Package 'forrel'

May 7, 2025

Type Package

Title Forensic Pedigree Analysis and Relatedness Inference

Version 1.8.0

Description Forensic applications of pedigree analysis, including likelihood ratios for relationship testing, general relatedness inference, marker simulation, and power analysis. 'forrel' is part of the 'pedsuite', a collection of packages for pedigree analysis, further described in the book 'Pedigree Analysis in R' (Vigeland, 2021, ISBN:9780128244302). Several functions deal specifically with power analysis in missing person cases, implementing methods described in Vigeland et al. (2020) <doi:10.1016/j.fsigen.2020.102376>. Data import from the 'Familias' software (Egeland et al. (2000) <doi:10.1016/S0379-0738(00)00147-X>) is supported through the 'pedFamilias' package.

License GPL (>= 2)

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

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

Depends pedtools (>= 2.8.1), R (>= 4.2.0)

Imports glue, pbapply, pedprobr (>= 1.0.0), ribd (>= 1.7.1), verbalisr (>= 0.7.1)

Suggests ggplot2, ggrepel, ibdsim2, plotly, poibin, scales, testthat

Encoding UTF-8

Language en-GB

LazyData true

RoxygenNote 7.3.2

NeedsCompilation no

Author Magnus Dehli Vigeland [aut, cre] (ORCID: <https://orcid.org/0000-0002-9134-4962>), Thore Egeland [ctb]

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

Repository CRAN

Date/Publication 2025-05-07 11:10:02 UTC

Contents

`checkPairwise` *Check pedigree data for relationship errors*

Description

The `checkPairwise()` function provides a convenient way to check for pedigree errors, given the available marker data. The function calls `ibdEstimate()` to estimate IBD coefficients for all pairs of typed pedigree members, and uses the estimates to test for potential errors. By default, the results are shown in a colour-coded plot (based on `ribd::ibdTriangle()`) where unlikely relationships are easy to spot.

Usage

```
checkPairwise(
  x,
  ids = typedMembers(x),
  includeInbred = FALSE,
  acrossComps = TRUE,
  plotType = c("base", "ggplot2", "plotly", "none"),
  GLRthreshold = 1000,
  pvalThreshold = NULL,
  nsim = 0,
  seed = NULL,
  plot = TRUE,
  verbose = TRUE,
  excludeInbred = NULL,
  ...
)

plotCP(
  cpRes = NULL,
  plotType = c("base", "ggplot2", "plotly"),
  labels = FALSE,
  errtxt = "Potential error",
  seed = NULL,
  ...
)
```

Arguments

x	A ped object or a list of such.
ids	A vector of ID labels; the individuals to include in the check. Default: All typed members of x.
includeInbred	A logical, by default FALSE, indicating if inbred individuals should be excluded from the analysis.
acrossComps	A logical indicating if pairs of individuals in different components should be considered. Default: TRUE.
plotType	Either "base" (default), "ggplot2", "plotly" or "none". Abbreviations are allowed.
GLRthreshold	A positive number, by default 1000. Threshold for the generalised likelihood ratio (see Details). Scores exceeding this are flagged as potential errors in the output table and encircled in the plot.
pvalThreshold	A positive number, or NULL (default). If given, this is used instead of GLRthreshold to identify potential errors. Ignored if nsim = 0.
nsim	A nonnegative number; the number of simulations used to estimate p-values. If 0 (default), this step is skipped.
seed	An integer seed for the random number generator (optional, and only relevant if nsim > 0).

plot	Deprecated. To suppress the triangle plot, use plotType = "none".
verbose	A logical.
excludeInbred	Deprecated; renamed to ´includeInbred´.
...	Further parameters passed on to ribd::ibdTriangle().
cpRes	A data frame: the output from checkPairwise().
labels	A logical (default: FALSE). If TRUE, labels are included in the IBD triangle plot.
errtxt	A character string to use for the error legend.

