1,1,1,0.75,0.5,1,0.5,0),
c(1,0.5,0,0.5,0.25,0,0,0,0,1,1,1,1,0.75,0.5,1,0.5,0)),
   nrow=3, ncol=12),
simulNo = 5000)
```

Arguments
- **gen** An object of class GLgen obtained with gen.genealogy, gen.lineages or gen.branching. Required.
- **pro** Vector of proband id numbers to be included. Required.
- **ancestors** Vector of ancestors id numbers to be included. Required.
- **stateAncestors** Required vector indicating, for each ancestor in ancestors, the genotype state: 0 = no disease allele present 1 = 1 disease allele present 2 = 2 disease alleles present
- **probMatrix** Matrix of transmission probabilities of 0, 1, or 2 disease alleles. Default is: [.1] [.2] [.3] [.4] [.5] [.6] [.7] [.8] [.9] [.10] [.11] [.12] [1] 1.0 0.50 0 1 1.00 1.0 1.0 0.50 0 1 1.00 1.0 [2] 0.5 0.25 0 1 0.75 0.5 0.25 0 1 0.75 0.5 0.25 0 1 0.50 0.0 0.0 0.0 0.0 0.0 0.0 where rows are genotype states (0,1,2) for the father and columns are states for the mother (in blocks of 3). 1st block (col
1-3) are the probabilities of transmitting 0 allele to a son. 2nd block (col 4-6) are the probabilities of transmitting 1 allele to a son. 3rd block (col 7-9) are the probabilities of transmitting 0 allele to a daughter. 4th block (col 10-12) are the probabilities of transmitting 1 allele to a daughter.

**Value**

A matrix with number of columns equal to the number of simulations and number of rows equal to the number of probands.

**See Also**

`gen.genealogy` `gen.simuProb` `gen.simuSample`

**Examples**

```r
## Not run: A case where only male subjects can receive alleles
data(geneaji)
genJi<-gen.genealogy(geneaji)
onlyThroughMale<-matrix(c(c(1,0.5,0.5,0,0.25,0,0,0,0,1,1,1,0.75,0.25,0,0.5,1,0.5,0), rep(1,18)),
nrow=3, ncol=12)
gen.graph(genJi, indVarAffected=c(28,27,25,17,12,8,4,1), varAffected=c(28,27,25,17,12,8,4,1))
simu_1000a<-gen.simuSet(genJi, pro=c(1,28), ancestors=c(1,28), stateAncestors=c(1),
simulNo = 1000, probMatrix=onlyThroughMale)
## Not run: Number of alleles received by probants
table(simu_1000a)
## Not run: Number of alleles received by all probants at each simulation
table(colSums(simu_1000a))
## Not run: Number of alleles received by each probant
table(simu_1000a[1,], simu_1000a[2,])

## Not run: A case where subjects are limited to one copy compared to what is normally expected
max1allreceived<-matrix(rep(c(1,0.5,0,0.5,0.25,0,0,0,0,1,1,1,1,1), 2), nrow=3, ncol=12)
gen.graph(genJi)
simu_1000b<-gen.simuSet(genJi, pro=c(1,29), ancestors=25, stateAncestors=1, simulNo=10000,
probMatrix=max1allreceived)
## Not run: Normal case matrix not changed
simu_1000original<-gen.simuSet(genJi, pro=c(1,29), ancestors=25, stateAncestors=1, simulNo=10000)
## Not run: Number of alleles received by each probant
table(simu_1000b)
table(simu_1000original)
## Not run: Number of alleles received by all probants at each simulation
table(colSums(simu_1000b))
table(colSums(simu_1000original))
## Not run: Number of alleles received by each probant
table(simu_1000b[1,], simu_1000b[2,])
table(simu_1000original[1,], simu_1000original[2,])
```
genea140  Genealogical information for 140 individuals from the Quebec Reference Sample

Description
A genealogical corpus made of 41523 individuals from the province of Quebec, Canada. A total of 140 individuals have been sampled in seven sub-populations, listed in pop140, and their genealogies were reconstructed as far back as possible using the BALSAC population register and the Early Quebec Population Register.

Usage
genea140

Format
A data frame with 41523 observations on 11 variables. 
