

# Package ‘paramlink’

March 13, 2017

**Type** Package

**Title** Parametric Linkage and Other Pedigree Analysis in R

**Version** 1.1-0

**Date** 2017-03-11

**Author** Magnus Dehli Vigeland

**Maintainer** Magnus Dehli Vigeland <magnusdv@studmed.uio.no>

**Description** 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).

**Depends** R (>= 3.2)

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

**Suggests** identity, igraph

**SystemRequirements** For the MERLIN wrapper, MERLIN (<http://csg.sph.umich.edu/abecasis/Merlin/>) must be installed and pointed to in the PATH environment variable.

**License** GPL (>= 2)

**LazyData** Yes

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2017-03-13 13:18:35

**R topics documented:**

paramlink-package . . . . .	3
allGenotypes . . . . .	3
as.data.frame.linkdat . . . . .	4
as.matrix.linkdat . . . . .	5
createPedigree . . . . .	7
dominant . . . . .	9
examineKinships . . . . .	10
exclusionPower . . . . .	12
Familias2linkdat . . . . .	14
IBDtriangle . . . . .	16
inbreeding . . . . .	17
is.linkdat . . . . .	19
likelihood . . . . .	20
linkage.power . . . . .	22
linkageSim . . . . .	24
linkdat . . . . .	25
linkdat.utils . . . . .	29
linkres . . . . .	31
lod . . . . .	32
lod.peaks . . . . .	34
loops . . . . .	35
markers . . . . .	37
markerSim . . . . .	39
mergePed . . . . .	41
merlin . . . . .	42
modifyPedigree . . . . .	44
oneMarkerDistribution . . . . .	46
plot.linkdat . . . . .	48
plotPedList . . . . .	50
randomPed . . . . .	52
readDatfile . . . . .	53
relationLR . . . . .	54
setAvailable . . . . .	56
setModel . . . . .	57
setPlotLabels . . . . .	58
simpleSim . . . . .	59
toyped . . . . .	60
twoloops . . . . .	61
twoMarkerDistribution . . . . .	61
Xped . . . . .	63

---

paramlink-package	<i>Parametric Linkage and Other Pedigree Analysis in R</i>
-------------------	--

---

### Description

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).

### Author(s)

Magnus Dehli Vigeland

---

allGenotypes	<i>Genotype combinations</i>
--------------	------------------------------

---

### Description

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

### Usage

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

```
allGenotypes(n)
```

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

### Arguments

x	a <a href="#">linkdat</a> object.
partialmarker	a <a href="#">marker</a> 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.
n	a positive integer.
argslist	a list of vectors.
as.list	if TRUE, the output is a list, otherwise a matrix.

**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](#).

**Author(s)**

Magnus Dehli Vigeland

**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 <a href="#">linkdat</a> object.
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.
...	further arguments (not used).

## 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.

## Author(s)

Magnus Dehli Vigeland

## See Also

`print.linkdat`, `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

`as.matrix` creates a matrix representing the given linkdat object in standard LINKAGE format, with marker annotations and other info attached as attributes.

`restore_linkdat` is the reverse of `as.matrix`.

## Usage

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

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

**Arguments**

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

**Details**

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  $x \times n_{\text{Ind}}$  rows and  $6 + 2 \times x \times n_{\text{Mark}}$  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 attributes `markerattr` (a list of marker annotations), `available` (the availability vector) and `model` (the disease model, if present).

For `restore_linkdat`: A linkdat object.

**Author(s)**

Magnus Dehli Vigeland

**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))
```

---

createPedigree	<i>Create simple pedigrees</i>
----------------	--------------------------------

---

**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  $n=1, \dots, \text{generations}$ .

`cousinPed` and `halfCousinPed` (written without the 's') are deprecated 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.

### Author(s)

Magnus Dehli Vigeland

### 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)
```

---

dominant

*Example pedigree for linkage analysis: 23 individuals, 650 SNPs*

---

## 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 24 observations on 1305 variables. The columns are FAMID, ID, FID MID, SEX, AFF, M1\_1, M1\_2, ..., M650\_1, M650\_2. The first 6 columns describe the pedigree structure and affection status, while the final 1300 columns contain the marker alleles. 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	<i>Relatedness estimation</i>
-----------------	-------------------------------

---

### Description

The function `IBDestimate` estimates IBD coefficients ( $k_0$ ,  $k_1$ ,  $k_2$ ) for specified pairs of pedigree members, using maximum likelihood methods (imported from the `maxLik` package).

Checking for pedigree errors is conveniently done with the `examineKinships` function: This computes IBD coefficients for all pairs in the pedigree(s) and produces a color-coded plot where wrong relationships are easy to spot.

These functions work with SNP markers only.

