Package ‘paramlink’

July 23, 2025

Title Parametric Linkage and Other Pedigree Analysis in R

Version 1.1-6

Maintainer Magnus Dehli Vigeland <m.d.vigeland@medisin.uio.no>

Description NOTE: 'PARAMLINK' HAS BEEN SUPERSEDED BY THE 'PEDSUITE' PACKAGES (<https://magnusdv.github.io/pedsuite/>). 'PARAMLINK' IS MAINTAINED ONLY FOR LEGACY PURPOSES AND SHOULD NOT BE USED IN NEW PROJECTS. A suite of tools for analysing pedigrees with marker data, including parametric linkage analysis, forensic computations, relatedness analysis and marker simulations. The core of the package is an implementation of the Elston-Stewart algorithm for pedigree likelihoods, extended to allow mutations as well as complex inbreeding. Features for linkage analysis include singlepoint LOD scores, power analysis, and multipoint analysis (the latter through a wrapper to the 'MERLIN' software). Forensic applications include exclusion probabilities, genotype distributions and conditional simulations. Data from the 'Familias' software can be imported and analysed in 'paramlink'. Finally, 'paramlink' offers many utility functions for creating, manipulating and plotting pedigrees with or without marker data (the actual plotting is done by the 'kinship2' package).

License GPL (>= 2)

URL https://github.com/magnusdv/paramlink

BugReports https://github.com/magnusdv/paramlink/issues

Depends R (>= 3.3)

Imports assertthat, graphics, kinship2, maxLik, stats, utils

Suggests igraph

Encoding UTF-8

Language en-GB

LazyData Yes

RoxygenNote 7.3.2

SystemRequirements For multipoint linkage analysis, 'MERLIN' (http://csg.sph.umich.edu/abecasis/merlin/index.html).

NeedsCompilation no

Author Magnus Dehli Vigeland [aut, cre],
Thore Egeland [ctb],
Guro Doerum [ctb]

Repository CRAN

Date/Publication 2025-07-21 20:21:05 UTC

Contents

`allGenotypes` *Genotype combinations*

Description

Auxiliary functions computing possible genotype combinations in a pedigree. These are not normally intended for end users.

Usage

```
allGenotypes(n)

fast.grid(argslist, as.list = FALSE)

geno.grid.subset(x, partialmarker, ids, chrom, make.grid = T)
```

Arguments

`n`	a positive integer.
`argslist`	a list of vectors.
`as.list`	if TRUE, the output is a list, otherwise a matrix.
`x`	a `linkdat` object.
`partialmarker`	a `marker` object compatible with `x`.
`ids`	a numeric with ID labels of one or more pedigree members.
`chrom`	a character, either 'X' or 'AUTOSOMAL'. If missing, the 'chrom' attribute of `partialmarker` is used. If this is also missing, then 'AUTOSOMAL' is taken as the default value.
`make.grid`	a logical. If FALSE, a list is returned, otherwise `fast.grid` is applied to the list before returning it.

Value

`allGenotypes` returns a matrix with 2 columns and `n + n*n(n-1)/2` rows containing all possible (unordered) genotypes at a biallelic locus with alleles `1,2,...{},n`. `fast.grid` is basically a stripped down version of `expand.grid`.

Examples

```
m = allGenotypes(2)
stopifnot(m == rbind(c(1,1), c(2,2), 1:2))
```

---

as.data.frame.linkdat *linkdat to data.frame conversion*

Description

Convert a linkdat object to data.frame for pretty printing.

Usage

```
## S3 method for class 'linkdat'
as.data.frame(
  x,
  ...,
  famid = F,
  markers = seq_len(x$nMark),
  alleles = NULL,
  missing = NULL,
  singleCol = FALSE,
  sep = ""
)
```

Arguments

`x`	a `linkdat` object.
`...`	further arguments (not used).
`famid`	a logical indicating if the family identifier should be included as the first column.
`markers`	a numeric indicating which markers should be included/printed.
`alleles`	a character containing allele names, e.g. `alleles=c('A','B')`.
`missing`	the character (of length 1) used for missing alleles. Defaults to '0'.
`singleCol`	a logical: Should the two alleles for each marker be pasted into one column or kept in separate columns?
`sep`	a single character to be used as allele separator if `singleCol=TRUE`.

Details

This function is mainly intended for pretty-printing `linkdat` objects (for instance it is called by `print.linkdat`). For direct manipulation of the pedigree and/or marker matrices, it is better to use `as.matrix.linkdat`.

Value

A `data.frame`.

See Also

`as.matrix.linkdat`

Examples

```
x = linkdat(toyped)
x

# Printing x as above is equivalent to:
as.data.frame(x, sep = '/', missing = '-', singleCol = TRUE)
```

`as.matrix.linkdat` *linkdat to matrix conversion*

Description

Converts a `linkdat` object to a matrix (basically following a pre-makeped LINKAGE format), with marker annotations and other info attached as attributes.

Usage

```
## S3 method for class 'linkdat'
as.matrix(x, include.attrs = TRUE, ...)

restore_linkdat(x, attrs = NULL, checkped = TRUE)
```

Arguments

`x`	a `linkdat` object. In `restore_linkdat`: A numerical matrix in LINKAGE format.
`include.attrs`	a logical indicating if marker annotations and other info should be attached as attributes. See value.
`...`	not used.
`attrs`	a list containing marker annotations and other `linkdat` info compatible with `x`, in the format produced by `as.matrix`. If NULL, the attributes of `x` itself are used.
`checkped`	a logical, forwarded to `linkdat`. If FALSE, no checks for pedigree errors are performed.

Details

restore_linkdat is the reverse of as.matrix.

The way linkdat objects are created in paramlink, marker data are stored as a list of marker objects. Each of these is essentially a matrix with various attributes like allele frequencies, map info a.s.o.. This format works well for marker-by-marker operations (e.g. likelihoods and LOD scores), but makes it somewhat awkward to operate 'horizontally', i.e. individual-by-individual, for instance if one wants to delete all genotypes of a certain individual, or rearrange the pedigree in some way.

It is therefore recommended to convert the linkdat object to a matrix first, do the necessary manipulations on the matrix, and finally use restore_linkdat. Attributes are often deleted during matrix manipulation, so it may be necessary to store them in a variable and feed them manually to restore_linkdat using the attrs argument.

With default parameters, restore_linkdat(as.matrix(x)) should reproduce x exactly.

Value

For as.matrix: A matrix with x$nInd rows and 6 + 2*x$nMark columns. The 6 first columns describe the pedigree in LINKAGE format, and the remaining columns contain marker alleles, using the internal (numerical) allele coding and 0 for missing alleles. If include.attrs = TRUE the matrix has the following attributes:

- markerattr (a list of marker annotations)
- available (the availability vector)
- model (the disease model, if present)
- plot.labels (plot labels, if present)
- orig.ids (original individual IDs)

For restore_linkdat: A linkdat object.

See Also

linkdat, as.data.frame.linkdat

Examples

```
x = linkdat(toyped, model=1)
y = restore_linkdat(as.matrix(x))
stopifnot(all.equal(x,y))

# If attributes are lost during matrix manipulation: Use the 'attrs' argument.
xmatr = as.matrix(x)
newmatr = xmatr[-4, ] # NB: attributes are lost here
z = restore_linkdat(newmatr, attrs = attributes(xmatr))

# Should be the same as:
z2 = removeIndividuals(x, 4)
stopifnot(all.equal(z, z2))
```

`dominant` *Example pedigree for linkage analysis*

Description

Medical pedigree with 23 individuals of which 15 are genotyped with 650 SNP markers. Eleven family members are affected by a disease, showing an autosomal dominant inheritance pattern.

Usage

```
dominant
```

Format

A data frame with 23 rows and 1306 columns, describing the pedigree and marker data in pre-makeped format. The first 6 columns contain the pedigree structure and affection status, while the final 1300 columns hold the marker alleles.

- FAMID. Family ID
- ID. Individual ID
- FID. Father ID
- MID. Mother ID
- SEX. Gender (male=1, female=2)
- AFF. Affection status (unaffected=1, affected=2, unknown=0)
- M1_1. First allele of marker 1
- M1_2. Second allele of marker 1
- ...
- M650_1. First allele of marker 650
- M650_2. Second allele of marker 650

All markers are SNPs, whose alleles are written as 1 and 2. Missing alleles are denoted by 0.

Examples

```
x = linkdat(dominant)
summary(x)
```

`examineKinships` *Check pedigree for relationship errors*

Description

This function provides a convenient way to check for pedigree errors in a linkage project or other situations where marker data is available for several members. The function calls `IBDestimate` to estimate IBD coefficients for all indicated pairs of pedigree members and produces a colour-coded plot where wrong relationships are easy to spot.

Usage

```
examineKinships(
  x,
  who = "all",
  interfam = c("founders", "none", "all"),
  makeplot = T,
  pch = 4,
  ...
)
```

Arguments

`x`	A `linkdat` object, or a list of such.
`who`	A character vector of one or more of the words 'parents', 'siblings', 'hugs' (= halfsibs/uncles/grandparents), 'cousins' and 'unrelated'. Two additional single-word values are possible: 'all' (all of the above, plus 'other') and 'close' (= 'parents', 'siblings', 'hugs', 'cousins').
`interfam`	A character; either 'founders', 'none' or 'all', indicating which interfamiliar pairs of individuals should be included. Only relevant if `x` is a list of several `linkdat` objects.
`makeplot`	A logical.
`pch`	Plotting symbol (default: cross).
`...`	Other plot arguments passed on to `showInTriangle`.

Value

A list of data.frames (one for each relation category) with IBD estimates.

See Also

`IBDestimate`, `IBDtriangle`, `showInTriangle`

Examples

```
x = cousinsPed(1)
x = simpleSim(x, 500, alleles=1:2)
examineKinships(x)

# Pretend we didn't know the brothers (3 and 6) were related
x1 = branch(x, 3)
x2 = branch(x, 6)
x2$famid = 2

# Notice the error: An 'unrelated' dot close to the sibling point
examineKinships(list(x1, x2))
```

---

exclusionPower *Power of exclusion*

Description

Computes the power (of a single marker) of excluding a claimed relationship, given the true relationship.

Usage

```
exclusionPower(
  ped_claim,
  ped_true,
  ids,
  markerindex = NULL,
  alleles = NULL,
  afreq = NULL,
  known_genotypes = list(),
  Xchrom = FALSE,
  plot = TRUE
)
```

Arguments

`ped_claim`	a `linkdat` object, or a list of several linkdat and/or singleton objects, describing the claimed relationship. If a list, the sets of ID labels must be disjoint, that is, all ID labels must be unique.
`ped_true`	a `linkdat` object, or a list of several linkdat and/or singleton objects, describing the true relationship. ID labels must be consistent with `ped_claim`.
`ids`	individuals available for genotyping.
`markerindex`	NULL, or a single numeric indicating the index of a marker of `ped_claim` from which `alleles`, `afreq` and `known_genotypes` will be extracted.

`alleles`	a numeric or character vector containing marker alleles names. Ignored if `markerindex` is non-NULL.
`afreq`	a numerical vector with allele frequencies. An error is given if they don't sum to 1 (rounded to 3 decimals). Ignored if `markerindex` is non-NULL.
`known_genotypes`	list of triplets `(a, b, c)`, indicating that individual `a` has genotype `b/c`. Must be NULL if `markerindex` is non-NULL.
`Xchrom`	a logical: Is the marker on the X chromosome? Ignored if `markerindex` is non-NULL.
`plot`	either a logical or the character 'plot_only', controlling if a plot should be produced. If 'plot_only', a plot is drawn, but no further computations are done.

Details

This function computes the 'Power of exclusion', as defined and discussed in (Egeland et al., 2014).

Value

A single numeric value. If `plot='plot_only'`, the function returns NULL after producing the plot.