[,1] ind: An individual’s ID number 
[,2] father: This individual’s father ID number 
[,3] mother: This individual’s mother ID number 
[,4] sex: Individual’s sex coded 1/2 for male/female

Source
Balsac

References

geneaji  Highly inbred pedigree

Description

Usage
geneaji

Format
A data frame with 29 observations on 4 variables. 
[, 1] ind An individual’s ID number 
[, 2] father This individual’s father ID number 
[, 3] mother This individual’s mother ID number 
[, 4] sex Individual’s sex coded 1/2 for male/female
Source

CHAPMAN & JACQUART (1971)

References


---

**GLgen-class**  
*Class "GLgen"*

Description

Object containing a genealogy.

Objects from the Class

Objects can be created by calls of the form `genNgenealogy('data.frame')`.

Extends

Class "GLgroupe", directly.

Methods

- **initialize** signature(.Data="integer",Date="character"): ...
- **depth** signature(x = "GLgen") Gives the depth of the genealogy.
- **length** signature(x = "GLgen") Gives the number of individuals in the genealogy.

Examples

```
showClass("GLgen")
```

---

**GLgroupe-class**  
*Class "GLgroupe"*

Description

Object representing a set of proband in different groups. Each element of this list is a group named with the proband number.

Objects from the Class

Objects can be created by calls of the form `new("GLgroupe", 'list')`.

---
GLmultiList-class

Extends

Class "GLgen", directly.

Methods

initialize signature(.Data=list): ... 
signature(object = "GLgroupe", ANY,ANY,ANY): ...

Examples

showClass("GLgroupe")

GLmultiList-class  Class "GLmultiList"

Description

Objects created to carry information mostly between invisible functions.

Objects from the Class

Objects can be created by calls of the form new("GLmultiList", 'Array').

Slots

.Data: Object of class "list" ~~
liste: Object of class "list" ~~

Extends

Class "list", from data part. Class "vector", by class "list", distance 2.

Methods

No methods defined with class "GLmultiList" in the signature.

Examples

showClass("GLmultiList")
Description

The 140 individuals from the genealogical corpus from Quebec were sampled from 7 different populations from 5 regions; Quebec City, Montreal, Saguenay, North Shore, Gaspesia. In Gaspesia we find 3 different populations: French-Canadians, Acadians and Loyalists.

Usage

pop140

Format

, 1 ind: An individual’s ID number
, [ 2] pop: This individual’s population

Source

Balsac

References

http://www.quebecgenpop.ca/home.html

## Index

**Topic Utilities**
- gen.nomen, 35
**Topic aplot**
- gen.graph, 23
**Topic attribute**
- gen.depth, 11
**Topic classes**
- Classes of basic handling of genealogy data, 3
  - Classes of GLgroupe handling, 4
    - GLgen-class, 52
    - GLgroupe-class, 52
    - GLmultiList-class, 53
**Topic datagen**
- gen.simuProb, 46
  - gen.simuSample, 47
  - gen.simuSampleFreq, 48
  - gen.simuSet, 49
**Topic datasets**
- genea140, 51
  - geneaji, 51
  - pop140, 54
**Topic manip**
- gen.branching, 6
  - gen.children, 7
  - gen.completeness, 8
  - gen.completenessVar, 9
  - gen.f, 11
  - gen.fCI, 13
  - gen.find.Min.Distance.MRCA, 14
  - gen.findDistance, 15
  - gen.findFounders, 16
  - gen.findMRCA, 17
  - gen.founder, 18
  - gen.gc, 19
  - gen.genealogy, 20
  - gen.genout, 21
  - gen.half.founder, 25