### Usage

```
examineKinships(x, who = "all",
                interfam = c("founders", "none", "all"),
                makeplot = T,
                showRelationships = c("UN", "PO", "MZ", "S", "H,U,G", "FC", "SC"),
                ...)
```

```
IBDestimate(g1, g2=NULL, dat, f=NULL, error=NULL, plot.action=2, pointcol=2, cex=1, ...)
```

### Arguments

<code>x</code>	A <a href="#">linkdat</a> object, or a list of such.
<code>who</code>	A character vector of one or more of the words "parents", "siblings", "grandparents", "cousins", "distant" and "unrelated". Two short forms are possible: "all" (all of the above) and "close" (all of the above except "distant" and "unrelated").
<code>interfam</code>	A character; either "founders", "none" or "all", indicating which interfamilial pairs of individuals should be included. Only relevant if <code>x</code> is a list of several <code>linkdat</code> objects.
<code>makeplot</code>	A logical.
<code>showRelationships</code>	A character vector passed on to the <code>relationships</code> argument of <code>IBDtriangle</code> .
<code>g1, g2</code>	The pairs of individuals for which IBD estimates should be computed. Several formats are possible. If <code>g1</code> is a vector of ID labels, and <code>g2</code> <code>NULL</code> , then all pairs among the <code>g1</code> are included. If <code>g2</code> is not <code>NULL</code> , only pairs intersecting both <code>g1</code> and <code>g2</code> are considered. Finally, <code>g1</code> can be given as a matrix with two columns and each row indicating a pair.
<code>dat</code>	If <code>dat</code> is a list of <code>linkdat</code> objects, then all ID labels must be characters of the form "X-Y", where X is the family ID, and Y the individual ID.
<code>dat</code>	Either a data frame (typically produced by <code>paramlink:::ibdPrep</code> ), a single <code>linkdat</code> object or a list of such.

f	A numeric vector containing the frequencies of the "2" allele. If NULL (default), this is computed from dat.
error	A single numeric to be used in error modelling, or NULL. See details.
plot.action	An integer: 2 = make new triangle and draw points; 1 = add points to existing triangle; 0 = no plotting.
pointcol	A single numeric or character: Color specification for the points in the plot.
cex	A single numeric: Point size. (See <a href="#">par</a> )
...	Other arguments.

### Details

Maximum likelihood estimation is done by the [maxLik](#) function in the `maxLik` package.

More to come here.

### Value

A matrix (or a list of matrices) with 6 columns: ID1, ID2, N (#markers with no missing alleles), k0, k1, k2.

### Author(s)

Magnus Dehli Vigeland

### 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

[maxLik](#), [linkdat](#), [points](#)

### Examples

```
x = cousinsPed(2)
x = simpleSim(x, 1000, 1:2) # 1000 equifrequent SNPs
examineKinships(x, who=c("par", "sib"))

#dontrun
# comparing the founders with another unrelated person:
y = singleton(1, famid=2)
y = simpleSim(y, 1000, 1:2)
examineKinships(list(x, y), who="unrelated", interfam="founders")

# IBD estimates for the cousins (famid=1, iid=7 and 8) vs.
# unrelated singleton (famid=2, iid=1):
ibd = IBDestimate(c("1-7", "1-8"), "2-1", list(x,y), error=0.005)
ibd
```

---

exclusionPower	<i>Power of exclusion</i>
----------------	---------------------------

---

### 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 <a href="#">linkdat</a> 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 <a href="#">linkdat</a> 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 "A general approach to power calculation for relationship testing" (Egeland et al., submitted 2012).

### Value

A single numeric value. If plot="plot\_only", the function returns NULL after producing the plot.

**Author(s)**

Magnus Dehli Vigeland

**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.

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).

## Usage

```
Familias2linkdat(familiasped, datamatrix, loci)
```

```
connectedComponents(ID, FID, MID)
```

**Arguments**

familiasped	A Familias::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 Familias::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.

**Value**

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

**Author(s)**

Magnus Dehli Vigeland and Thore Egeland

**References**

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

**See Also**

See Familias and `linkdat`.

**Examples**

```
# Example
## Not run:
library(Familias)
data(NorwegianFrequencies)
locus1 = FamiliasLocus(NorwegianFrequencies$TH01)
persons <- c("mother", "daughter", "AF")
ped1 <- FamiliasPedigree(id=persons, dadid=c(NA, "AF", NA), momid=c(NA, "mother", NA),
                        sex=c("female", "female", "male"))
datamatrix <- data.frame(TH01.1=c(NA, 8, NA), TH01.2=c(NA,9.3, NA))
rownames(datamatrix) = persons
x = Familias2linkdat(ped1, datamatrix, locus1)
plotPedList(list(x), new=TRUE, frametitles=c("H1"), available="shaded", marker = 1)

# Example
library(fam2r)
data(F21)
pedigrees = F21$pedigrees
datamatrix = F21$datamatrix
loci = F21$loci
x = Familias2linkdat(pedigrees, datamatrix, loci)
plotPedList(x, new=TRUE, frametitles=c("H1", "H2"), available="shaded")
```

```
# Give dev.width explicitly to allow for long names
plotPedList(x, new=TRUE, frametitles=c("H1", "H2"), available="shaded",
            dev.width=17)

# Numerical labels work better
plotPedList(x, new=TRUE, id.labels="num", frametitles=c("H1", "H2"),
            available="shaded")

## End(Not run)
```