References

T. Egeland, N. Pinto and M. D. Vigeland, *A general approach to power calculation for relationship testing.* Forensic Science International: Genetics 9 (2014): 186-190. DOI:10.1016/j.fsigen.2013.05.001

Examples

```
############################################
### A standard case paternity case:
### Compute the power of exclusion when the claimed father is in fact unrelated to the child.
############################################

claim = nuclearPed(noffs=1, sex=2)     # Specifies individual 1 as the father of 3
true = list(singleton(id=1,sex=1), singleton(id=3, sex=2))    # Specifies 1 and 3 as unrelated
available = c(1, 3)     # Individuals 1 and 3 are available for genotyping

# Equifrequent autosomal SNP:
PE1 = exclusionPower(claim, true, available, alleles = 2, afreq=c(0.5,0.5))

# If the child is known to have genotype 1/1:
PE2 = exclusionPower(claim, true, available, alleles = 2, afreq=c(0.5,0.5),
                     known_genotypes=list(c(3,1,1)))

# Equifrequent SNP on the X chromosome:
PE3 = exclusionPower(claim, true, available, alleles = 2, afreq=c(0.5,0.5), Xchrom=TRUE)

stopifnot(PE1==0.125, PE2==0.25, PE3==0.25)

############################################
### Example from Egeland et al. (2012):
```

```
### Two females claim to be mother and daughter. Below we compute the power of various
### markers to reject this claim if they in reality are sisters.
############################################

mother_daughter = nuclearPed(1, sex = 2)
sisters = relabel(nuclearPed(2, sex = c(2, 2)), c(101, 102, 2, 3))

# Equifrequent SNP:
PE1 = exclusionPower(ped_claim = mother_daughter, ped_true = sisters, ids = c(2, 3),
                     alleles = 2)

# SNP with MAF = 0.1:
PE2 = exclusionPower(ped_claim = mother_daughter, ped_true = sisters, ids = c(2, 3),
                     alleles = 2, afreq=c(0.9, 0.1))

# Equifrequent tetra-allelic marker:
PE3 = exclusionPower(ped_claim = mother_daughter, ped_true = sisters, ids = c(2, 3),
                     alleles = 4)

# Tetra-allelic marker with one major allele:
PE4 = exclusionPower(ped_claim = mother_daughter, ped_true = sisters, ids = c(2, 3),
                     alleles = 4, afreq=c(0.7, 0.1, 0.1, 0.1))

stopifnot(round(c(PE1,PE2,PE3,PE4), 5) == c(0.03125, 0.00405, 0.08203, 0.03090))

####### How does the power change if the true pedigree is inbred?
sisters_LOOP = addParents(sisters, 101, father = 201, mother = 202)
sisters_LOOP = addParents(sisters_LOOP, 102, father = 201, mother = 203)

# Equifrequent SNP:
PE5 = exclusionPower(ped_claim = mother_daughter, ped_true = sisters_LOOP,
                     ids = c(2, 3), alleles = 2)

# SNP with MAF = 0.1:
PE6 = exclusionPower(ped_claim = mother_daughter, ped_true = sisters_LOOP,
                     ids = c(2, 3), alleles = 2, afreq=c(0.9, 0.1))

stopifnot(round(c(PE5,PE6), 5) == c(0.03125, 0.00765))

## Not run:
# Equifrequent tetra-allelic marker:
PE7 = exclusionPower(ped_claim = mother_daughter, ped_true = sisters_LOOP,
                     ids = c(2, 3), alleles = 4)

# Tetra-allelic marker with one major allele:
PE8 = exclusionPower(ped_claim = mother_daughter, ped_true = sisters_LOOP,
                     ids = c(2, 3), alleles = 4, afreq=c(0.7, 0.1, 0.1, 0.1))

stopifnot(round(c(PE7,PE8), 5) == c(0.07617, 0.03457))

## End(Not run)
```

`Familias2linkdat` *Convert 'Familias' output to linkdat objects*

Description

Familias is a widely used program for computations in forensic genetics. The function documented here facilitates the use of `paramlink` for specialized computations which are not implemented in Familias, e.g. conditional simulations.

Usage

```
Familias2linkdat(familiasped, datamatrix, loci)

readFamiliasLoci(loci)

connectedComponents(ID, FID, MID)
```

Arguments

`familiasped`	A `FamiliasPedigree` object or a list of such.
`datamatrix`	A data frame with two columns per marker (one for each allele) and one row per individual.
`loci`	A `FamiliasLocus` object or a list of such.
`ID`	An integer vector: Individual ID.
`FID`	An integer vector: ID of father.
`MID`	An integer vector: ID of mother.

Details

The Familias program represents pedigrees and marker data in a way that differs from `paramlink` in several ways, mostly because of `paramlink`'s stricter definition of a 'pedigree'. In `paramlink`, a pedigree must be connected, have numerical IDs, and each member must have either 0 or 2 parents present in the pedigree. None of this is required by `FamiliasPedigree` objects. The conversion function `Familias2linkdat` takes care of all of these potential differences: It converts each `FamiliasPedigree` into a list of connected linkdat objects, additional parents are added where needed, and non-numerical ID labels are stored in the `plot.labels` slot of the linkdat object(s).

Value

A `linkdat` object, or a list of such.

Author(s)

Magnus Dehli Vigeland, Thore Egeland

References

Windows Familias is freely available from https://familias.name.

Examples

```
x = nuclearPed(1)
```

---

hasCA *Pairwise common ancestors*

Description

Computes a matrix A whose entry A[i,j] is TRUE if pedigree members i and j have a common ancestor, and FALSE otherwise.

Usage

```
hasCA(x)
```

Arguments

x	a linkdat object.

Examples

```
x = fullSibMating(3)
A = hasCA(x)
stopifnot(A[1,1], !A[1,2], all(A[3:8, 3:8]))
```

IBDestimate *Relatedness estimation*

Description

Estimate the pairwise IBD coefficients $(\kappa_0, \kappa_1, \kappa_2)$ for specified pairs of pedigree members, using maximum likelihood methods. The optimization machinery is imported from the `maxLik` package.

Usage

```
IBDestimate(x, ids, markers = NULL, start = c(0.99, 0.001), tol = 1e-07)
```

Arguments

`x`	A single `linkdat` object or a list of `linkdat` and/or `singleton` objects.
`ids`	Either a vector of length 2, or a matrix with two columns, indicating the the pair(s) of individuals for which IBD estimates should be computed. If a matrix, each row corresponds to a pair. The entries can be either characters (matching the `plot.labels` of the linkdat object(s)) or integers (matching the `orig.ids` identifiers of the linkdat object(s)).
`markers`	A numeric indicating which marker(s) to include. If NULL (default), all markers are used.
`start`	Numeric of length 2, indicating the initial value of (κ_0, κ_2) in the optimisation (passed on to `maxLik`).
`tol`	A single numeric: the optimising tolerance value; passed on to `maxLik`).

Details

This function optimises the log-likelihood function first described in (Thompson, 1975). Optimisation is done in the (κ_0, κ_2)-plane and restricted to the probability triangle defined by $\kappa_0 \geq 0, \kappa_2 \geq 0, \kappa_0 + \kappa_2 \leq 1$.

Value

A data.frame with 8 columns: ID1, ID2 (numeric IDs), Name1, Name2 (plot labels, if present), N (#markers with no missing alleles), κ_0, κ_1, κ_2.

References

E. A. Thompson (2000). *Statistical Inferences from Genetic Data on Pedigrees.* NSF-CBMS Regional Conference Series in Probability and Statistics. Volume 6.

See Also

`examineKinships`, `IBDtriangle`, `maxLik`

Examples

```
if (requireNamespace("maxLik", quietly = TRUE)) {

# Simulate marker data for two siblings
x = nuclearPed(2)
x = setPlotLabels(x, c("Sib1", "Sib2"), c(3,4))
x = simpleSim(x, 200, 1:2) # 200 equifrequent SNPs

# Estimate the IBD coefficients for the siblings
est1 = IBDestimate(x, ids=c(3,4))

# Estimate should be the same if pedigree structure is unknown
xlist = list(branch(x, 3), branch(x, 4))
est2 = IBDestimate(xlist, ids=c(3,4))
stopifnot(identical(est1, est2))
# If the pedigree has plot.labels, they can be used as IDs
est3 = IBDestimate(x, ids=c("Sib1", "Sib2"))
stopifnot(identical(est1, est3))

}
```

IBDtriangle *IBD triangle plot*

Description

The IBD triangle is typically used to visualize the pairwise relatedness of non-inbred individuals. Various annotations are available, including points marking the most common relationships, contour lines for the kinship coefficients, and shading of the unattainable region.

Usage

```
IBDtriangle(
  relationships = c("UN", "PO", "MZ", "S", "H,U,G", "FC", "SC", "DFC", "Q"),
  kinship.lines = numeric(),
  shading = "lightgray",
  pch = 16,
  cex_points = 1.2,
  cex_text = 1,
  axes = FALSE
)
```

Arguments

`relationships`	A character vector indicating relationships points to be included in the plot. By default all of the following are included: UN=unrelated; PO=parent/offspring; MZ=monozygotic twins; S=full siblings; H=half siblings; U=uncle/niece and similar; G=grandparent/grandchild; FC=first cousins; SC=second cousins; DFC=double first cousins; Q=quadruple first half cousins.
`kinship.lines`	A numeric vector. (See Details.)
`shading`	The shading colour for the unattainable region.
`pch`	Symbol used for the relationship points (see `par`).
`cex_points`	A single numeric controlling the symbol size for the relationship points.
`cex_text`	A single numeric controlling the font size for the relationship labels.
`axes`	Draw surrounding axis box?

Details

For any pair of non-inbred individuals A and B, their genetic relationship can be summarized by the IBD coefficients $(\kappa_0, \kappa_1, \kappa_2)$, where

$$\kappa_i = P(AandBshareiallelesIBDatrandomautosomallocus).$$

Since $\kappa_0 + \kappa_1 + \kappa_2 = 1$, any relationship corresponds to a point in the triangle in the (κ_0, κ_2)-plane defined by $\kappa_0 \geq 0, \kappa_2 \geq 0, \kappa_0 + \kappa_2 \leq 1$. The choice of κ_0 and κ_2 as the axis variables is done for reasons of symmetry and is not significant (other authors have used different views of the triangle).

As shown in (Thompson, 1976) points in the subset of the triangle defined by $4\kappa_0\kappa_2 > \kappa_1^2$ is unattainable for pairwise relationships. By default this region in shaded in a 'lightgray' colour.

The IBD coefficients are linearly related to the kinship coefficient ϕ by the formula

$$\phi = 0.25\kappa_1 + 0.5\kappa_2.$$

By indicating values for ϕ in the `kinship.lines` argument, the corresponding contour lines are shown as dashed lines in the triangle plot.

References

E. A. Thompson (1975). *The estimation of pairwise relationships.* Annals of Human Genetics 39.

E. A. Thompson (1976). *A restriction on the space of genetic relationships.* Annals of Human Genetics 40.

See Also

`examineKinships`

Examples

```
IBDtriangle()
IBDtriangle(kinship=c(0.25, 0.125), shading=NULL, cex_text=0.8)
```

`is.linkdat` *Is an object a linkdat object?*

Description

Functions for checking whether an object is a `linkdat` object, a `singleton` or a list of such.

Usage

```
is.linkdat(x)

is.singleton(x)

is.linkdat.list(x)
```

Arguments

x	Any R object.

Details

Note that the `singleton` class inherits from `linkdat`, so if `x` is a singleton, `is.linkdat(x)` returns TRUE.

Value

For `is.linkdat`: TRUE if `x` is a linkdat (or singleton) object, and FALSE otherwise.
For `is.singleton`: TRUE if `x` is a singleton object, and FALSE otherwise.
For `is.linkdat.list`: TRUE if `x` is a list of linkdat/singleton objects.

See Also

`linkdat`

Examples

```
x1 = nuclearPed(1)
x2 = singleton(1)
stopifnot(is.linkdat(x1), !is.singleton(x1),
          is.linkdat(x2), is.singleton(x2),
          is.linkdat.list(list(x1,x2)))
```

`likelihood` *Pedigree likelihood*

Description

Calculates various forms of pedigree likelihoods.

Usage

```
likelihood(x, ...)

## S3 method for class 'linkdat'
likelihood(
  x,
  locus1,
  locus2 = NULL,
  theta = NULL,
  startdata = NULL,
  eliminate = 0,
  logbase = NULL,
  loop_breakers = NULL,
  ...
)

## S3 method for class 'singleton'
likelihood(x, locus1, logbase = NULL, ...)

## S3 method for class 'list'
likelihood(x, locus1, locus2 = NULL, ..., returnprod = TRUE)

likelihood_LINKAGE(
  x,
  marker,
  theta = NULL,
  afreq = NULL,
  logbase = NULL,
  TR.MATR = NULL,
  initialCalc = NULL,
  singleNum.geno = NULL,
  loop_breakers = NULL
)
```

Arguments

`x`	a `linkdat` object, a `singleton` object, or a list of such objects. In `likelihood_LINKAGE`, `x` must be a `linkdat` object, with `x$model` different from NULL.
`...`	further arguments.
`locus1`	a `marker` object compatible with `x`. If `x` is a list, then `locus1` must be a list of corresponding `marker` objects.
`locus2`	either NULL, the character 'disease', or a `marker` object compatible with `x`. See Details.
`theta`	the recombination rate between locus1 and locus2 (in `likelihood_LINKAGE`: between the marker and the disease locus). To make biological sense theta should be between 0 and 0.5.
`startdata`	for internal use.
`eliminate`	mostly for internal use: a non-negative integer indicating the number of iterations in the internal genotype-compatibility algorithm. Positive values can save time if `partialmarker` is non-empty and the number of alleles is large.
`logbase`	a numeric, or NULL. If numeric the log-likelihood is returned, with `logbase` as basis for the logarithm.
`loop_breakers`	a numeric containing IDs of individuals to be used as loop breakers. If NULL, automatic selection of loop breakers will be performed. See `breakLoops`.
`returnprod`	a logical; if TRUE, the product of the likelihoods is returned, otherwise a vector with the likelihoods for each pedigree in the list.
`marker`	an integer between 0 and `x$nMark`, indicating which marker to use in the calculation.
`afreq`	a numeric containing the marker allele frequencies.

`TR.MATR`, `initialCalc`, `singleNum.geno`
: for internal use, speeding up linkage computations with few-allelic markers.

Details

All likelihoods are calculated using the Elston-Stewart algorithm.

If `locus2 = NULL`, the result is simply the likelihood of the genotypes observed at the marker in locus1.

If `locus2 = 'disease'`, the result is the likelihood of the marker genotypes in locus1, given the affection statuses of the pedigree members, the disease model and the recombination rate `theta` between the marker and disease loci. The main use of this is for computation of LOD scores in parametric linkage analysis.

If `locus2` is a marker object, the result is the likelihood of the genotypes at the two markers, given the recombination rate theta between them.