Details

To identify potential pedigree errors, the function calculates the *generalised likelihood ratio* (GLR) for each pairwise relationship, as explained by Egeland & Vigeland (2025). This compares the likelihood of the estimated coefficients with that of the coefficients implied by the pedigree. By default, relationships whose GLR exceed 1000 are flagged as errors and shown with a circle in the plot. Alternatively, if arguments nsim and pvalThreshold are supplied, the p-value of each score is estimated by simulation, and used as threshold for calling errors.

By default, inbred individuals are excluded from the analysis, since pairwise relationships involving inbred individuals have undefined kappa coefficients (and therefore no position in the triangle). In some cases it may still be informative to include their estimates; set includeInbred = TRUE to enforce this.

Value

If plotType is "none" or "base": A data frame containing both the estimated and pedigree-based IBD coefficients for each pair of typed individuals. The last columns (GLR, pval and err) contain test results using the GLR scores to identify potential pedigree errors.

If plotType is "ggplot2" or "plotly", the plot objects are returned.

References

T. Egeland and M.D. Vigeland, *Kinship cases with partially specified hypotheses*. Forensic Science International: Genetics 78 (2025). doi:10.1016/j.fsigen.2025.103270

See Also

ibdEstimate().

Examples

```
### Example with realistic data

x = avuncularPed() |>
  profileSim(markers = NorwegianFrequencies, seed = 1729)

checkPairwise(x)

### Create an error: sample swap 1 <-> 3
```

```
als = getAlleles(x)
als[c(1,3), ] = als[c(3,1), ]
y = setAlleles(x, alleles = als)

checkPairwise(y)

# Using p-values instead of GLR
nsim = 10 # increase!
checkPairwise(y, nsim = nsim, pvalThreshold = 0.05)

# Plot can be done separately
res = checkPairwise(y, nsim = nsim, pvalThreshold = 0.05, plotType = "none")
plotCP(res, plotType = "base", errtxt = "Not good!")


# Combined plot of pedigree and check results
dev.new(height = 5, width = 8, noRStudioGD = TRUE)
layout(rbind(1:2), widths = 2:3)
plot(y, margins = 2, title = "Swapped 1 - 3")
plotCP(res, labels = TRUE)
```

`exclusionPower` *Power of exclusion*

Description

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

Usage

```
exclusionPower(
  claimPed,
  truePed,
  ids,
  markers = NULL,
  source = "claim",
  disableMutations = NA,
  exactMaxL = Inf,
  nsim = 1000,
  seed = NULL,
  alleles = NULL,
  afreq = NULL,
  knownGenotypes = NULL,
  Xchrom = FALSE,
  plot = FALSE,
  plotMarkers = NULL,
```

```
  verbose = TRUE
)
```

Arguments

claimPed	A ped object (or a list of such), describing the claimed relationship. If a list, the sets of ID labels must be disjoint, that is, all ID labels must be unique.
truePed	A ped object (or a list of such), describing the true relationship. ID labels must be consistent with claimPed.
ids	Individuals available for genotyping.
markers	A vector indicating the names or indices of markers attached to the source pedigree. If NULL (default), then all markers attached to the source pedigree are used. If alleles or afreq is non-NULL, then this parameter is ignored.
source	Either "claim" (default) or "true", deciding which pedigree is used as source for marker data.
disableMutations	This parameter determines how mutation models are treated. Possible values are as follows: • NA (the default): Mutations are disabled only for those markers whose known genotypes are compatible with both claimPed and truePed. This is determined by temporarily removing all mutation models and checking which markers have nonzero likelihood in both alternatives. • TRUE: Mutations are disabled for all markers. • FALSE: No action is done to disable mutations. • A vector containing the names or indices of those markers for which mutations should be disabled.
exactMaxL	A positive integer, or Inf (default). Exact EPs are calculated for markers whose number of alleles is less or equal to exactMaxL; remaining markers are handled by simulation.
nsim	A positive integer; the number of simulations used for markers whose number of alleles exceeds exactMaxL.
seed	An integer seed for the random number generator (optional).
alleles, afreq, Xchrom	If these are given, they are used (together with knownGenotypes) to create a marker object on the fly.
knownGenotypes	A list of triplets (a, b, c), indicating that individual a has genotype b/c. Ignored unless alleles or afreq is non-NULL.
plot	Either a logical or the character "plotOnly". If the latter, a plot is drawn, but no further computations are done.
plotMarkers	A vector of marker names or indices whose genotypes are to be included in the plot.
verbose	A logical.