---

IBDtriangle

*IBD triangle plot*


---

### Description

This function draws an IBD triangle, as typically used to visualize 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

<code>relationships</code>	A character vector indicating relationships points to be included in the plot. UN=unrelated; PO=parent/offspring; MZ=monozygotic twins; S=full siblings; H=half sibling; U=uncle/aunt;
<code>kinship.lines</code>	A numeric vector.
<code>shading</code>	The shading color for the unattainable region.
<code>pch</code>	Symbol used for the relationship points. (See <a href="#">par</a> .)
<code>cex_points</code>	A single numeric controlling the symbol size for the relationship points.
<code>cex_text</code>	A single numeric controlling the font size for the relationship labels.
<code>axes</code>	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

$$\kappa_i = P(A \text{ and } B \text{ share } i \text{ alleles IBD at random autosomal locus}).$$

Since  $\kappa_0 + \kappa_1 + \kappa_2 = 1$ , any relationship corresponds to a point in the triangle in the  $(\kappa_0, \kappa_2)$ -plane defined by  $\kappa_0 \geq 0, \kappa_2 \geq 0, \kappa_0 + \kappa_2 \leq 1$ . The choice of  $\kappa_0$  and  $\kappa_2$  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 (76) points in the subset of the triangle defined by  $4\kappa_0\kappa_2 > \kappa_1$  is unattainable for pairwise relationships. By default this region is shaded in a "lightgray" color.

The IBD coefficients are linearly related to the kinship coefficient  $\phi$  by the formula

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

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

**Value**

NULL

**Author(s)**

Magnus Dehli Vigeland

**References**

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

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

**See Also**

[examineKinships](#)

**Examples**

```
IBDtriangle()
```

---

inbreeding

*Inbreeding coefficients and Jacquard 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.coeff(x, ids=NULL)
jacquard(x, ids)
jacquard2(x, ids, verbose=FALSE, cleanup=TRUE)
```

**Arguments**

x	a <a href="#">linkdat</a> 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.coeff` are thin wrappers of `kinship2::kinship`. Similarly, `jacquard` wraps `identity::identity.coefs`, which is an R implementation of the C program `IdCoefs` written by Mark Abney. The function `identity::identity.coefs` sometimes causes R to crash, hence as an alternative wrapper, `jacquard2` executes an external call to the original C program `IdCoefs` (version 2.1.1). Of course, this requires `IdCoefs` to be installed on the computer (see link in the References section below) and the executable to be 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.coeff`, either a single numeric (if `ids` is a pair of pedigree members) or the whole kinship matrix, with `x$orig.ids` as dimnames.

For `jacquard` and `jacquard2`, a numerical vector of length 9 (in the standard order of Jacquard's condensed identity coefficients).

**Author(s)**

Magnus Dehli Vigeland

**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](http://home.uchicago.edu/~abney/abney_web/Software.html)

**See Also**

[kinship](#), `identity::identity.coefs`

**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

# the inbreeding coeff of the child equals the kinship coeff of parents
kin = kinship.coeff(y, parents(y, child))
stopifnot(inb==kin, inb==1/16)
```

---

is.linkdat	<i>Is an object a linkdat object?</i>
------------	---------------------------------------

---

## 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.

## Author(s)

Magnus Dehli Vigeland

## 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 <a href="#">linkdat</a> object, a <a href="#">singleton</a> object, or a list of such objects. In likelihood_LINKAGE, x must be a linkdat object, with x\$model different from NULL.
locus1	a <a href="#">marker</a> 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 <a href="#">marker</a> 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.
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 <a href="#">breakLoops</a> .
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.
startdata	for internal use.
TR.MATR, initialCalc, singleNum.geno	for internal use, speeding up linkage computations with few-allelic markers.
...	further arguments. Not used.

### 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  $\log(\text{likelihood}, \text{logbase})$ .

### Author(s)

Magnus Dehli Vigeland

### 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	<i>Power of a linkage study</i>
---------------	---------------------------------

---

### 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 <a href="#">linkdat</a> object with a valid model. (See <a href="#">setModel</a> .)
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 <a href="#">breakLoops</a> .
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 <a href="#">linkageSim</a> . If TRUE, some details are shown during the marker simulation.
object	a <a href="#">powres</a> object, normally produced by <a href="#">linkage.power</a> .
...	not used.