The function `likelihood_LINKAGE` is a fast version of `likelihood.linkdat` in the case where `locus2 = 'disease'` and the marker in locus1 has less than 5 alleles.

Value

The likelihood of the data. If the parameter `logbase` is a positive number, the output is `log(likelihood, logbase)`.

See Also

`lod`

Examples

```
x = linkdat(toyped, model=1) #dominant model

lod1 = likelihood_LINKAGE(x, marker=1, theta=0, logbase=10) -
       likelihood_LINKAGE(x, marker=1, theta=0.5, logbase=10)
lod2 = lod(x, markers=1, theta=0)

# these should be the same:
stopifnot(identical(lod1, as.numeric(lod2)), round(lod1, 2)==0.3)

# likelihood of inbred pedigree (grandfather/granddaughter incest)
y = addOffspring(addDaughter(nuclearPed(1, sex=2), 3), father=1, mother=5, 1)
m = marker(y, 1, 1, 6, 1:2)
l1 = likelihood(y, m)
l2 = likelihood(y, m, loop_breaker=5) # manual specification of loop_breaker
stopifnot(l1==0.09375, l2==l1)
```

`linkage.power` *Power of a linkage study*

Description

Power analysis of parametric linkage studies

Usage

```
linkage.power(
  x,
  N = 100,
  available = x$available,
  afreq = c(0.5, 0.5),
  loop_breakers = NULL,
  threshold = NULL,
  seed = NULL,
  verbose = FALSE
)

## S3 method for class 'powres'
summary(object, threshold = NULL, ...)
```

Arguments

`x`	a `linkdat` object with a valid `model`. (See `setModel`.)
`N`	an integer; the number of markers to simulate.
`available`	a vector containing IDs of the available individuals, i.e. those whose genotypes should be simulated.
`afreq`	a numerical vector with sum 1; the population frequencies for the marker alleles.
`loop_breakers`	a numeric containing IDs of individuals to be used as loop breakers. Relevant only if the pedigree has loops. See `breakLoops`.
`threshold`	NULL, or a single numeric. If numeric, the output includes the percentage of simulated markers having LOD larger than `threshold`.
`seed`	NULL, or a numeric seed for the random number generator.
`verbose`	a logical passed on to `linkageSim`. If `TRUE`, some details are shown during the marker simulation.
`object`	a `powres` object, normally produced by `linkage.power`.
`...`	not used.

Value

The function prints a summary and returns invisibly a `powres` object, which is a list with the following entries:

`sim`	A `linkdat` object with the simulated markers
`lod`	The LOD scores (computed with recombination fraction theta=0) of the simulated markers
`maxlod`	The highest LOD score of the simulated markers
`elod`	The average LOD score for the simulated markers

returns the maximum LOD score for each element of `values`.

References

Marker simulation is inspired by the SLINK algorithm.

See Also

`linkdat`, `linkageSim`

Examples

```
# Note: In the examples below N is set very low in order to reduce time consumption.
# Increase N to get more interesting results.

x = nuclearPed(3)
x = swapAff(x, c(1,3,4))
x = setModel(x, 1) # Autosomal dominant
linkage.power(x, N=1)

# X-linked recessive example:
y = linkdat(Xped, model=4)
linkage.power(y, N=1)

# Power of homozygosity mapping:
z = addOffspring(cousinPed(1), father=7, mother=8, noffs=1, aff=2)
z = setModel(z, 2) # Autosomal recessive model
pow = linkage.power(z, N=1, loop_breaker=7, seed=123)
stopifnot(round(pow$maxlod, 1) == 1.2)
```

`linkageSim` *Simulate markers linked to a disease locus.*

Description

Simulates markers (with up to 4 alleles) conditional on the pedigree structure, affection statuses and disease model.

Usage

```
linkageSim(
  x,
  N = 1,
  available = x$available,
  afreq = NULL,
  partialmarker = NULL,
  loop_breakers = NULL,
  unique = FALSE,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

`x`	a `linkdat` object
`N`	a positive integer: the number of markers to be simulated
`available`	a vector containing IDs of the available individuals, i.e. those whose genotypes should be simulated.
`afreq`	a vector of length < 5 containing the population frequencies for the marker alleles.
`partialmarker`	Either NULL (indicating no given marker data), or a `marker` object.
`loop_breakers`	a numeric containing IDs of individuals to be used as loop breakers. Relevant only if the pedigree has loops. See `breakLoops`.
`unique`	a logical indicating if duplicates among the simulated markers should be removed.
`seed`	NULL, or a numeric seed for the random number generator.
`verbose`	a logical.

Details

All markers are simulated under the condition that the recombination fraction between the marker and the disease locus is 0. This is an implementation of the algorithm used in SLINK of the LINKAGE/FASTLINK suite.

Value

a `linkdat` object equal to `x` except its `markerdata` entry, which consists of the `N` simulated markers.

References

G. M. Lathrop, J.-M. Lalouel, C. Julier, and J. Ott (1984). *Strategies for Multilocus Analysis in Humans*, PNAS 81, pp. 3443-3446.

See Also

`linkage.power`

Examples

```
x = linkdat(toyped, model=1)
y = linkageSim(x, N=10, afreq=c(0.5, 0.5))
stopifnot(length(mendelianCheck(y))==0)

z = addOffspring(cousinPed(1), father=7, mother=8, noffs=1, aff=2)
z = setModel(z, 2)
linkageSim(z, N=1, afreq = c(0.1, 0.2, 0.7))
```

`linkdat` *Linkdat objects*

Description

Functions to create and display 'linkdat' objects.

Usage

```
linkdat(
  ped,
  model = NULL,
  map = NULL,
  dat = NULL,
  freq = NULL,
  annotations = NULL,
  missing = 0,
  header = FALSE,
  checkped = TRUE,
  verbose = TRUE,
  ...
)

singleton(id, sex = 1, famid = 1, verbose = FALSE, ...)

## S3 method for class 'linkdat'
print(x, ..., markers)

## S3 method for class 'linkdat'
summary(object, ...)

write.linkdat(
  x,
  prefix = "",
  what = c("ped", "map", "dat", "freq", "model"),
  merlin = FALSE
)
```

```
## S3 method for class 'linkdat'
subset(x, subset = x$orig.ids, ..., markers = seq_len(x$nMark))
```

Arguments

`ped`	a matrix, data.frame or a character with the path to a pedigree file in standard LINKAGE format. (See details)
`model`	either a `linkdat.model` object (typically `y$model` for some linkdat object `y`), or a single integer with the following meaning: 1 = autosomal dominant; 2 = autosomal recessive; 3 = X-linked dominant; 4 = X-linked recessive. In each of these cases, the disease is assumed fully penetrant and the disease allele frequency is set to 0.00001. If `model=NULL`, no model is set.
`map`	a character with the path to a map file in MERLIN format, or NULL. If non-NULL, a dat file must also be given (next item).
`dat`	a character with the path to a dat file in MERLIN format, or NULL. (Only needed if `map` is non-NULL.)
`freq`	a character with the path to a allele frequency file in MERLIN (short) format, or NULL. If NULL, all markers are interpreted as equifrequent.
`annotations`	a list (of the same length and in the same order as the marker columns in `x`) of marker annotations. If this is non-NULL, then all of `map, dat, freq` should be NULL.
`missing`	the character (of length 1) used for missing alleles. Defaults to '0'.
`header`	a logical, relevant only if ped points to a ped file: If TRUE, the first line of the ped file is skipped.
`checkped`	a logical. If FALSE, no checks for pedigree errors are performed.
`verbose`	a logical: verbose output or not.
`...`	further arguments.
`id, sex`	single numerics describing the individual ID and gender of the singleton.
`famid`	a numeric: the family ID of the singleton.
`x, object`	a `linkdat` object.
`markers`	a numeric indicating which markers should be included/printed.
`prefix`	a character string giving the prefix of the files. For instance, if `prefix='fam1'` and `what=c('ped', 'map')`, the files 'fam1.ped' and 'fam1.map' will be created.
`what`	a character vector forming a subset of c('ped', 'map', 'dat', 'freq', 'model'), indicating which files should be created. All files are written in MERLIN style (but see the next item!)
`merlin`	a logical. If TRUE, the marker alleles are relabeled to 1,2,..., making sure that the generated files are readable by MERLIN (which does not accept non-numerical allele labels in the frequency file.) If FALSE (the default) the allele labels are unchanged. In this case, `x` should be exactly reproducible from the files. (See examples.)
`subset`	a numeric containing the individuals in the sub-pedigree to be extracted. NB: No pedigree checking is done here, so make sure the subset form a meaningful, closed pedigree.

Details

The file (or matrix or data.frame) `ped` must describe one or several pedigrees in standard LINKAGE format, i.e. with the following columns in correct order:

1 Family id (optional) (FAMID)

2 Individual id (ID),

3 Father id (FID),

4 Mother id (MID),

5 Gender (SEX): 1 = male, 2 = female,

6 Affection status (AFF): 1 = unaffected, 2 = affected, 0 = unknown,

7 First allele of first marker,

8 Second allele of first marker,

9 First allele of second marker,

a.s.o.

Only columns 2-6 are mandatory. The first column is automatically interpreted as family id if it has repeated elements.

Internally the individuals are relabeled as 1,2,..., but this should rarely be of concern to the end user. Some pedigree checking is done, but it is recommended to plot the pedigree before doing any analysis.

Details on the formats of map, dat and frequency files can be found in the online MERLIN tutorial: `http://csg.sph.umich.edu/abecasis/Merlin/`

A singleton is a special `linkdat` object whose pedigree contains 1 individual. The class attribute of a singleton is `c('singleton','linkdat')`

Value

A `linkdat` object, or a list of `linkdat` objects. A linkdat object is essentially a list with the following entries, some of which can be NULL.

`pedigree`	`data.frame` with 5 columns (ID, FID, MID, SEX, AFF) describing the pedigree in linkage format. (NB: Internal labeling used.)
`orig.ids`	the original individual id labels.
`nInd`	the number of individuals in the pedigree.
`founders`	vector of the founder individuals. (NB: Internal labeling used.)
`nonfounders`	vector of the nonfounder individuals (NB: Internal labeling used.)
`hasLoops`	a logical: TRUE if the pedigree is inbred.
`subnucs`	list containing all (maximal) nuclear families in the pedigree. Each nuclear family is given as a vector of the form c(pivot, father, mother, child1, ...), where the pivot is either the id of the individual linking the nuclear family to the rest of the pedigree, or 0 if there are none. (NB: Internal labeling used.)
`markerdata`	a list of `marker` objects.
`nMark`	the number of markers.

`available`	a numeric vector containing IDs of available individuals. Used for simulations and plots.
`model`	a `linkdat.model` object, essentially a list containing the model parameters. See `setModel` for details.
`loop_breakers`	a matrix with original loop breaker ID's in the first column and their duplicates in the second column. This is set by `breakLoops`.

See Also

`pedCreate`, `pedModify`, `pedParts`, `setModel`

Examples

```
x = linkdat(toyped, model=1)
x
summary(x)

#### test read/write:
x = modifyMarker(x, 1, alleles=c('B','C'), afreq=c(.9, .1), chrom=2, name='SNP1', pos=123)
write.linkdat(x, prefix='toy')
y = linkdat('toy.ped', map='toy.map', dat='toy.dat', freq='toy.freq', model=1)
unlink(c('toy.ped', 'toy.map', 'toy.dat', 'toy.freq', 'toy.model'))
stopifnot(isTRUE(all.equal(x,y)))

#### test singletons:
w = singleton(id=3, sex=2)
T1 = all.equal(w, linkdat(ped=rbind(c(3,0,0,2,1))))
w = markerSim(w, N=5, alleles=2, afreq=c(0.1,.9))
T2 = all.equal(w, relabel(relabel(w, 10), 3))
T3 = all.equal(w, swapSex(swapSex(w, 3), 3))
T4 = all.equal(w, swapAff(swapAff(w, 3), 3))
stopifnot(T1, T2, T3, T4)

#### several ways of creating the same linkdat object:
alleles = c(157,160,163)
afreq = c(0.3, 0.3, 0.4)
gt10 = c(160, 160)
gt14 = c(160, 163)

z1 = relabel(addOffspring(nuclearPed(1), father=3, noffs=1, aff=2), 10:14)
z1 = addMarker(z1, marker(z1, 10, gt10, 14, gt14, alleles=alleles, afreq=afreq))
z1 = setModel(z1, 2)

z2 = addParents(relabel(nuclearPed(1), 12:14), 12, father=10, mother=11)
z2 = addMarker(z2, rbind(gt10, 0, 0, 0, gt14), alleles=alleles, afreq=afreq)
z2 = setModel(swapAff(z2, 14), 2)

z3 = linkdat(data.frame(ID=10:14, FID=c(0,0,10,0,12), MID=c(0,0,11,0,13),
             SEX=c(1,2,1,2,1), AFF=c(1,1,1,1,2),
             M=c('160/160', '0/0', '0/0', '0/0', '160/163')), model=2)
z3 = modifyMarker(z3, 1, alleles=alleles, afreq=afreq)
```

```
write.linkdat(z1, prefix='test')
z4 = linkdat('test.ped', map='test.map', dat='test.dat', freq='test.freq',
             model=2)
z4 = modifyMarker(z4, 1, alleles=alleles, chrom=NA, pos=NA, name=NA)

write.linkdat(z1, prefix='test', merlin=TRUE)
z5 = linkdat('test.ped', map='test.map', dat='test.dat', freq='test.freq',
             model=2)
z5 = modifyMarker(z5, 1, alleles=alleles, chrom=NA, pos=NA, name=NA)

stopifnot(isTRUE(all.equal(z1,z2)), isTRUE(all.equal(z1,z3)),
          isTRUE(all.equal(z1,z4)), isTRUE(all.equal(z1,z5)))
unlink(c('test.ped', 'test.map', 'test.dat', 'test.freq', 'test.model'))
```

---

linkres *S3 methods for class 'linkres'.*

Description

Functions for printing, summarizing and plotting the results of a linkage analysis.

Usage

```
## S3 method for class 'linkres'
print(x, ...)