  - gen.implex, 26
  - gen.implexVar, 27
  - gen.lineages, 28
  - gen.max, 29
  - gen.mean, 30
  - gen.meangendepth, 31
  - gen.meangendepthVar, 32
  - gen.min, 33
  - gen.nochildren, 34
  - gen.occ, 37
  - gen.parent, 38
  - gen.phi, 39
  - gen.phiCI, 40
  - gen.phiMean, 41
  - gen.phiOver, 42
  - gen.pro, 43
  - gen.rec, 44
  - gen.sibship, 45
**Topic methods**
- gen.climbPAR, 8
  - gen.getAncestorsPAR, 22
  - gen.getFoundersPAR, 23
**Topic package**
- GenlibR-package, 3
**Topic utilities**
- gen.noind, 35
  - gen.nowomen, 36
array, 4, 5

Classes of basic handling of genealogy data, 3
Classes of GLgroupe handling, 4
depth, GLgen-method (GLgen-class), 52
depth, GLmultiArray4-method (Classes of basic handling of genealogy data), 3
depth, GLmultiMatrix-method (Classes of basic handling of genealogy data), 3
depth, GLmultiNumber-method (Classes of basic handling of genealogy data), 3
depth, GLmultiPhiGroup-method (Classes of basic handling of genealogy data), 3
depth, GLmultiPhiGroupSingle-method (Classes of GLgroupe handling), 4
depth, GLmultiVector-method (Classes of basic handling of genealogy data), 3
Dim, GLmultiArray4-method (Classes of basic handling of genealogy data), 3
Dim, GLmultiFGroup-method (Classes of GLgroupe handling), 4
Dim, GLmultiFGroupSingle-method (Classes of GLgroupe handling), 4
Dim, GLmultiMatrix-method (Classes of basic handling of genealogy data), 3
Dim, GLmultiPhiGroup-method (Classes of GLgroupe handling), 4
Dim, GLmultiPhiGroupSingle-method (Classes of GLgroupe handling), 4
Dim, GLmultiVector-method (Classes of basic handling of genealogy data), 3
gen.branching, 6, 21, 24, 29
gen.children, 7, 34, 38, 43, 45
gen.climbPAR, 8, 22
gen.completeness, 8, 19, 26, 31
gen.completenessVar, 9
gen.depth, 11, 29, 30, 33
gen.f, 11, 13, 39
gen.fCI, 13
gen.find.Min.Distance.MRCA, 14, 15–17
gen.findDistance, 14, 15, 16, 17, 23
gen.findFounders, 14, 15, 16, 17, 23
gen.findMRCA, 8, 14–16, 17, 22
gen.founder, 7, 14–17, 18, 25, 38, 43, 45
INDEX

gen.gc, 10, 19, 26, 28, 31, 33, 37, 44

gen.genealogy, 6, 7, 9, 11, 12, 14–19, 20, 21, 24–26, 29–31, 33–47, 49, 50

gen.genout, 21, 24, 29

gen.getAncestorsSPAR, 8, 22

gen.getFoundersSPAR, 23

gen.graph, 6, 21, 23, 29

gen.half_founder, 18, 25, 43

gen.implex, 9, 19, 26, 31, 37, 44

gen.implexVar, 27

gen.lineages, 6, 21, 28

gen.max, 29, 30, 33

gen.mean, 11, 29, 30, 33

gen.meangendepth, 9, 11, 19, 26, 29, 30, 31, 33, 37, 44

gen.meangendepthVar, 32

gen.min, 11, 29, 30, 33

gen.nochildren, 34, 35, 36

gen.noid, 34, 35, 36

gen.omen, 34, 35, 36

gen.nowomen, 34–36, 36

gen.occ, 9, 19, 26, 31, 37, 44

gen.parent, 7, 18, 25, 38, 43, 45

gen.phi, 12, 39, 40–42

gen.phiCI, 40, 41, 42

gen.phimean, 40, 41, 42

gen.phiover, 40, 41, 42

gen.pro, 7, 18, 25, 38, 43, 45

gen.rec, 9, 19, 26, 31, 37, 44

gen.sibship, 7, 38, 45

gen.simuProb, 46, 47, 49, 50

gen.simuSample, 46, 47, 49, 50

gen.simuSampleFreq, 46, 47, 48

gen.simuSet, 46, 47, 49, 49

genea1TP, 51

geneaji, 51

GenlibR-package, 3

GLCGMatrixGroupSingle-class (Classes of Glgroupe handling), 4

GLgen, 53
GLgen-class, 52
GLgroupe, 52
GLgroupe-class, 52

GLmultiArray4-class (Classes of basic handling of genealogy data), 3

GLmultiFGroup-class (Classes of Glgroupe handling), 4

GLmultiFGroupSingle-class (Classes of