**Value**

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

<code>sim</code>	A <code>linkdat</code> object with the simulated markers
<code>lod</code>	The LOD scores (computed with recombination fraction $\theta=0$ ) of the simulated markers
<code>maxlod</code>	The highest LOD score of the simulated markers
<code>elod</code>	The average LOD score for the simulated markers

**Author(s)**

Magnus Dehli Vigeland

**References**

The marker simulation takes place in `linkageSim`, which implements the algorithm used by SLINK: <http://linkage.rockefeller.edu/ott/SLINK.htm>

**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 <a href="#">linkdat</a> object
N	a positive integer: the number of markers to be simulated
partialmarker	Either NULL (indicating no given marker data), or a marker object.
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.
loop_breakers	a numeric containing IDs of individuals to be used as loop breakers. Relevant only if the pedigree has loops. See <a href="#">breakLoops</a> .
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.

**Author(s)**

Magnus Dehli Vigeland



**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**

[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, ...)
```

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

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

**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.
checkedped	a logical. If FALSE, no checks for pedigree errors are performed.
verbose	a logical: verbose output or not.
id, sex	single numerics describing the individual ID and gender of the singleton.
markers	a numeric indicating which markers should be included/printed.
x, object	a linkdat object.
famid	a numeric: the family ID of the singleton.
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.
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.)
...	further arguments.

## 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 <code>c(pivot, father, mother, child1, ...)</code> , 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 <code>marker</code> 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 <a href="#">setModel</a> 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 <a href="#">breakLoops</a> .

**Author(s)**

Magnus Dehli Vigeland

**See Also**[setModel](#), [modifyPedigree](#), [linkdat.utils](#)**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"))

```

---

linkdat.utils

*linkdat utilities*


---

## Description

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

## Usage

```

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

cousins(x, id, degree=1, removal=0, half=NA, original.id=TRUE)
grandparents(x, id, degree=2, original.id=TRUE)
nephews_nieces(x, id, removal=1, half=NA, original.id=TRUE)
parents(x, id, original.id=TRUE)
offspring(x, id, original.id=TRUE)
siblings(x, id, half=NA, original.id=TRUE)
spouses(x, id, original.id=TRUE)
unrelated(x, id, original.id=TRUE)
related.pairs(x, relation=c('parents', 'siblings', 'grandparents', 'nephews_nieces',
                          'cousins', 'spouses', 'unrelated'), available=F, original.id=T,
              interfam=c("none", "founders", "all"), ...)

leaves(x)

## S3 method for class 'linkdat'
all.equal(target, current, ...)

```

**Arguments**

<code>x</code>	a <code>linkdat</code> object. In <code>related.pairs</code> possibly a list of <code>linkdat</code> objects.
<code>id</code>	a numerical ID label.
<code>original.id</code>	a logical indicating whether 'id' refers to the original ID label or the internal labeling.
<code>degree</code>	a non-negative integer.
<code>removal</code>	a non-negative integer
<code>half</code>	a logical or NA. If TRUE (resp FALSE), only half (resp. full) siblings/cousins/nephews/nieces are returned. If NA, both categories are included.
<code>relation</code>	one of the words (possibly truncated) <code>parents</code> , <code>siblings</code> , <code>grandparents</code> , <code>nephews_nieces</code> , <code>cousins</code> , <code>spouses</code> , <code>unrelated</code> .
<code>interfam</code>	one of the words (possibly truncated) <code>none</code> , <code>founders</code> or <code>all</code> , specifying which interfamilial pairs should be included as unrelated in the case where <code>x</code> is a list of several pedigrees. If <code>none</code> , only intrafamilial pairs are considered; if <code>founders</code> all interfamilial pairs of (available) founders are included; if <code>all</code> , all interfamilial (available) pairs are included.
<code>available</code>	a logical, if TRUE only pairs of available individuals are returned.
<code>target</code> , <code>current</code>	the <code>linkdat</code> objects to be compared.
<code>...</code>	Not used.

**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.

**Author(s)**

Magnus Dehli Vigeland

**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))

```

---

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'
as.data.frame(x, ..., sort=TRUE)

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

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

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

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

```

**Arguments**

x, object	a linkres object (normally produced by <a href="#">lod</a> or <a href="#">merlin</a> ).
sort	a logical, indicating if the data frame should be sorted according to map position.
chrom	NULL, or a numeric containing chromosome numbers. In the latter case only results for the markers on the indicated chromosomes will be plotted.
yylim	NULL, or a numeric of length 2: to be passed on to plot.default.
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.
...	further arguments.