## S3 method for class 'linkres'
summary(object, ...)

## S3 method for class 'linkres'
as.data.frame(x, ..., sort = TRUE)

peakSummary(x, threshold, width = 1, physmap = NULL)

## S3 method for class 'linkres'
plot(x, chrom = NULL, ylim = NULL, ...)
```

Arguments

`x, object`	a `linkres` object (normally produced by `lod` or `merlin`).
`...`	further arguments.
`sort`	a logical, indicating if the data frame should be sorted according to map position.
`threshold`	a single numeric. A peak is defined as a regions of at least `width` consecutive markers LOD score above `threshold`.
`width`	a single numeric.

`physmap`	a matrix or data frame with three columns: Marker name, chromosome and physical position. This argument is optional.
`chrom`	NULL, or a numeric containing chromosome numbers. In the latter case only results for the markers on the indicated chromosomes will be plotted.
`ylim`	NULL, or a numeric of length 2: to be passed on to plot.default.

See Also

`lod`, `merlin`

Examples

```
x = linkdat(toyped, model=1)
lods = lod(x, theta='max')
summary(lods)
as.data.frame(lods)
```

`lod` *Two-point LOD score*

Description

Calculates the two-point LOD scores of a pedigree for the specified markers. The recombination ratio between the disease and marker loci can be either fixed at specific values, or optimized.

Usage

```
lod(
  x,
  markers = seq_len(x$nMark),
  theta = 0,
  loop_breakers = NULL,
  max.only = FALSE,
  verbose = FALSE,
  tol = 0.01
)
```

Arguments

`x`	a `linkdat` object.
`markers`	an integer vector denoting which markers to use.
`theta`	either a numeric containing specific recombination ratio(s), or the word 'max', indicating that the recombination ratio should be optimized by the program.
`loop_breakers`	a numeric containing IDs of individuals to be used as loop breakers. Relevant only if the pedigree has loops. See `breakLoops`.

`max.only`	a logical indicating whether only the maximum LOD score should be returned.
`verbose`	a logical: verbose output or not.
`tol`	a numeric passed on to `optimize` as its tolerance parameter.

Details

The LOD score of a marker is defined as

$$LOD(\theta) = \log[10] \frac{L(\theta)}{L(0.5)}$$

where $L(\theta)$ denotes the likelihood of the observed marker genotypes given a recombination ratio θ between the marker and the disease locus.

Value

If `max.only=TRUE`, the highest computed LOD score is returned, as a single number.

Otherwise a `linkres` object, which is essentially a matrix containing the LOD scores. The details depend on the other parameters:

If `theta` is numeric, the matrix has dimensions `length(theta) * length(markers)`, and the entry in row `t`, column `m` is the lod score of the pedigree for marker `m` assuming a recombination rate of `t`.

If `theta='max'`, the `linkres` matrix has one column per marker and two rows: The first containing the LOD score and the second the optimal recombination ratio for each marker.

If a marker has incompatible values (i.e. if a child of homozygous 1/1 parents has a 2 allele), the corresponding output entries are `NaN`.

See Also

`likelihood`, `optimize`, `breakLoops`

Examples

```
x = linkdat(toyped, model=1)
res = lod(x)
res_theta = lod(x, theta=c(0, 0.1, 0.2, 0.5))
res_max = lod(x, theta='max')
stopifnot(all(0.3 == round(c(res, res_theta['0',], res_max['LOD',]), 1)))

# bigger pedigree with several markers
y = linkdat(dominant)
y = setModel(y, model=1, penetrances=c(.001, .9, .99))
lod(y, markers=305:310)
lod(y, markers=305:310, theta='max')

# Example with pedigree with loops:
z = linkdat(twoloops, model=2) # fully penetrant autosomal recessive model.

# add SNP for which individuals 15 and 16 are homozygous for the rare allele.
m = marker(z, 15:16, c(1,1), alleles=1:2, afreq=c(0.001, 0.999))
z = addMarker(z, m)
```

```
res1 = lod(z)
# manual specification of loop breakers gives same result
res2 = lod(z, loop_breakers=c(8,12))

# making the marker triallelic and adding some genotypes.
z = modifyMarker(z, marker=1, ids=c(7,9,11,13), genotype=3, alleles=1:3, afreq=c(0.001, 0.499, 0.5))
plot(z, 1)
res3 = lod(z)

z = modifyMarker(z, marker=1, alleles=1:4, afreq=c(0.001, 0.499, 0.25, 0.25))
res4 = lod(z)

stopifnot(all(3 == round(c(res1, res2, res3, res4), 1)))
```

lod.peaks *LOD score peaks*

Description

Identify LOD score peaks

Usage

```
lod.peaks(x, threshold, width = 1)
```

Arguments

x	a linkres object
threshold	a single numeric
width	a positive integer

Details

The function first transforms x to a data frame (using as.data.frame.linkres with sort=T. A peak is defined a run of at least width consecutive markers with LOD score above or equal to threshold. If possible, one flanking marker is included on each side of the peak.

Value

A list of data frames.

See Also

linkres, lod, merlin,

Examples

```
## minimal example
x = linkdat(toyped, model=1)
res = lod(x)
peak1 = lod.peaks(res, threshold=0)
peak2 = lod.peaks(res, threshold=0, width=2)
peak3 = lod.peaks(res, threshold=1)
stopifnot(length(peak1)==1, nrow(peak1[[1]])==1, length(peak2)==0, length(peak3)==0)
```

`LR` *Likelihood ratios of pedigree hypotheses*

Description

This function computes likelihood ratios for a given a list of pedigrees (linkdat/singletons objects), one of which is the 'reference', with genotype data from the same set of markers. Data exported from the 'Familias' software can be analysed by using `Familias2linkdat` prior to calling this function.

Usage

```
LR(x, ref, markers)
```

Arguments

`x`	A list of pedigrees. Each pedigree is either a single linkdat/singleton object, or a list of such objects (the latter is necessary if the pedigree is disconnected).
`ref`	A single integer, indicating the index of the reference pedigree. This is used in the denominator of each LR.
`markers`	A vector of integers, indexing which markers should be included. If NULL (the default) all markers are used.

Value

A list with entries

`LR`	Likelihood ratios
`LRperMarker`	Likelihood ratios for each marker
`likelihoodsPerSystem`	Likelihoods for each marker
`time`	user, system and elapsed time

Author(s)

Magnus Dehli Vigeland and Thore Egeland

See Also

IBDtriangle, examineKinships

Examples

```
# Simulate genotypes for 5 tetraallelic markers for a pair of full sibs
set.seed(123)
sibs = simpleSim(nuclearPed(2), N=5, alleles=1:4, available=3:4)

# Create two alternative hypotheses and transfer the simulated genotypes to them
halfsibs = addOffspring(nuclearPed(1),father=1,noffs=1,id=4)
halfsibs = transferMarkerdata(sibs, halfsibs)

unrel = list(singleton(3), singleton(4))
unrel = transferMarkerdata(sibs, unrel)

# Compute LR with 'unrelated' as reference
LR(list(sibs, halfsibs, unrel), ref=3)
```

---

markers *Marker functions*

Description

Functions for setting and manipulating marker genotypes for 'linkdat' objects.

Usage

```
marker(
  x,
  ...,
  allelematrix,
  alleles = NULL,
  afreq = NULL,
  missing = 0,
  chrom = NA,
  pos = NA,
  name = NA,
  mutmat = NULL
)

addMarker(x, m, ...)

setMarkers(x, m, annotations = NULL, missing = 0)

modifyMarker(x, marker, ids, genotype, alleles, afreq, chrom, name, pos)
```

```
getMarkers(x, markernames = NULL, chroms = NULL, fromPos = NULL, toPos = NULL)

removeMarkers(
  x,
  markers = NULL,
  markernames = NULL,
  chroms = NULL,
  fromPos = NULL,
  toPos = NULL
)

swapGenotypes(x, ids)

modifyMarkerMatrix(x, ids, new.alleles)
```

Arguments

`x`	a `linkdat` object
`...`	an even number of vectors, indicating individuals and their genotypes. See examples.
`allelematrix`	a matrix with one row per pedigree member and two columns per marker, containing the alleles for a single marker.
`alleles`	a numeric or character vector containing allele names.
`afreq`	a numerical vector with allele frequencies. An error is given if they don't sum to 1 (rounded to 3 decimals).
`missing`	a numeric - or character - of length 1, indicating the code for missing alleles.
`chrom`	NA or an integer (the chromosome number of the marker).
`pos`	NA or a non-negative real number indicating the genetic position (in cM) of the marker.
`name`	NA or a character (the name of the marker).
`mutmat`	a mutation matrix, or a list of two such matrices named 'female' and 'male'. The matrix/matrices must be square, with the allele labels as dimnames, and each row must sum to 1 (after rounding to 3 decimals).
`m`	a `marker` object or a matrix with alleles. (In `setMarkers` this matrix can contain data of several markers.)
`annotations`	a list of marker annotations.
`marker, markers`	a numeric indicating which marker(s) to use/modify.
`ids`	a numeric indicating individual(s) to be modified. In `swapGenotypes` this must have length 2.
`genotype`	a vector of length 1 or 2, containing the genotype to be given the `ids` individuals. See examples.
`markernames`	NULL or a character vector.
`chroms`	NULL or a numeric vector of chromosome numbers.
`fromPos, toPos`	NULL or a single numeric.
`new.alleles`	a numerical matrix of dimensions `length(ids),2*x$nMark`. Entries refer to the internal allele numbering.

Value

The `marker` function returns an object of class `marker`: This is a numerical 2-column matrix with one row per individual, and attributes 'alleles' (a character vector with allele names), 'nalleles' (the number of alleles) and 'missing' (the input symbol for missing marker alleles), 'chrom' (chromosome number), 'name' (marker identifier), 'pos' (chromosome position in cM).

For `addMarker`, `setMarkers`, `removeMarkers`, `modifyMarker`, `modifyMarkerMatrix` and `swapGenotypes`, a `linkdat` object is returned, whose `markerdata` element has been set/modified.

For `getMarkers` a numeric vector containing marker numbers (i.e. their indices in `x$markerdata`) for the markers whose 'name' attribute is contained in `markernames`, 'chrom' attribute is contained in `chroms`, and 'pos' attribute is between `from` and `to`. NULL arguments are skipped, so `getMarkers(x)` will return `seq_len(x$nMark)` (i.e. all markers).

See Also

`linkdat`

Examples

```
x = linkdat(toyped)
x = removeMarkers(x, 1) # removing the only marker.
x

# Creating and adding a SNP marker with alleles 'a' and 'b', for which
# individual 1 is heterozygous, individuals 2 and 4 are homozygous for the
# 'b' allele, and individual 3 has a missing genotype.
m1 = marker(x, 1, c('a','b'), c(2,4), c('b','b'))
x = addMarker(x, m1)

# A rare SNP for which both children are heterozygous.
# The 'alleles' argument can be skipped, but is recommended to ensure
# correct order of the frequencies.
m2 = marker(x, 3:4, 1:2, alleles=1:2, afreq=c(0.99, 0.01))
x = addMarker(x, m2)

# Modifying the second marker:
# Making it triallelic, and adding a genotype to the father.
x = modifyMarker(x, marker=2, alleles=1:3, ids=1, genotype=2:3)

# Adding an empty SNP (all genotypes are missing):
x = addMarker(x, 0, alleles=c('A', 'B'))

# Similar shortcut for creating a marker for which all
# pedigree members are homozygous for an allele (say 'b'):
x = addMarker(x, 'b')
# Alternative: m = marker(x, 'b'); addMarker(x, m)
```

markerSim *Marker simulation*

Description

Simulates marker genotypes conditional on the pedigree structure, affection statuses and disease model.

Usage

```
markerSim(
  x,
  N = 1,
  available = NULL,
  alleles = NULL,
  afreq = NULL,
  partialmarker = NULL,
  loop_breakers = NULL,
  eliminate = 0,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

`x`	a `linkdat` object
`N`	a positive integer: the number of markers to be simulated
`available`	a vector containing IDs of the available individuals, i.e. those whose genotypes should be simulated. By default, all individuals are included.
`alleles`	a vector containing the alleles for the marker to be simulation. If a single integer is given, this is interpreted as the number of alleles, and the actual alleles as `1:alleles`. Must be NULL if `partialmarker` is non-NULL.
`afreq`	a vector of length 2 containing the population frequencies for the marker alleles. Must be NULL if `partialmarker` is non-NULL.
`partialmarker`	Either NULL (resulting in unconditional simulation), a marker object (on which the simulation should be conditioned) or the index of an existing marker of `x`.
`loop_breakers`	a numeric containing IDs of individuals to be used as loop breakers. Relevant only if the pedigree has loops, and only if `partialmarker` is non-NULL. See `breakLoops`.
`eliminate`	A non-negative integer, indicating the number of iterations in the internal genotype-compatibility algorithm. Positive values can save time if `partialmarker` is non-NULL and the number of alleles is large.
`seed`	NULL, or a numeric seed for the random number generator.
`verbose`	a logical.