Details

This function implements the formula for exclusion power as defined and discussed in (Egeland et al., 2014).

It should be noted that `claimPed` and `truePed` may be any (lists of) pedigrees, as long as they both contain the individuals specified by `ids`. In particular, either alternative may have inbred founders (with the same or different coefficients), but this must be set individually for each.

Value

If `plot = "plotOnly"`, the function returns NULL after producing the plot.

Otherwise, the function returns an `EPresult` object, which is essentially a list with the following entries:

- `EPperMarker`: A numeric vector containing the exclusion power of each marker. If the known genotypes of a marker are incompatible with the true pedigree, the corresponding entry is `NA`.
- `EPtotal`: The total exclusion power, computed as `1 - prod(1 - EPperMarker, na.rm = TRUE)`.
- `expectedMismatch`: The expected number of markers giving exclusion, computed as `sum(EPperMarker, na.rm = TRUE)`.
- `distribMismatch`: The probability distribution of the number of markers giving exclusion. This is given as a numeric vector of length `n+1`, where `n` is the number of nonzero elements of `EPperMarker`. The vector has names `0:n`.
- `time`: The total computation time.
- `params`: A list containing the (processed) parameters `ids`, `markers` and `disableMutations`.

Author(s)

Magnus Dehli Vigeland

References

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

Examples

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

# Claim: 'AF' is the father of 'CH'
claim = nuclearPed(father = "AF", children = "CH")

# Attach two (empty) markers
claim = claim |>
  addMarker(alleles = 1:2) |>
  addMarker(alleles = 1:3)
```

```
# Truth: 'AF' and 'CH' are unrelated
true = singletons(c("AF", "CH"))

# EP when both are available for genotyping
exclusionPower(claim, true, ids = c("AF", "CH"))

# EP when the child is typed; homozygous 1/1 at both markers
claim2 = claim |>
  setGenotype(marker = 1:2, id = "CH", geno = "1/1")

exclusionPower(claim2, true, ids = "AF")


##########################################
### Two females claim to be mother and daughter, but are in reality sisters.
### We compute the power of various markers to reject the claim.
##########################################

ids = c("A", "B")
claim = nuclearPed(father = "NN", mother = "A", children = "B", sex = 2)
true = nuclearPed(children = ids, sex = 2)

# SNP with MAF = 0.1:
PE1 = exclusionPower(claimPed = claim, truePed = true, ids = ids,
                     alleles = 1:2, afreq = c(0.9, 0.1))

stopifnot(round(PE1$EPtotal, 5) == 0.00405)

# Tetra-allelic marker with one major allele:
PE2 = exclusionPower(claimPed = claim, truePed = true, ids = ids,
                     alleles = 1:4, afreq = c(0.7, 0.1, 0.1, 0.1))

stopifnot(round(PE2$EPtotal, 5) == 0.03090)


### How does the power change if the true pedigree is inbred?
trueLOOP = halfSibPed(sex2 = 2) |> addChildren(4, 5, ids = ids)

# SNP with MAF = 0.1:
PE3 = exclusionPower(claimPed = claim, truePed = trueLOOP, ids = ids,
                     alleles = 1:2, afreq = c(0.9, 0.1))

# Power almost doubled compared with PE1
stopifnot(round(PE3$EPtotal, 5) == 0.00765)
```

---

expectedLR *Expected likelihood ratio*

Description

This function computes the expected LR for a single marker, in a kinship test comparing two hypothesised relationships between a set of individuals. The true relationship may differ from both hypotheses. Some individuals may already be genotyped, while others are available for typing. The implementation uses `oneMarkerDistribution()` to find the joint genotype distribution for the available individuals, conditional on the known data, in each pedigree.

Usage