**Author(s)**

Magnus Dehli Vigeland

**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 <a href="#">linkdat</a> 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 <a href="#">breakLoops</a> .
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 <a href="#">optimize</a> as its tolerance parameter.

**Details**

The LOD score is defined as  $\text{LOD}(\theta) = \log_{10}(L(\theta)) - \log_{10}(L(\theta=0.5))$ , where  $L(\theta)$  denotes the likelihood of the observed pedigree data as a function of the recombination ratio  $\theta$  between the marker and the disease locus.

**Value**

If `max.only=FALSE`, a `linkres` object, essentially a matrix containing the LOD scores. If `theta` is numeric, the matrix has dimensions `length(theta)`, `length(markers)`, and the entry in row `t_0`, column `m_0` is the lod score of the pedigree for marker `m_0` assuming a recombination rate of `t_0`.

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 matrix entries are NaN.

If `max.only=TRUE`, only the highest LOD score is returned, as a numeric of length 1.

**Author(s)**

Magnus Dehli Vigeland

**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 <a href="#">linkres</a> 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.

**Author(s)**

MAGnus Dehli Vigeland

**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)
```

---

loops

*Pedigree loops*

---

**Description**

Functions for identifying, breaking and restoring loops in pedigrees.

**Usage**

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

pedigreeLoops(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 in 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.

### Author(s)

Magnus Dehli Vigeland

### 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)
LOD2 = lod(loopfree, theta=0.1)
stopifnot(all.equal(x, tieLoops(loopfree)))
stopifnot(all.equal(LOD1, LOD2))

# Pedigree with marriage loop: Double first cousins
y = doubleCousins(1, 1, child=TRUE)
```

```
findLoopBreakers(y) # --> 9
findLoopBreakers2(y) # --> 9 and 4
breakLoops(y) # uses both 9 and 4
```

---

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)

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

removeMarkers(x, markers, markernames=NULL, chroms=NULL, from=NULL, to=NULL)

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

modifyMarkerMatrix(x, ids, new.alleles)

swapGenotypes(x, ids)

transferMarkerdata(from, to)

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

## Arguments

x	a <a href="#">linkdat</a> 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.
m	a marker object or a matrix with alleles. (In <code>setMarkers</code> this matrix can contain data of several markers.)
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).
annotations	a list of marker annotations.
markernames	NULL or a character vector.
chroms	NULL or a numeric vector of chromosome numbers.
fromPos, toPos	NULL or a single numeric.
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.
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).
new.alleles	a numerical matrix of dimensions $\text{length}(\text{ids})$ , $2 \times n_{\text{Mark}}$ . Entries refer to the internal allele numbering.
from, to	linkdat objects, or list of linkdat objects.
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

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).

For transferMarkerdata, 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.

For mendelianCheck, 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.

**Author(s)**

Magnus Dehli Vigeland

**See Also**[linkdat](#)**Examples**

```

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

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

# A rare SNP for which all individuals are homozygous for the common allele.
# The 'alleles' argument must be specified since only one allele is present.
m2 = marker(x, 1:4, 1, alleles=1:2, afreq=c(0.99, 0.01))
x = addMarker(x, m2)

# Creating a Mendelian error in the first marker
x = modifyMarker(x, marker=1, ids=3, genotype='a')

# Making the second marker triallelic, with heterozygous parents.
x = modifyMarker(x, marker=2, ids=1:2, genotype=1:2, alleles=1:3)

# Testing mendelianCheck
err_index = mendelianCheck(x, remove=FALSE)
stopifnot(err_index==1)

xnew = mendelianCheck(x, remove=TRUE)
stopifnot(all.equal(xnew, removeMarkers(x, 1)))

### Testing transferMarkerdata
x0 = list(singleton(id=5), nuclearPed(noffs=2))
x = markerSim(x0, N=5, alleles=1:5, verbose=FALSE, available=4:5)
y0 = nuclearPed(noffs=3)
y = transferMarkerdata(x, y0)
stopifnot(all.equal(x[[1]], branch(y,5)))
stopifnot(all.equal(x[[2]], subset(y,1:4)))

```

markerSim

*Marker simulation***Description**

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

**Usage**

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

**Arguments**

x	a <a href="#">linkdat</a> 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.
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 <a href="#">breakLoops</a> .
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.

**Author(s)**

Magnus Dehli Vigeland

**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)
```

---

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**

<code>x,y</code>	<a href="#">linkdat</a> objects
<code>quick</code>	a logical. If TRUE, no pedigree checks are performed, and the individual ordering may be unfortunate.

**Value**

A linkdat object.

**Author(s)**

Magnus Dehli Vigeland

**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
par(mfrow=c(1,3))
plot(x); plot(y); plot(z)

```

---

merlin

*MERLIN wrappers*


---

## Description

These functions require MERLIN to be installed and correctly pointed to in the PATH environment variable. The required input files are automatically created (in the working directory) and deleted afterwards. The MERLIN command is executed via a call to `system()`.