Details

This implements (with various time savers) the algorithm used in SLINK of the LINKAGE/FASTLINK suite. If `partialmarker` is NULL, genotypes are simulated by simple gene dropping, using `simpleSim`.

Value

a `linkdat` object equal to `x` except its `markerdata` entry, which consists of the `N` simulated markers.

References

G. M. Lathrop, J.-M. Lalouel, C. Julier, and J. Ott, *Strategies for Multilocus Analysis in Humans*, PNAS 81(1984), pp. 3443-3446.

See Also

`simpleSim`, `linkage.power`

Examples

```
x = nuclearPed(2)
partial = marker(x, 3, 1, alleles=1:3)
markerSim(x, N=1, alleles=1:3)
markerSim(x, N=1, partialmarker=partial)
markerSim(x, N=1, partialmarker=partial)
markerSim(x, N=1, available=4, partialmarker=partial)
```

`mendelianCheck` *Check for Mendelian errors*

Description

Check marker data for Mendelian inconsistencies

Usage

```
mendelianCheck(x, remove = FALSE, verbose = !remove)
```

Arguments

`x`	a `linkdat` object
`remove`	a logical. If FALSE, the function returns the indices of markers found to incorrect. If TRUE, a new `linkdat` object is returned, where the incorrect markers have been deleted.
`verbose`	a logical. If TRUE, details of the markers failing the tests are shown.

Value

A numeric containing the indices of the markers that did not pass the tests, or (if `remove=TRUE`) a new `linkdat` object where the failing markers are removed.

Examples

```
x = nuclearPed(3)

# Adding a SNP with a mendelian error:
# Individual 3 has an allele 'c' not carried by either parents
m1 = marker(x, 1, c('a','a'), 2, c('a','b'), 3, c('a','c'))

# Another erroneous marker: The siblings carry more than 4 different alleles.
m2 = marker(x, 3, c(1,2), 4, c(3,4), 5, c(1,5))

# Another marker with incosistent genotypes among the siblings:
m3 = marker(x, 3, c(1,1), 4, c(2,2), 5, c(3,3))

# Another marker with incosistent genotypes among the siblings:
m4 = marker(x, 3, c(1,1), 4, c(2,3), 5, c(1,4))

# A correct marker (all homozygous for allele 'A')
m5 = marker(x, 1:5, 'A')

# An empty marker
m6 = marker(x)

x = setMarkers(x, list(m1,m2,m3,m4,m5,m6))

# Finding the errors
err_index = mendelianCheck(x, remove=FALSE)
stopifnot(all.equal(err_index, 1:4))

x_remove = mendelianCheck(x, remove=TRUE)
stopifnot(x_remove$nMark == 2)
```

`mergePed` *Merge two pedigrees*

Description

This function merges two linkdat objects, joining them at the individuals with equal ID labels. This is especially useful for building 'top-heavy' pedigrees. Only linkdat objects without marker data are supported.

Usage

```
mergePed(x, y, quick = FALSE)
```

Arguments

x, y	linkdat objects
quick	a single logical. If TRUE, no pedigree checks are performed, and the individual ordering may be unfortunate.

Value

A linkdat object.

Examples

```
# Creating a trio where each parent have first cousin parents.
# (Alternatively, this could be built using many calls to addParents().)

x = cousinPed(1)
x = addOffspring(x, father=7, mother=8, noffs=1, id=9)
x = addOffspring(x, father=9, mother=10, noffs=1, id=11)

y = relabel(cousinPed(1), 101:108)
y = addOffspring(y, father=107, mother=108, noffs=1, sex=2, id=10)
y = addOffspring(y, father=9, mother=10, noffs=1, id=11)

# Joining x and y at the common individuals 9,10,11:
z = mergePed(x,y)

# plot all three pedigrees
op = par(mfrow = c(1,3))
plot(x); plot(y); plot(z)
par(op)
```

---

merlin *MERLIN wrappers*

Description

Wrappers for the MERLIN software, providing multipoint LOD scores and other computations on pedigrees with marker data. These functions require MERLIN to be installed and correctly pointed to in the PATH environment variable.

Usage

```
merlin(
  x,
  markers = seq_len(x$nMark),
  model = TRUE,
  theta = NULL,
  options = "",
  verbose = FALSE,
  generate.files = TRUE,
  cleanup = generate.files,
  logfile = ""
)

merlinUnlikely(x, remove = FALSE, verbose = !remove)
```

Arguments

`x`	a `linkdat` object
`markers`	an integer vector indicating which markers to use (default: all).
`model`	a logical: If TRUE (and x$model is not NULL), the file 'merlin.model' is created and '–model merlin.model' is included to the MERLIN command.
`theta`	a numeric with values between 0 and 0.5: The recombination value(s) for which the LOD score is computed. The values of `theta` are converted to centiMorgan positions using the Haldane map function and included in the MERLIN command using the `--position` parameter. Works only for single markers (i.e. `markers` must consist of a single integer).
`options`	a character with additional options to the MERLIN command. See details.
`verbose`	a logical: Show MERLIN output and other information, or not.
`generate.files`	a logical. If TRUE, the files 'merlin.ped', 'merlin.dat', 'merlin.map', 'merlin.freq' and (if `model=TRUE`) 'merlin.model' are created in the working directory.
`cleanup`	a logical: Should the MERLIN files be deleted automatically?
`logfile`	a character. If this is given, the MERLIN screen output will be written to a file with this name.
`remove`	a logical. If FALSE, the function returns the indices of markers found to unlikely. If TRUE, a new `linkdat` object is returned, where the unlikely markers have been deleted.

Details

For these functions to work, MERLIN must be installed and the path to merlin.exe included in the PATH variable. The `merlin` function is first and foremost a wrapper to the parametric linkage functionality of MERLIN.

By default the following MERLIN command is run (via a call to `system`) after creating appropriate files in the current working directory:

```
_merlin.freq --model _merlin.model --tabulate --markerNames --quiet
```

The resulting multipoint LOD scores are extracted from the output and returned in R as a `linkres` object.

Additional command parameters can be passed on using the `options` argument (this is simply pasted onto the MERLIN command, so dashes must be included). For example, to obtain singlepoint LOD scores instead of multipoint, set `options='--singlepoint'`. (The singlepoint scores should agree with the results of `lod(x)`, except in cases where some individuals have partial genotypes (see Examples).)

If `model=FALSE` the `--model merlin.model` part is removed from the MERLIN command above. This is necessary for some calculations, e.g. likelihoods (see Examples).

The `merlinUnlikely` function is a wrapper for MERLIN's '–error' command. The syntax is similar to that of `mendelianCheck`.

Value

If `model=TRUE`, a `linkres` object. Otherwise a character containing the complete MERLIN output.

For `merlinUnlikely`, a numeric containing the indices of the unlikely, or (if `remove=TRUE`) a new `linkdat` object where the unlikely markers are removed.

References

http://csg.sph.umich.edu/abecasis/Merlin/

Examples

```
## Not run:
x = linkdat(toyped, model=1)
x

# MERLIN treats partial genotypes (i.e. one known and one unknown allele) as missing:
lod_merlin = merlin(x)
lod_partial = lod(x)
x = modifyMarker(x, marker=1, ids=1, genotype=0)
lod_missing = lod(x)
stopifnot(lod_merlin == round(lod_missing, 4))

# Likelihood computation by MERLIN:
merlin(x, model=F, options='--lik')

## End(Not run)
```

`oneMarkerDistribution` *Genotype probability distribution*

Description

Computes the (joint) genotype probability distribution of one or several pedigree members, possibly conditional on partial marker data.

Usage

```
oneMarkerDistribution(
  x,
  ids,
  partialmarker,
  theta = NULL,
  grid.subset = NULL,
  loop_breakers = NULL,
  eliminate = 0,
  ignore.affection.status = FALSE,
  verbose = TRUE
)
```

Arguments

`x`	A `linkdat` object.
`ids`	A numeric with ID labels of one or more pedigree members.
`partialmarker`	Either a `marker` object compatible with `x`, or the index (a single integer) of an existing marker of `x`.
`theta`	The recombination fraction between marker and disease locus. Only relevant if at least one individual is affected by disease. In that case an error is raised if `theta` is NULL, and if `x` does not have a disease model.
`grid.subset`	(Not relevant for most end users.) A numeric matrix describing a subset of all marker genotype combinations for the `ids` individuals. The matrix should have one column for each of the `ids` individuals, and one row for each combination: The genotypes are described in terms of the matrix `M = allGenotypes(n)`, where `n` is the number of alleles for the marker. If the entry in column `j` is the integer `k`, this means that the genotype of individual `ids[j]` is row `k` of `M`.
`loop_breakers`	A numeric containing IDs of individuals to be used as loop breakers. Relevant only if the pedigree has loops. See `breakLoops`.
`eliminate`	A non-negative integer, indicating the number of iterations in the internal genotype-compatibility algorithm. Positive values can save time if `partialmarker` is non-empty and the number of alleles is large.
`ignore.affection.status`	A logical indicating if the 'AFF' column should be ignored (only relevant if some family members are marked as affected).
`verbose`	A logical.

Value

A named array (of dimension `length(ids)`) giving the joint marker genotype distribution for the `ids` individuals, conditional on 1) the marker allele frequencies given in `partialmarker`, 2) non-missing alleles in `partialmarker`, and 3) the disease model of `x` (if the pedigree is affected).

See Also

`twoMarkerDistribution`, `allGenotypes`

Examples

```
x = nuclearPed(2)
x_aff = swapAff(x, c(1,3))
x_aff = setModel(x_aff, model=1) # dominant model

snp = marker(x, 1, c(1,1), 2, c(1,0), alleles=1:2, afreq=c(0.1, 0.9))
res1 = oneMarkerDistribution(x, ids=3:4, partialmarker=snp)
res2 = oneMarkerDistribution(x_aff, ids=3:4, partialmarker=snp, theta=0.5)

# should give same result, since theta=0.5 implies that marker is independent of disease.
stopifnot(all.equal(res1, res2))

#### Different example for the same pedigree. A marker with 4 alleles:
m2 = marker(x, 3:4, c('C','D'), alleles=LETTERS[1:4])
oneMarkerDistribution(x, ids=1, partialmarker=m2)

# Same as above, but computing only the cases where individual 1 is heterozygous.
# (The numbers 5:10 refer to the 6 last rows of allGenotypes(4),
# which contain the heterozygous genotypes.)
oneMarkerDistribution(x, ids=1, partialmarker=m2, grid.subset=matrix(5:10, ncol=1))

#### Expanding on the previous example:
# Joint genotype probabilities of the parents, but including only the combinations
# where the father is heterozygous and the mother is homozygous:
grid = expand.grid(5:10, 1:4)
oneMarkerDistribution(x, ids=1:2, partialmarker=m2, grid.subset=grid)

#### Something else:
# The genotype distribution of an individual whose half cousin is homozygous
# for a rare allele.
y = halfCousinPed(degree=1)
snp = marker(y, 9, c('a','a'), alleles=c('a', 'b'), afreq=c(0.01, 0.99))
oneMarkerDistribution(y, ids=8, partialmarker=snp)

#### X-linked example:
z = linkdat(Xped, model=4) # X-linked recessive model
z2 = swapAff(z, 1:z$nInd, 1) # disease free version of the same pedigree

snpX = marker(z, c(5,15), c('A','A'), alleles=c('A', 'B'), chrom=23)

r1 = oneMarkerDistribution(z, ids=13, partialmarker=snpX, theta=0.5) # results: A - 0.8; B - 0.2
r2 = oneMarkerDistribution(z2, ids=13, partialmarker=snpX)       # should be same as above
r3 = oneMarkerDistribution(z, ids=13, partialmarker=snpX, theta=0) # results: A - 0.67; B - 0.33

stopifnot(all.equal(r1,r2), round(r1[1], 2)==0.8, round(r3[1], 2) == 0.67)
```

---

pedCreate *Create simple pedigrees*

Description

These are utility functions for creating some common pedigree structures as `linkdat` objects.

Usage

```
nuclearPed(noffs, sex)

cousinsPed(degree, removal = 0, degree2 = NULL, child = FALSE)

halfCousinsPed(degree, removal = 0, degree2 = NULL, child = FALSE)

doubleCousins(degree1, degree2, removal1 = 0, removal2 = 0, child = FALSE)

doubleFirstCousins()

quadHalfFirstCousins()

fullSibMating(generations)

halfSibStack(generations)

cousinPed(degree)

halfCousinPed(degree)
```

Arguments

`noffs`	A positive integer, the number of offspring in the nuclear family.
`sex`	A vector of length `noffs`; indicating the genders (1=male, 2=female) of the offspring. If missing, all offspring are taken to be males.
`degree`, `degree1`, `degree2`	Non-negative integers, indicating the degree of cousin-like relationships: 0=siblings, 1=first cousins; 2=second cousins, a.s.o. See Details and Examples.
`removal`, `removal1`, `removal2`	Non-negative integers, indicating removals of cousin-like relationships. See Details and Examples.
`child`	A logical: Should an inbred child be added to the two cousins?
`generations`	A positive integer indicating the number of crossings.

Details

All individuals are created as unaffected. Use `swapAff` to edit this (see Examples). Use `swapSex` to change gender of pedigree members.