```
expectedLR(numeratorPed, denominatorPed, truePed = numeratorPed, ids, marker)
```

Arguments

`numeratorPed`	A ped object.
`denominatorPed`	A ped object.
`truePed`	A ped object.
`ids`	A vector of ID labels corresponding to untyped pedigree members. (These must be members of all three input pedigrees).
`marker`	either a marker object compatible with `numeratorPed`, or the name or index of a marker attached to `numeratorPed`.

Value

A positive number.

Examples

```
#---------
# Curious example showing that ELR may decrease
# by typing additional reference individuals
#---------

# Numerator ped
numPed = nuclearPed(father = "fa", mother = "mo", child = "ch")

# Denominator ped: fa, mo, ch are unrelated. (Hack!)
denomPed = halfSibPed() |> relabel(old = 1:3, new = c("mo", "fa", "ch"))

# Scenario 1: Only mother is typed; genotype 1/2
p = 0.9
m1 = marker(numPed, mo = "1/2", afreq = c("1" = p, "2" = 1-p))
expectedLR(numPed, denomPed, ids = "ch", marker = m1)

1/(8*p*(1-p)) + 1/2 # exact formula

# Scenario 2: Include father, with genotype 1/1
m2 = m1
genotype(m2, id = "fa") = "1/1"
expectedLR(numPed, denomPed, ids = "ch", marker = m2)
```

```
1/(8*p*(1-p)) + 1/(4*p^2) # exact formula
```

`familias` *Import/export from Familias*

Description

Functions for reading .fam files associated with the Familias software for forensic kinship computations.

Deprecated These functions have been moved to a separate package, `pedFamilias`, and will be removed from `forrel` in a future version.

Usage

```
readFam(...)
```

Arguments

`...`	Arguments passed on to the respective `pedFamilias` function.

`findExclusions` *Find markers excluding an identification*

Description

Find markers for which the genotypes of a candidate individual is incompatible with a pedigree

Usage

```
findExclusions(x, id, candidate, removeMut = TRUE)
```

Arguments

`x`	A `ped` object or a list of such.
`id`	A character of length 1; the name of an untyped member of `x`.
`candidate`	A singleton pedigree, with genotypes for the same markers as `x`.
`removeMut`	A logical. If TRUE (default), all mutations models are stripped.

Value

A character vector containing the names of incompatible markers.

Examples

```
db = NorwegianFrequencies[1:5]

# Pedigree with 3 siblings; simulate data for first two
x = nuclearPed(3) |>
  profileSim(ids = 3:4, markers = db, seed = 1)

# Simulate random person
poi = singleton("POI") |>
  profileSim(markers = db, seed = 1)

# Identify incompatible markers
findExclusions(x, id = 5, candidate = poi)   # D21S11

# Inspect
plotPedList(list(x, poi), marker = "D21S11", hatched = typedMembers)
```

FORCE *FORCE panel kinship SNPs*

Description

A data frame describing (a subset of) the FORCE panel of SNPs designed for applications in forensic genetics (Tillmar et al., 2021). The subset included here are the SNPs recommended for kinship analysis. As the original publication did not include allele frequencies, these were downloaded from Ensembl via the biomaRt package. 15 markers were removed as frequency information could not be retrieved.

Usage

```
FORCE
```

Format

A data frame with 3915 rows and 6 columns:

- `CHROM`: Chromosome (1-22)
- `MARKER`: Marker name (rs number)
- `MB`: Physical position in megabases (build GRCh38)
- `A1`: First allele
- `A2`: Second allele
- `FREQ1`: Allele frequency of `A1`

Details

To attach the FORCE markers to a pedigree, use `pedtools::setSNPs()` (see Examples).

Source

Tillmar et al. The FORCE Panel: An All-in-One SNP Marker Set for Confirming Investigative Genetic Genealogy Leads and for General Forensic Applications. Genes. (2021)

Examples

```
x = setSNPs(nuclearPed(), snpData = FORCE)
summary(x)

getMap(x, markers = 1:5)
getFreqDatabase(x, markers = 1:5)
```

---

ibdBootstrap *Bootstrap estimation of IBD coefficients*

Description

This function produces (parametric or nonparametric) bootstrap estimates of the IBD coefficients between two individuals; either the three κ -coefficients or the nine condensed identity coefficients Δ (see `ibdEstimate()`).

Usage