## 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 <a href="#">linkdat</a> object
markers	an integer vector indicating which markers to use (default: all).
model	a logical: If TRUE (and <code>x\$model</code> is not NULL), the file "merlin.model" is created and " <code>--model merlin.model</code> " 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 <code>theta</code> are converted to centiMorgan positions using the Haldane map function and included in the MERLIN command using the <code>--position</code> 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 <code>model=TRUE</code> ) "merlin.model" are created in the working directory.
cleanup	a logical: Should the files be deleted automatically after the MERLIN run?

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 -p _merlin.ped -d _merlin.dat -m _merlin.map -f _merlin.freq --model _merlin.model --tabulate -
```

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 single-point 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.

## Author(s)

Magnus Dehli Vigeland

## 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, 3))

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

## End(Not run)
```

---

modifyPedigree	<i>Modify the pedigree of 'linkdat' objects</i>
----------------	---

---

### 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, sex=1, aff=1, verbose=TRUE)

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

addSon(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 <a href="#">linkdat</a> object
id, ids	Individual ID label(s). In <code>addOffspring</code> the (optional) <code>ids</code> argument is used to specify ID labels for the offspring to be created.
newval	A numeric, indicating affection status values for the <code>ids</code> individuals: 1=unaffected, 2=affected, 0=unknown. If <code>NULL</code> , the affection statuses are swapped 1 <-> 2, hence the main use of the <code>newval</code> 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).
verbose	A logical: Verbose output or not.
parent	Integer ID of any pedigree member, which will be the father or mother (depending on its gender) of the new child.
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.

### Author(s)

Magnus Dehli Vigeland

### 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)))

```

---

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 <a href="#">linkdat</a> object.
ids	A numeric with ID labels of one or more pedigree members.
partialmarker	Either a <a href="#">marker</a> 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	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 $M = \text{allGenotypes}(n)$ , 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 <a href="#">breakLoops</a> .
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).

**Author(s)**

Magnus Dehli Vigeland

**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)

```

---

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 <a href="#">linkdat</a> object.
marker	either NULL, a vector of positive integers, a <a href="#">marker</a> 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 if this is set (see <a href="#">setPlotLabels</a> ), 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 color name, or the word "shaded". If a color name is given, the available individuals (as defined by x\$available) are plotted in this color. If available=F no coloring is used, while (for backwards compatibility) available=T is equivalent to available="red". The "shaded" option results in diagonal shading.
col	a vector with color 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 = T, frametitles = NULL, fmar = NA,
            newdev = F, dev.height = NA, dev.width = NA, ...)
```

### Arguments

<code>plot.arg.list</code>	A list of lists. Each element of <code>plot.arg.list</code> is a list, where the first element is the <code>linkdat</code> object to be plotted, and the remaining elements are passed on to <code>plot.linkdat</code> . These elements must be correctly named. See examples below.
<code>widths</code>	A numeric vector of relative widths of the subplots. Recycled to <code>length(plot.arg.list)</code> if necessary, before passed on to <code>layout</code> . Note that the vector does not need to sum to 1.
<code>frames</code>	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 <code>plot.arg.list</code> .
<code>frametitles</code>	A character vector of titles for each frame. If this is non-NULL, titles for individual subplots are ignored.
<code>fmar</code>	A single number in the interval [0,0.5) controlling the position of the frames.
<code>newdev</code>	A logical, indicating if a new plot window should be opened.
<code>dev.height</code> , <code>dev.width</code>	The dimensions of the new device (only relevant if <code>newdev</code> is TRUE). If these are NA suitable values are guessed from the pedigree sizes.
<code>...</code>	Further arguments passed on to each call to <code>plot.linkdat</code> .

### 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.

### Author(s)

Magnus Dehli Vigeland

**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, 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 <a href="#">setModel</a> .
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.

**Author(s)**

Magnus Dehli Vigeland

**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	<i>Read dat file in LINKAGE format</i>
-------------	--

---

**Description**

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

**Usage**

```
readDatfile(datfile, chrom, write_to = NULL)
```

**Arguments**

datfile	character. The path to the dat file.
chrom	integer chromosome number (needed to create the MERLIN map).
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.

**Author(s)**

Magnus Dehli Vigeland

**Examples**