The call `cousinsPed(degree=n, removal=k)` creates a pedigree with two n'th cousins, k times removed. By default, removals are added on the right side. To override this, the parameter `degree2` can be used to indicate explicitly the number of generations on the right side of the pedigree. When `degree2` is given `removal` is ignored. (Similarly for `halfCousinsPed`.)

The function doubleCousins creates two individuals whose fathers are cousins (degree1, removal1) as well as their mothers (degree2, removal2). For simplicity, a wrapper doubleFirstCousins is provided for the most common case, double first cousins. Finally quadHalfFirstCousins produces a pedigree with quadruple half first cousins.

fullSibMating crosses full sibs continuously for the indicated number of generations.

halfSibStack produces a breeding scheme where the two individuals in the final generation are simultaneously half siblings and half n'th cousins, where n=1,...,generations.

cousinPed and halfCousinPed (written without the 's') are depreciated functions kept for backwards compatibility. They create cousin pedigrees, but without possibility for removals, and with a different ordering than their replacements cousinsPed and halfCousinsPed.

Value

A linkdat object.

See Also

swapAff, swapSex, removeIndividuals, addOffspring, relabel

Examples

```
# A nuclear family with 2 boys and 3 girls,
# where the father and the two boys are affected.
x = nuclearPed(noffs=5, sex=c(1,1,2,2,2))
x = swapAff(x, ids=c(1,3,4))

# Half sibs:
halfCousinsPed(degree=0)

# Grand aunt:
cousinsPed(degree=0, removal=2)

# Second cousins once removed.
cousinsPed(degree=2, removal=1)

# Again second cousins once removed,
# but with the 'removal' on the left side.
cousinsPed(degree=3, degree2=2)

# A child of first cousin parents.
cousinsPed(degree=1, child=TRUE)

# Consecutive brother-sister matings.
fullSibMating(3)

# Simultaneous half siblings and half first cousins
halfSibStack(2)

# Double first cousins
doubleFirstCousins()
```

```
# Quadruple half first cousins
# Weird plotting behaviour for this pedigree.
x = quadHalfFirstCousins()
#plot(x)
```

`pedigreeLoops` *Pedigree loops*

Description

Functions for identifying, breaking and restoring loops in pedigrees.

Usage

```
pedigreeLoops(x)

breakLoops(x, loop_breakers = NULL, verbose = TRUE)

tieLoops(x)

findLoopBreakers(x)

findLoopBreakers2(x)
```

Arguments

`x`	a `linkdat` object.
`loop_breakers`	either NULL (resulting in automatic selection of loop breakers) or a numeric containing IDs of individuals to be used as loop breakers.
`verbose`	a logical: Verbose output or not?

Details

Most of paramlink's handling of pedigree loops is done under the hood - using the functions described here - without need for explicit action from end users. When a linkdat object `x` is created, an internal routine detects if the pedigree contains loops, in which case `x$hasLoops` is set to TRUE. In analyses of `x` where loops must be broken (e.g. lod score computation or marker simulation), this is done automatically by calling `breakLoops`.

In some cases with complex inbreeding, it can be instructive to plot the pedigree after breaking the loops. Duplicated individuals are plotted with appropriate labels (see examples).

The function `findLoopBreakers` identifies a set of individuals breaking all inbreeding loops, but not marriage loops. These require more machinery for efficient detection, and paramlink does this is a separate function, `findLoopBreakers2`, utilizing methods from the `igraph` package. Since this is rarely needed for most users, `igraph` is not imported when loading paramlink, only when `findLoopBreakers2` is called.

In practice, `breakLoops` first calls `findLoopBreakers` and breaks at the returned individuals. If the resulting linkdat object still has loops, `findLoopBreakers2` is called to break any marriage loops.

Value

For `breakLoops`, a `linkdat` object in which the indicated loop breakers are duplicated. The returned object will also have a non-null `loop_breakers` entry, namely a matrix with the IDs of the original loop breakers in the first column and the duplicates in the second.

For `tieLoops`, a `linkdat` object in which any duplicated individuals (as given in the `x$loop_breakers` entry) are merged. For any linkdat object `x`, the call `tieLoops(breakLoops(x))` should return `x`.

For `pedigreeLoops`, a list containing all inbreeding loops (not marriage loops) found in the pedigree. Each loop is represented as a list with elements 'top', a 'bottom' individual, 'pathA' (individuals forming a path from top to bottom) and 'pathB' (creating a different path from top to bottom, with no individuals in common with pathA). Note that the number of loops reported here counts all closed paths in the pedigree and will in general be larger than the genus of the underlying graph.

For `findLoopBreakers` and `findLoopBreakers2`, a numeric vector of individual ID's.

Examples

```
x = cousinsPed(1, child=TRUE)

# Make the child affected, and homozygous for rare allele.
x = swapAff(x, 9)
x = setMarkers(x, marker(x, 9, c(2,2), alleles=1:2, afreq=c(0.99, 0.01)))

# Compute the LOD score under a recessive model. Loops are automatically broken in lod().
x = setModel(x, 2)
LOD1 = lod(x, theta=0.1)
stopifnot(round(LOD1, 2) == 0.88)

# Or we can break the loop manually before computing the LOD:
loopfree = breakLoops(x, loop_breaker=8)
plot(loopfree)
LOD2 = lod(loopfree, theta=0.1)
stopifnot(all.equal(x, tieLoops(loopfree)))
stopifnot(all.equal(LOD1, LOD2))

# Pedigree with marriage loop: Double first cousins
if(requireNamespace("igraph", quietly = TRUE)) {
    y = doubleCousins(1, 1, child=TRUE)
    findLoopBreakers(y) # --> 9
    findLoopBreakers2(y) # --> 9 and 4
    breakLoops(y) # uses both 9 and 4
}
```

pedModify *Modify the pedigree of 'linkdat' objects*

Description

Functions to modify the pedigree of a 'linkdat' object.

Usage

```
swapSex(x, ids, verbose = TRUE)

swapAff(x, ids, newval = NULL)

addOffspring(
  x,
  father,
  mother,
  noffs,
  ids = NULL,
  sex = 1,
  aff = 1,
  verbose = TRUE
)

addSon(x, parent, id = NULL, aff = 1, verbose = TRUE)

addDaughter(x, parent, id = NULL, aff = 1, verbose = TRUE)

addParents(x, id, father, mother, verbose = TRUE)

removeIndividuals(x, ids, verbose = TRUE)

branch(x, id)

trim(x, keep = c("available", "affected"), return.ids = FALSE, verbose = TRUE)

relabel(x, new, old)
```

Arguments

`x`	A `linkdat` object
`verbose`	A logical: Verbose output or not.
`newval`	A numeric, indicating affection status values for the `ids` individuals: 1=unaffected, 2=affected, 0=unknown. If NULL, the affection statuses are swapped 1 <-> 2, hence the main use of the `newval` argument is to assign 0's.
`father, mother`	Integers indicating the IDs of parents. If missing, a new founder individual is created (whose ID will be 1+the largest ID already in the pedigree).
`noffs`	A single integer indicating the number of offspring to be created.
`sex, aff`	Integer vectors indicating the gender and affection statuses of the offspring to be created (recycled if less than `noffs` elements).
`parent`	Integer ID of any pedigree member, which will be the father or mother (depending on its gender) of the new child.
`id, ids`	Individual ID label(s). In `addOffspring` the (optional) `ids` argument is used to specify ID labels for the offspring to be created.

`keep`	A character, either 'available' (trimming the pedigree for unavailable members) or 'affected' (trimming for unaffected members).
`return.ids`	A logical. If FALSE, the trimmed pedigree is returned as a new `linkdat` object. If TRUE, a vector containing the IDs of 'removable' individuals is returned
`new`	a numeric containing new labels to replace those in `old`.
`old`	a numeric containing ID labels to be replaced by those in `new`. If missing, `old` is set to `x$orig.ids`, i.e. all members in their original order.

Details

When removing an individual, all descendants are also removed as well as founders remaining without offspring.

The `branch()` function extracts the pedigree subset consisting of all descendants of `id`, including `id` itself and all relevant spouses.

Value

The modified `linkdat` object.

See Also

`linkdat`, `nuclearPed`

Examples

```
x = linkdat(toyped)

# To see the effect of each command below, use plot(x) in between.
x = addParents(x, id=2, father=5, mother=6)

x = swapSex(x, c(1,5))
x = swapSex(x, c(2,6))

x = addOffspring(x, mother=6, noffs=2, id=c(7,10))
x = removeIndividuals(x, 3)
x = swapAff(x, c(4,10))

stopifnot(setequal(x$orig.ids, c(1,2,4,5,6,7,10,11)))

# Trimming a pedigree
x = linkdat(dominant)
x_affectedOnly = trim(x, keep='affected')

unavail = trim(x, keep='available', return.ids=TRUE)
nonaff = trim(x, keep='affected', return.ids=TRUE)
stopifnot(setequal(unavail, c(5, 19:23)), setequal(nonaff, c(6:7, 12:13, 19:23)))
```

pedParts *Pedigree subsets*

Description

Utility functions for 'linkdat' objects, mainly for extracting various pedigree information.

Usage

```
offspring(x, id, original.id = TRUE)

spouses(x, id, original.id = TRUE)

related.pairs(
  x,
  relation = c("parents", "siblings", "grandparents", "nephews_nieces", "cousins",
    "spouses", "unrelated"),
  available = F,
  interfam = c("none", "founders", "all"),
  ...
)

unrelated(x, id, original.id = TRUE)

leaves(x)

parents(x, id, original.id = TRUE)

grandparents(x, id, degree = 2, original.id = TRUE)

siblings(x, id, half = NA, original.id = TRUE)

cousins(x, id, degree = 1, removal = 0, half = NA, original.id = TRUE)

nephews_nieces(x, id, removal = 1, half = NA, original.id = TRUE)

ancestors(x, id)

descendants(x, id, original.id = TRUE)
```

Arguments

`x`	a `linkdat` object. In `related.pairs` possibly a list of `linkdat` objects.
`id`	a numerical ID label.
`original.id`	a logical indicating whether 'id' refers to the original ID label or the internal labeling.

`relation`	one of the words (possibly truncated) `parents`, `siblings`, `grandparents`, `nephews_nieces`, `cousins`, `spouses`, `unrelated`.
`available`	a logical, if TRUE only pairs of available individuals are returned.
`interfam`	one of the words (possibly truncated) `none`, `founders` or `all`, specifying which interfamiliar pairs should be included as unrelated in the case where `x` is a list of several pedigrees. If `none`, only intrafamiliar pairs are considered; if `founders` all interfamiliar pairs of (available) founders are included; if `all`, all interfamiliar (available) pairs are included.
`...`	further parameters
`degree`	a non-negative integer.
`half`	a logical or NA. If TRUE (resp FALSE), only half (resp. full) siblings/cousins/nephews/nieces are returned. If NA, both categories are included.
`removal`	a non-negative integer

Value

For `ancestors(x,id)`, a vector containing the ID's of all ancestors of the individual `id`. For `descendants(x,id)`, a vector containing the ID's of all descendants (i.e. children, grandchildren, a.s.o.) of individual `id`.

The functions `cousins`, `grandparents`, `nephews_nieces`, `offspring`, `parents`, `siblings`, `spouses`, `unrelated`, each returns an integer vector containing the ID's of all pedigree members having the specified relationship with `id`.

For `related.pairs` a matrix with two columns. Each row gives of a pair of pedigree members with the specified relation. If the input is a list of multiple pedigrees, the matrix entries are characters of the form 'X-Y' where X is the family ID and Y the individual ID of the person.

For `leaves`, a vector of IDs containing all pedigree members without children.

Examples

```
p = cbind(ID=2:9, FID=c(0,0,2,0,4,4,0,2), MID=c(0,0,3,0,5,5,0,8),
        SEX=c(1,2,1,2,1,2,2,2), AFF=c(2,1,2,1,2,1,1,2))
x = linkdat(p)
stopifnot(setequal(spouses(x, 2), c(3,8)),
          setequal(offspring(x, 2), c(4,9)),
          setequal(descendants(x, 2), c(4,6,7,9)),
          setequal(leaves(x), c(6,7,9)))

# Creating a loop and detecting it with 'pedigreeLoops'
# (note that we get two loops, one for each inbred child):
loopx = addOffspring(x, father=4, mother=9, noffs=2)
lps = pedigreeLoops(loopx)
stopifnot(lps[[1]]$top == 2, setequal(sapply(lps, '[[', 'bottom'), 10:11))

# We add genotypes for a single SNP marker and compute a LOD score under a dominant model.
loopx = setMarkers(loopx, cbind(1,c(2,1,2,1,2,1,1,2,1,1)))
loopx = setModel(loopx, 1)

# Loops are automatically broken in lod():
```

```
LOD1 = lod(loopx, theta=0.1)
stopifnot(round(LOD1, 3) == 1.746)

# Or we can break the loop manually before computing the LOD:
loopfree = breakLoops(loopx, loop_breaker=4)
LOD2 = lod(loopfree, theta=0.1)
stopifnot(all.equal(loopx, tieLoops(loopfree)))
stopifnot(all.equal(LOD1, LOD2))
```

---

plot.linkdat *Plot pedigrees with genotypes*

Description

This is the main function for pedigree plotting, with many options for controlling the appearance of pedigree symbols, labels and marker genotypes. Most of the work is done by the plotting functionality in the 'kinship2' package.

Usage

```
## S3 method for class 'linkdat'
plot(
  x,
  marker = NULL,
  alleles = NULL,
  sep = "/",
  missing = "-",
  skip.empty.genotypes = FALSE,
  id.labels = NULL,
  title = NULL,
  available = FALSE,
  col = 1,
  deceased = numeric(0),
  starred = numeric(0),
  aff2 = NULL,
  margins = c(0.6, 1, 4.1, 1),
  ...
)

## S3 method for class 'singleton'
plot(
  x,
  marker = NULL,
  alleles = NULL,
  sep = "/",
  missing = "-",
  skip.empty.genotypes = FALSE,
  id.labels = NULL,
  title = NULL,
  available = FALSE,
  col = 1,
  deceased = numeric(0),
  starred = numeric(0),
  aff2 = NULL,
  margins = c(8, 0, 0, 0),
  ...
)
```

Arguments

`x`	a `linkdat` object.
`marker`	either NULL, a vector of positive integers, a `marker` object, or a list of `marker` objects. If NULL, no genotypes are plotted. If a `marker` object (or a list of such), the genotypes are written below each individual in the pedigree, in the format determined by `alleles`, `sep` and `missing`. If a vector of integers is given, the corresponding marker objects are extracted from `x$markerdata`.
`alleles`	a character vector with allele names.
`sep`	a character of length 1 separating alleles for diploid markers.
`missing`	the symbol (integer or character) for missing alleles.
`skip.empty.genotypes`	a logical. If TRUE, and `marker` is non-NULL, empty genotypes (which by default looks like '-/-') are not printed.
`id.labels`	a vector with labels for each pedigree member. This defaults to `x$plot.labels` is this is set (see `setPlotLabels`), otherwise to `as.character(x$orig.ids)`.
`title`	the plot title. If NULL or '', no title is added to the plot.
`available`	either a logical, a colour name, or the word 'shaded'. If a colour name is given, the available individuals (as defined by `x$available`) are plotted in this colour. If `available=F` no colouring is used, while (for backwards compatibility) `available=T` is equivalent to `available='red'`. The 'shaded' option results in diagonal shading.
`col`	a vector with colour indicators for the pedigree members. Recycled if necessary. By default everyone is drawn black.
`deceased`	a numeric containing ID's of deceased pedigree members.
`starred`	a numeric containing ID's of pedigree members that should be marked with a star in the pedigree plot.
`aff2`	NULL, or a numeric with affection statuses (2=affected, 1=unaffected, 0=unknown) for a second trait.
`margins`	a numeric of length 4 indicating the plot margins. For singletons only the first element (the 'bottom' margin) is used.
`...`	arguments passed on to `plot.pedigree` in the `kinship2` package. In particular `symbolsize` and `cex` can be useful.

Details

`plot.linkdat` is in essence a wrapper for 'plot.pedigree()' in the `kinship2` package.

Author(s)

Magnus Dehli Vigeland, Guro Doerum

See Also

`plot.pedigree`, `setPlotLabels`

Examples

```
data(toyped)
x = linkdat(toyped)
plot(x, marker=1, alleles=c('a1','a2'), sep=' | ', deceased=2)

y = singleton(id=1)
m = marker(y, 1, c('A',0), alleles=c('A','B'))
plot(y, marker=m, id='indiv 1', title='Singleton', available=TRUE)
```

`plotPedList` *Plot a list of pedigrees.*

Description

This function creates a row of pedigree plots, each created by `plot.linkdat`. Each parameter accepted by `plot.linkdat` can be applied here. Some effort is made to guess a reasonable window size and margins, but in general the user must be prepared to do manual resizing of the plot window.

Usage

```
plotPedList(
  plot.arg.list,
  widths = NA,
  frames = TRUE,
  frametitles = NULL,
  fmar = NA,
  newdev = FALSE,
  dev.height = NA,
  dev.width = NA,
  ...
)
```

Arguments

`plot.arg.list`	A list of lists. Each element of `plot.arg.list` is a list, where the first element is the `linkdat` object to be plotted, and the remaining elements are passed on to `plot.linkdat`. These elements must be correctly named. See examples below.
`widths`	A numeric vector of relative widths of the subplots. Recycled to `length(plot.arg.list)` if necessary, before passed on to `layout`. Note that the vector does not need to sum to 1.
`frames`	Either a single logical (FALSE = no frames; TRUE = automatic framing) or a list of numeric vectors: Each vector must consist of consecutive integers, indicating subplots to be framed together. By default the framing follows the list structure of `plot.arg.list`.
`frametitles`	A character vector of titles for each frame. If this is non-NULL, titles for individuals subplots are ignored.
`fmar`	A single number in the interval [0,0.5) controlling the position of the frames.
`newdev`	A logical, indicating if a new plot window should be opened.
`dev.height, dev.width`	The dimensions of the new device (only relevant if newdev is TRUE). If these are NA suitable values are guessed from the pedigree sizes.
`...`	Further arguments passed on to each call to `plot.linkdat`.

Details

See various examples in the Examples section below.

Note that for tweaking dev.height and dev.width the function `dev.size` is useful to determine the size of the active device.

See Also

`plot.linkdat`

Examples

```
# Simplest use: Just give a list of linkdat objects.
# To guess suitable plot window dimensions, use 'newdev=T'
peds = list(nuclearPed(3),cousinPed(2), singleton(12), halfCousinsPed(0))
plotPedList(peds) # try with newdev=TRUE

## Not run:
# Modify the relative widths (which are not guessed)
widths = c(2, 3, 1, 2)
plotPedList(peds, widths=widths)

# In most cases the guessed dimensions are not perfect.
# Resize plot window manually, and then plot again with newdev=F (default)
# plotPedList(peds, widths=widths)

## Remove frames
```

```
plotPedList(peds, widths=widths, frames=F)

# Non-default frames
frames = list(1, 2:3)
plotPedList(peds, widths=widths, frames=frames, frametitles=c('First', 'Second'))

# To give *the same* parameter to all plots, it can just be added at the end:
margins=c(2,4,2,4)
title='Same title'
id.labels=''
symbolsize=1.5 # note: doesn't work as expected for singletons
plotPedList(peds, widths=widths, frames=frames, margins=margins, title=title,
            id.labels=id.labels, symbolsize=symbolsize)

# For more control of individual plots, each plot and all its parameters
# can be specified in its own list:
x1 = nuclearPed(3)
x1$available = 3:5
m1 = marker(x1, 3, 1:2)
marg1 = c(5,4,5,4)
plot1 = list(x1, marker=m1, margins=marg1, title='Plot 1', deceased=1:2)

x2 = cousinsPed(2)
x2$available = leaves(x2)
m2 = marker(x2, leaves(x2), 'A')
marg2 = c(3,4,2,4)
plot2 = list(x2, marker=m2, margins=marg2, title='Plot 2', symbolsize=1.2,
             skip.empty.genotypes=T)

x3 = singleton(12)
x3 = setAvailable(x3, 12)
marg3 = c(10,0,0,0)
plot3 = list(x3, margins=marg3, title='Plot 3', available='shaded', symbolsize=2)

x4 = halfCousinsPed(0)
names4 = c(Father=1, Brother=3, Sister=5)
marg4 = marg1
plot4 = list(x4, margins=marg4, title='Plot 4', id.labels=names4)

plotPedList(list(plot1, plot2, plot3, plot4), widths=c(2,3,1,2),
            frames=list(1,2:3,4), available=T, newdev=T)

# Different example:
plotPedList(list(halfCousinPed(4), cousinsPed(7)), title='Many generations',
    new=T, dev.height=9, dev.width=9)

## End(Not run)
```

`randomPed` *Random pedigree*

Description

Creates a random medical pedigree with specified number of generations.

Usage

```
randomPed(
  gen,
  lambda = 2,
  penetrances = c(0, 1, 1),
  naff = "last.gen",
  founder.mut = 1
)
```

Arguments

`gen`	an integer in the interval `[2,5]` indicating the number of generations.
`lambda`	a positive numeric. For each descendant of the first generation, the number of offspring is sampled from a Poisson distribution with parameter `lambda`.
`penetrances`	a numeric of length 3, as in `setModel`.
`naff`	an integer specifying a lower bound on the number of affected individuals, or the character 'last.gen'. The latter produce a pedigree where at least one in the youngest generation is affected.
`founder.mut`	an integer, the number of disease alleles to be distributed among the founders.

Details

The function produces a random simple pedigree. Each founder is given at most one disease allele. At least one of the two top founders carries a disease allele.

Value

A `linkdat` object.

See Also

`linkdat`

Examples

```
plot(randomPed(3))

# gives error message: Not enough founder mutations
## Not run:
randomPed(gen=4, penetrances=c(0,0,1), naff=2, founder.mut=1)

## End(Not run)
```

readDatfile *Read dat file in LINKAGE format*

Description

Converts dat files in LINKAGE format to dat/map/freq files in MERLIN format

Usage

```
readDatfile(datfile, chrom, comment_string = "<<", write_to = NULL)
```

Arguments

datfile	character. The path to the dat file.
chrom	integer chromosome number (needed to create the MERLIN map).
comment_string	character indicating comments (which are removed before processing).
write_to	a character prefix used for naming the output files, or NULL if no files should be written.

Value

If `write_to` is NULL, a list of data.frames named `dat`, `map` and `freq`.

Examples

```
# No example given.
```

relatednessCoeff *Relatedness coefficients*

Description

Computes inbreeding coefficients for all pedigree members, and Jacquard's condensed identity coefficients for any pair of members. These are simple wrappers for functions in other packages or external programs.

Usage

```
inbreeding(x)

kinship_coefs(x, ids = NULL)

jacquard(x, ids)

jacquard2(x, ids, verbose = FALSE, cleanup = TRUE)
```

Arguments

x	a linkdat object.
ids	a integer vector of length 2.
verbose	a logical, indicating if messages from IdCoefs should be printed.
cleanup	a logical: If TRUE, the pedfile and sample file created for the IdCoefs run are deleted automatically.

Details

Both `inbreeding` and `kinship_coefs` are thin wrappers of `kinship`. `jacquard2`, executes an external call to the C program `IdCoefs` (Abney, 2009). For this to function, `IdCoefs` must be installed on the computer (see link in the References section below) and the executable placed in a folder included in the PATH variable. The `jacquard2` wrapper works by writing the necessary files to disk and calling `IdCoefs` via `system`.

Value

For `inbreeding`, a numerical vector with the inbreeding coefficients, with names according to the ID labels `x$orig.ids`.
For `kinship_coefs`, either a single numeric (if `ids` is a pair of pedigree members) or the whole kinship matrix, with `x$orig.ids` as dimnames.
For `jaquard` and `jaquard2`, a numerical vector of length 9 (in the standard order of Jacquard's condensed identity coefficients).

References

The `IdCoefs` program: Abney, Mark (2009). A graphical algorithm for fast computation of identity coefficients and generalized kinship coefficients. Bioinformatics, 25, 1561-1563. http://home.uchicago.edu/~abney/abney_web/Software.html

See Also

`kinship`

Examples

```
# Offspring of first cousins
x = cousinsPed(1, child=TRUE)
inb = inbreeding(x)
stopifnot(inb[9] == 1/16)

# if ID labels are not 1:9, care must be taken in extracting correct elements.
set.seed(1357)
y = relabel(x, sample(1:9))
child = leaves(y)
inbreeding(y)[child] #wrong
inb = inbreeding(y)[as.character(child)] #correct
inb
# the inbreeding coeff of the child equals the kinship coeff of parents
```

```
kin = kinship_coefs(y, parents(y, child))
stopifnot(inb==kin, inb==1/16)
```

---

relationLR *Relationship Likelihood Ratio*

Description

Computes likelihood for two pedigrees and their ratio, the likelihood ratio (LR).

Usage

```
relationLR(
  ped_numerator,
  ped_denominator,
  ids,
  alleles,
  afreq = NULL,
  known_genotypes = list(),
  loop_breakers = NULL,
  Xchrom = FALSE,
  plot = TRUE,
  title1 = "",
  title2 = ""
)
```

Arguments

`ped_numerator`	a `linkdat` object, or a list of several linkdat and/or singleton objects, describing the relationship corresponding to the hypothesis H1 (numerator). If a list, the sets of ID labels must be disjoint, that is, all ID labels must be unique.
`ped_denominator`	a `linkdat` object, or a list of several linkdat and/or singleton objects, describing the relationship corresponding to the hypothesis H2 (denominator). ID labels must be consistent with `ped_claim`.
`ids`	genotyped individuals.
`alleles`	a numeric or character vector containing marker alleles names
`afreq`	a numerical vector with allele frequencies. An error is given if they don't sum to 1 (rounded to 3 decimals).
`known_genotypes`	list of triplets `(a, b, c)`, indicating that individual `a` has genotype `b/c`. Missing value is 0.
`loop_breakers`	Not yet implemented, only default value NULL currently handled
`Xchrom`	a logical: Is the marker on the X chromosome?

plot	either a logical or the character 'plot_only', controlling if a plot should be produced. If 'plot_only', a plot is drawn, but no further computations are done.
title1	a character, title of leftmost plot.
title2	a character, title of rightmost plot.