```
# No example given.
```

---

relationLR	<i>Relationship Likelihood Ratio</i>
------------	--------------------------------------

---

### 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 <a href="#">linkdat</a> 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 <a href="#">linkdat</a> 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 and 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 <a href="#">linkdat</a> 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.

### Author(s)

Magnus Dehli Vigeland

### 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	<i>Set, change or display the model parameters for 'linkdat' objects</i>
----------	--

---

**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 <a href="#">linkdat</a> 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.

**Author(s)**

Magnus Dehli Vigeland

**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.

**Author(s)**

Magnus Dehli Vigeland

**See Also**[plot.linkdat](#)**Examples**

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

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 <a href="#">linkdat</a> 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 <code>seq_along(afreq)</code> .
afreq	a vector of length 2 containing the population frequencies for the alleles. If missing, the alleles are assumed equiproportional.
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.

**Author(s)**

Magnus Dehli Vigeland

**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 1 SNP marker. Individual 1 is missing one allele.

**Usage**

toyped

**Format**

A data frame with one row for each individual, and the following columns:

FAMID family id

ID individual id

FID father id (0 if founder)

MID mother id (0 if founder)

SEX sex: male=1, female=2

AFF affection status: healthy=1, affected=2, unknown=0

M\_A1 first marker, first allele

M\_A2 first marker, second allele

**Details**

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

**Examples**

```
toyped  
linkdat(toyped)
```

---

twoloops	<i>A consanguineous pedigree</i>
----------	----------------------------------

---

**Description**

A consanguineous pedigree.

**Usage**