Details

This function computes the likelihood of two pedigrees (each corresponding to a hypothesis describing a family relationship). The likelihood ratio is also reported. Unlike other implementations we are aware of, partial DNA profiles are allowed here. For instance, if the genotype of a person is reported as 1/0 (0 is 'missing') for a triallelic marker with uniform allele frequencies, the possible ordered genotypes (1,1), (2,1), (1,2), (1,3) and (3,1) are treated as equally likely. (For general allele frequencies, genotype probabilities are obtained by assuming Hardy-Weinberg equilibrium.) A reasonable future extension would be to allow the user to weigh these genotypes; typically (1,1) may be more likely than the others. If `plot='plot_only'`, the function returns NULL after producing the plot.

Value

lik.numerator	likelihood of data given ped_numerator
lik.denominator	likelihood of data given ped_denominator
LR	likelihood ratio lik.numerator/lik.denominator

Author(s)

Thore Egeland, Magnus Dehli Vigeland

See Also

exclusionPower

Examples

```
############################################
# A partial DNA profile is obtained from the person
# denoted 4 in the figure produced below
# There are two possibilities:
# H1: 4 is the missing relative of 3 and 6 (as shown to the left) or
# H2: 4 is unrelated to 3 and 6.
############################################
p = c(0.2, 0.8)
alleles = 1:length(p)
g3 = c(1,1); g4 = c(1,0); g6 = c(2,2)
x1 = nuclearPed(2)
x1 = addOffspring(x1, father = 4, sex = 1, noff = 1)
m = marker(x1, 3, g3, 4, g4, 6, g6, alleles = alleles, afreq = p)
x1 = addMarker(x1, m)
x2 = nuclearPed(2)
x2 = addOffspring(x2, father = 4, sex = 1, noff = 1)
```

```
m = marker(x2, 3, g3, 6, g6, alleles = alleles, afreq = p)
x2 = addMarker(x2, m)
missing = singleton(4, sex = 1)
m.miss = marker(missing, g4, alleles = alleles, afreq = p)
missing = addMarker(missing, m.miss)
x2 = relabel(x2, c(1:3, 99, 5:6), 1:6)
known = list(c(3, g3), c(4,g4), c(6, g6))
LR = relationLR(x1, list(x2, missing), ids = c(3,4,6),
                alleles = alleles, afreq = p, known = known,
                title1 = 'H1: Missing person 4 related',
                title2 = 'H2:Missing person 4 unrelated')$LR
# Formula:
p = p[1]
LR.a = (1+p)/(2*p*(2-p))
stopifnot(abs(LR - LR.a) < 1e-10)
```

`setAvailable` *Functions for modifying availability vectors*

Description

Functions to set and modify the availability vector of a 'linkdat' object. This vector is used in 'linkage.power' and 'linkageSim', indicating for whom genotypes should be simulated.

Usage

```
setAvailable(x, available)

swapAvailable(x, ids)
```

Arguments

`x`	a `linkdat` object
`available`	a numeric containing the IDs of available individuals.
`ids`	the individual(s) whose availability status should be swapped.

Value

The modified `linkdat` object.

See Also

`plot.linkdat`, `linkage.power`, `linkageSim`

Examples

```
data(toyped)
x = linkdat(toyped)
x = setAvailable(x, 3:4)
x = swapAvailable(x, 2:3)
x$available
```

---

setModel *Set, change or display the model parameters for 'linkdat' objects*

Description

Functions to set, change and display model parameters involved in parametric linkage analysis.

Usage

```
setModel(x, model = NULL, chrom = NULL, penetrances = NULL, dfreq = NULL)

## S3 method for class 'linkdat.model'
print(x, ...)
```

Arguments

`x`	in `setModel`: a `linkdat` object. In `print.linkdat.model`: a `linkdat.model` object.
`model`	NULL, or an object of class `linkdat.model`, namely a list with elements `chrom`, `penetrances` and `dfreq`. In the `setModel` function, the `model` argument can be one of the integers 1-4, with the following meanings: 1 = autosomal dominant; fully penetrant, dfreq=1e-5 2 = autosomal recessive; fully penetrant, dfreq=1e-5 3 = X-linked dominant; fully penetrant, dfreq=1e-5 4 = X-linked recessive; fully penetrant, dfreq=1e-5
`chrom`	a character, either 'AUTOSOMAL' or 'X'. Lower case versions are allowed and will be converted automatically.
`penetrances`	if `chrom=='AUTOSOMAL'`: a numeric of length 3 - `(f0, f1, f2)` - where `fi` is the probability of being affected given `i` disease alleles. If `chrom=='X'`: a list of two vectors, containing the penetrances for each sex: `penetrances = list(male=c(f0, f1), female=c(f0, f1, f2))`.
`dfreq`	the population frequency of the disease allele.
`...`	further parameters

Value

`setModel` returns a new `linkdat` object, whose `model` entry is a `linkdat.model` object: A list containing the given `chrom`, `penetrances` and `dfreq`.

See Also

linkdat

Examples

```
data(toyped)
x = linkdat(toyped)
x1 = setModel(x, model=1)
summary(x1)

# The shortcut 'model=1' above is equivalent to
x2 = setModel(x, chrom='AUTOSOMAL', penetrances=c(0,1,1), dfreq=1e-5)
stopifnot(all.equal(x1, x2))

# X-linked recessive model:
y1 = setModel(x, model=4, dfreq=0.01)
summary(y1)

# Long version of the above:
y2 = setModel(x, chrom='X', penetrances=list(male=c(0,1), female=c(0,0,1)),
              dfreq=0.01)
stopifnot(all.equal(y1, y2))

stopifnot(all.equal(y1, setModel(x, y1$model)))
```

`setPlotLabels` *Attach plot labels to a linkdat object*

Description

This function attaches (or modifies) a character vector of plotting labels for the pedigree members of a linkdat object. This is useful since only numerical ID's are allowed in defining pedigrees in paramlink.

Usage

```
setPlotLabels(x, labels, ids = x$orig.ids)
```

Arguments

`x`	A linkdat object.
`labels`	A character vector of the same length as `ids`.
`ids`	A numeric vector of numerical IDs. Must be a subset of `x$orig.ids`.

Value

A new linkdat object, differing from `x` only in `x$plot.labels`.

See Also

plot.linkdat

Examples

```
x = nuclearPed(1)
x = setPlotLabels(x, labels=c('Father', 'Mother', 'Son'))
plot(x)
```

---

showInTriangle *Add points to the IBD triangle*

Description

Utility function for plotting points in the IBD triangle.

Usage

```
showInTriangle(
  k0,
  k2 = NULL,
  new = T,
  col = "blue",
  cex = 1,
  pch = 4,
  lwd = 2,
  labels = NULL,
  col_labels = col,
  cex_labels = 0.8,
  pos = 1,
  adj = NULL,
  ...
)
```

Arguments

`k0, k2`	Numerical vectors giving coordinates for points to be plotted in the IBDtriangle.
`new`	Logical indicating if a new IBDtriangle should be drawn.
`col, cex, pch, lwd`	Parameters passed onto `points`.
`labels`	A character of same length as `k0`, or NULL.
`col_labels, cex_labels, pos, adj`	Parameters passed onto `text` (if `labels` is non-NULL).
`...`	Plot arguments passed on to `IBDtriangle`.

See Also

`IBDtriangle`, `examineKinships`

Examples

```
showInTriangle(k0=3/8, k2=1/8, label="3/4 siblings", pos=1)
```

`simpleSim` *Unconditional marker simulation*

Description

Unconditional simulation of unlinked markers

Usage

```
simpleSim(
  x,
  N,
  alleles,
  afreq,
  available,
  Xchrom = FALSE,
  mutmat = NULL,
  seed = NULL,
  verbose = T
)
```

Arguments

`x`	a `linkdat` object
`N`	a positive integer: the number of markers to be simulated
`alleles`	a vector containing the allele names. If missing, the alleles are taken to be `seq_along(afreq)`.
`afreq`	a vector of length 2 containing the population frequencies for the alleles. If missing, the alleles are assumed equifrequent.
`available`	a vector containing IDs of the available individuals, i.e. those whose genotypes should be simulated.
`Xchrom`	a logical: X linked markers or not?
`mutmat`	a mutation matrix, or a list of two such matrices named 'female' and 'male'. The matrix/matrices must be square, with the allele labels as dimnames, and each row must sum to 1 (after rounding to 3 decimals).
`seed`	NULL, or a numeric seed for the random number generator.
`verbose`	a logical.

Details

This simulation is done by distributing alleles randomly to all founders, followed by unconditional gene dropping down throughout the pedigree (i.e. for each non-founder a random allele is selected from each of the parents). Finally the genotypes of any individuals not included in `available` are removed.

Value

a `linkdat` object equal to `x` in all respects except its `markerdata` entry, which consists of the `N` simulated markers.

See Also

`markerSim`, `linkageSim`

Examples

```
x = nuclearPed(1)
simpleSim(x, N=3, afreq=c(0.5, 0.5))

y = addOffspring(cousinPed(1), father=7, mother=8, noffs=1)
simpleSim(y, N=3, alleles=LETTERS[1:10])
```

`toyped` *Toy pedigree for linkage analysis*

Description

Toy pedigree with 4 individuals typed at a single SNP marker. Individual 1 is missing one allele.

Usage

```
toyped
```

Format

A data frame with 4 rows and 8 columns

Details

The format is standard LINKAGE (pre-makeped) format, with columns as follows:

- FAMID. Family ID
- ID. Individual ID
- FID. Father ID
- MID. Mother ID

- SEX. Gender (male=1, female=2)
- AFF. Affection status (unaffected=1, affected=2, unknown=0)
- M_A1. First allele of marker 1
- M_A2. Second allele of marker 1

Examples

```
toyped
linkdat(toyped)
```

`transferMarkerdata` *Transfer marker data*

Description

Transfer marker data between pedigrees (in the form of `linkdat` objects). Both the source and target can be lists of linkdat and/or singleton objects (these must have disjoint ID sets). Any previous marker data of the target is overwritten.

Usage

```
transferMarkerdata(from, to)
```

Arguments

`from`	a `linkdat` or `singleton` object, or a list of such objects.
`to`	a `linkdat` or `singleton` object, or a list of such objects.

Value

A `linkdat` object (or a list of such) similar to `to`, but where all individuals also present in `from` have marker genotypes copied over. Any previous marker data is erased.

See Also

`linkdat`

Examples

```
x = list(singleton(id=5), nuclearPed(noffs=2))
x = markerSim(x, N=5, alleles=1:5, verbose=FALSE, available=4:5)
y = nuclearPed(noffs=3)
y = transferMarkerdata(x, y)
stopifnot(all.equal(x[[1]], branch(y,5)))
stopifnot(all.equal(x[[2]], subset(y,1:4)))
```

twoloops *A consanguineous pedigree*

Description

A consanguineous pedigree with two inbreeding loops.

Usage

```
twoloops
```

Format

A data frame with 17 rows and 6 columns. See `toyped` for details about the format.

Examples

```
x = linkdat(twoloops)
plot(x)
```

twoMarkerDistribution *Genotype probability distribution*

Description

Computes the joint genotype distribution of two markers for a specified pedigree member, conditional on existing genotypes and pedigree information.

Usage

```
twoMarkerDistribution(
  x,
  id,
  partialmarker1,
  partialmarker2,
  theta,
  loop_breakers = NULL,
  eliminate = 99,
  verbose = TRUE
)
```

Arguments

`x`	A `linkdat` object.
`id`	The individual in question.
`partialmarker1, partialmarker2`	Either a single integer indicating the number of one of `x`'s existing markers, or a `marker` object.
`theta`	A single numeric in the interval [0, 0.5] - the recombination fraction between the two markers.
`loop_breakers`	A numeric containing IDs of individuals to be used as loop breakers. Relevant only if the pedigree has loops. See `breakLoops`.
`eliminate`	A non-negative integer, indicating the number of iterations in the internal genotype-compatibility algorithm. Positive values can save time if `partialmarker` is non-empty and the number of alleles is large.
`verbose`	A logical.

Value

A named matrix giving the joint genotype distribution.

See Also

`oneMarkerDistribution`

Examples

```
x = nuclearPed(2)
emptySNP = marker(x, alleles=c('a','b'))
SNP1 = marker(x, 1, c(1,1), 2, c(1,0), alleles=1:2, afreq=c(0.1, 0.9))
twoMarkerDistribution(x, id=2, emptySNP, SNP1, theta=0)
twoMarkerDistribution(x, id=2, emptySNP, SNP1, theta=0.5)
twoMarkerDistribution(x, id=3, emptySNP, SNP1, theta=0.5)

# X-linked example
SNPX_1 = marker(x, 2, c('a','b'), 3, 'b', alleles=c('a','b'), chrom=23)
SNPX_2 = marker(x, 2, c('a','b'), 3, 'b', alleles=c('a','b'), chrom=23)
r1 = twoMarkerDistribution(x, id=4, SNPX_1, SNPX_2, theta=0)
r2 = twoMarkerDistribution(x, id=4, SNPX_1, SNPX_2, theta=0.5)
stopifnot(all(r1==c(.5,0,0,.5)), all(r2==c(.25,.25,.25,.25)))
```

`Xped` *Example pedigree with X-linked disease pattern.*

Description

A complex pedigree with an X-linked recessive disease pattern

Usage

```
Xped
```

Format

A data frame with 15 rows and 6 columns. See `toyped` for details about the format.

Details

The format is standard LINKAGE (pre-makeped) format.

Examples

```
Xped

# Convert to a 'linkdat' object and set a recessive X-linked model:
x = linkdat(Xped, model=4)
summary(x)
```

Index