```
twoloops
```

**Format**

A data frame describing the pedigree in standard LINKAGE format (i.e. the columns are: Family ID, individual ID, father, mother, sex, affection status).

**Examples**

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

---

twoMarkerDistribution	<i>Genotype probability distribution</i>
-----------------------	--

---

**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 <a href="#">linkdat</a> 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 <a href="#">breakLoops</a> .
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.

**Author(s)**

Magnus Dehli Vigeland

**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 consistent with an X-linked recessive disease.

**Usage**

Xped

**Format**

A data frame with one row for each individual, and the following columns:

FAMID family id

ID individual id

FID father id (0 if founder)

MID mother id (0 if founder)

SEX sex: male=1, female=2

AFF affection status: healthy=1, affected=2, unknown=0

**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

## \*Topic **IO**

as.data.frame.linkdat, 4  
linkdat, 25  
readDatfile, 53

## \*Topic **datagen**

linkageSim, 24  
markerSim, 39  
simpleSim, 59

## \*Topic **datasets**

dominant, 9  
toyped, 60  
twoLoops, 61  
Xped, 63

## \*Topic **distributions**

oneMarkerDistribution, 46  
randomPed, 52  
twoMarkerDistribution, 61

## \*Topic **dplot**

setPlotLabels, 58

## \*Topic **manip**

as.matrix.linkdat, 5  
createPedigree, 7  
lod.peaks, 34  
markers, 37  
mergePed, 41  
merlin, 42  
modifyPedigree, 44  
setAvailable, 56

## \*Topic **math**

examineKinships, 10  
exclusionPower, 12  
inbreeding, 17  
likelihood, 20  
linkage.power, 22  
linkdat.utils, 29  
lod, 32  
loops, 35  
relationLR, 54  
setModel, 57

## \*Topic **methods**

allGenotypes, 3  
linkres, 31

## \*Topic **package**

paramlink-package, 3

## \*Topic **plot**

plot.linkdat, 48

addDaughter (modifyPedigree), 44  
addMarker (markers), 37  
addOffspring, 8  
addOffspring (modifyPedigree), 44  
addParents (modifyPedigree), 44  
addSon (modifyPedigree), 44  
all.equal.linkdat (linkdat.utils), 29  
allGenotypes, 3, 47  
ancestors (linkdat.utils), 29  
as.data.frame.linkdat, 4, 6  
as.data.frame.linkres, 34  
as.data.frame.linkres (linkres), 31  
as.matrix.linkdat, 5, 5

branch (modifyPedigree), 44  
breakLoops, 21, 22, 24, 28, 33, 40, 46, 62  
breakLoops (loops), 35

connectedComponents (Familias2linkdat),  
14  
cousinPed (createPedigree), 7  
cousins (linkdat.utils), 29  
cousinsPed (createPedigree), 7  
createPedigree, 7

descendants (linkdat.utils), 29  
dev.size, 50  
dominant, 9  
doubleCousins (createPedigree), 7  
doubleFirstCousins (createPedigree), 7  
examineKinships, 10, 17  
exclusionPower, 12, 55



- expand.grid, [4](#)
- Familias2linkdat, [14](#)
- fast.grid (allGenotypes), [3](#)
- findLoopBreakers (loops), [35](#)
- findLoopBreakers2 (loops), [35](#)
- fullSibMating (createPedigree), [7](#)
- geno.grid.subset (allGenotypes), [3](#)
- getMarkers (markers), [37](#)
- grandparents (linkdat.utils), [29](#)
- halfCousinPed (createPedigree), [7](#)
- halfCousinsPed (createPedigree), [7](#)
- halfSibStack (createPedigree), [7](#)
- IBDestimate (examineKinships), [10](#)
- IBDtriangle, [16](#)
- inbreeding, [17](#)
- is.linkdat, [19](#)
- is.singleton (is.linkdat), [19](#)
- jacquard (inbreeding), [17](#)
- jacquard2 (inbreeding), [17](#)
- kinship, [18](#)
- kinship.coeff (inbreeding), [17](#)
- layout, [50](#)
- leaves (linkdat.utils), [29](#)
- likelihood, [20](#), [33](#)
- likelihood\_LINKAGE (likelihood), [20](#)
- linkage.power, [22](#), [25](#), [41](#), [56](#)
- linkageSim, [23](#), [24](#), [56](#), [60](#)
- linkdat, [3](#), [4](#), [6](#), [8](#), [10–12](#), [15](#), [18–20](#), [22–24](#), [25](#), [30](#), [33](#), [35](#), [37](#), [39–42](#), [44–46](#), [48](#), [50](#), [53](#), [54](#), [56](#), [57](#), [59](#), [62](#)
- linkdat.utils, [28](#), [29](#)
- linkres, [31](#), [34](#), [35](#), [43](#)
- lod, [21](#), [32](#), [32](#), [35](#)
- lod.peaks, [34](#)
- loops, [35](#)
- marker, [3](#), [20](#), [27](#), [46](#), [48](#)
- marker (markers), [37](#)
- markers, [37](#)
- markerSim, [39](#), [60](#)
- maxLik, [11](#)
- mendelianCheck, [43](#)
- mendelianCheck (markers), [37](#)
- mergePed, [41](#)
- merlin, [32](#), [35](#), [42](#)
- merlinUnlikely (merlin), [42](#)
- modifyMarker (markers), [37](#)
- modifyMarkerMatrix (markers), [37](#)
- modifyPedigree, [28](#), [44](#)
- nephews\_nieces (linkdat.utils), [29](#)
- nuclearPed, [45](#)
- nuclearPed (createPedigree), [7](#)
- offspring (linkdat.utils), [29](#)
- oneMarkerDistribution, [46](#), [62](#)
- optimize, [33](#)
- par, [11](#), [16](#)
- paramlink (paramlink-package), [3](#)
- paramlink-package, [3](#)
- parents (linkdat.utils), [29](#)
- peakSummary (linkres), [31](#)
- pedigreeLoops (loops), [35](#)
- plot.linkdat, [48](#), [50](#), [51](#), [56](#), [59](#)
- plot.linkres (linkres), [31](#)
- plot.pedigree, [49](#)
- plot.singleton (plot.linkdat), [48](#)
- plotPedList, [50](#)
- points, [11](#)
- power.varyPar (linkage.power), [22](#)
- print.linkdat, [5](#)
- print.linkdat (linkdat), [25](#)
- print.linkdat.model (setModel), [57](#)
- print.linkres (linkres), [31](#)
- quadHalfFirstCousins (createPedigree), [7](#)
- randomPed, [52](#)
- readDatfile, [53](#)
- relabel, [8](#)
- relabel (modifyPedigree), [44](#)
- related.pairs (linkdat.utils), [29](#)
- relationLR, [54](#)
- removeIndividuals, [8](#)
- removeIndividuals (modifyPedigree), [44](#)
- removeMarkers (markers), [37](#)
- restore\_linkdat (as.matrix.linkdat), [5](#)
- setAvailable, [56](#)
- setMap (markers), [37](#)
- setMarkers (markers), [37](#)
- setModel, [22](#), [28](#), [52](#), [57](#)

setPlotLabels, [49](#), [58](#)  
siblings (linkdat.utils), [29](#)  
simpleSim, [40](#), [41](#), [59](#)  
singleton, [20](#)  
singleton (linkdat), [25](#)  
spouses (linkdat.utils), [29](#)  
subset.linkdat (linkdat), [25](#)  
summary.linkdat (linkdat), [25](#)  
summary.linkres (linkres), [31](#)  
summary.powres (linkage.power), [22](#)  
swapAff, [8](#)  
swapAff (modifyPedigree), [44](#)  
swapAvailable (setAvailable), [56](#)  
swapGenotypes (markers), [37](#)  
swapSex, [8](#)  
swapSex (modifyPedigree), [44](#)  
system, [18](#), [43](#)  
  
tieLoops (loops), [35](#)  
toyped, [60](#)  
transferMarkerdata (markers), [37](#)  
trim (modifyPedigree), [44](#)  
twoLoops, [61](#)  
twoMarkerDistribution, [47](#), [61](#)  
  
unrelated (linkdat.utils), [29](#)  
  
write.linkdat (linkdat), [25](#)  
  
Xped, [63](#)