```
ibdBootstrap(
  x = NULL,
  ids = NULL,
  param = NULL,
  kappa = NULL,
  delta = NULL,
  N,
  method = "parametric",
  freqList = NULL,
  plot = TRUE,
  seed = NULL
)
```

Arguments

x	A ped object, or a list of such. If `method = "parametric"`, this is only used to extract the allele frequencies, and can be skipped if `freqList` is provided.
ids	A pair of ID labels.
param	Either NULL (default), "kappa" or "delta". (See below.)
kappa, delta	Probability vectors of length 3 (kappa) or 9 (delta). Exactly one of `param`, `kappa` and `delta` must be non-NULL. If `kappa` and `delta` are both NULL, the appropriate set of coefficients is computed as `ibdEstimate(x, ids, param)`.

N	The number of simulations.
method	Either "parametric" (default) or "nonparametric". Abbreviations are allowed. see Details for more information about each method.
freqList	A list of probability vectors: The allele frequencies for each marker.
plot	A logical, only relevant for bootstraps of kappa. If TRUE, the bootstrap estimates are plotted in the IBD triangle.
seed	An integer seed for the random number generator (optional).

Details

The parameter method controls how bootstrap estimates are obtained in each replication:

- "parametric": new profiles for two individuals are simulated from the input coefficients, followed by a re-estimation of the coefficients.
- "nonparametric": the original markers are sampled with replacement, before the coefficients are re-estimated.

Note that the pedigree itself does not affect the output of this function; the role of x is simply to carry the marker data.

Value

A data frame with N rows containing the bootstrap estimates. The last column, dist, gives the Euclidean distance to the original coefficients (either specified by the user or estimated from the data), viewed as a point in R^3 (kappa) or R^9 (delta).

See Also

ibdEstimate()

Examples

```
# Frequency list of 15 standard STR markers
freqList = NorwegianFrequencies[1:15]

# Number of bootstrap simulations (increase!)
N = 5

# Bootstrap estimates for kappa of full siblings
boot1 = ibdBootstrap(kappa = c(0.25, .5, .25), N = N, freqList = freqList)
boot1

# Mean deviation
mean(boot1$dist)

# Same, but with the 9 identity coefficients.
delta = c(0, 0, 0, 0, 0, 0, .25, .5, .25)
boot2 = ibdBootstrap(delta = delta, N = N, freqList = freqList)

# Mean deviation
```

```
mean(boot2$dist)

#### Non-parametric bootstrap.
# Requires `x` and `ids` to be provided

x = nuclearPed(2)
x = markerSim(x, ids = 3:4, N = 50, alleles = 1:10, seed = 123)

bootNP = ibdBootstrap(x, ids = 3:4, param = "kappa", method = "non", N = N)

# Parametric bootstrap can also be done with this syntax
bootP = ibdBootstrap(x, ids = 3:4, param = "kappa", method = "par", N = N)
```

`ibdEstimate` *Pairwise relatedness estimation*

Description

Estimate the IBD coefficients $\kappa = (\kappa_0, \kappa_1, \kappa_2)$ or the condensed identity coefficients $\Delta = (\Delta_1, ..., \Delta_9)$ between a pair (or several pairs) of pedigree members, using maximum likelihood methods. Estimates of κ may be visualised with `showInTriangle()`.

Usage

```
ibdEstimate(
  x,
  ids = typedMembers(x),
  param = c("kappa", "delta"),
  acrossComps = TRUE,
  markers = NULL,
  start = NULL,
  tol = sqrt(.Machine$double.eps),
  beta = 0.5,
  sigma = 0.5,
  contourPlot = FALSE,
  levels = NULL,
  maxval = FALSE,
  verbose = TRUE
)
```

Arguments

`x`	A ped object or a list of such.
`ids`	Either a vector with ID labels, or a data frame/matrix with two columns, each row indicating a pair of individuals. The entries are coerced to characters, and must match uniquely against the ID labels of `x`. By default, all pairs of genotyped members of `x` are included.

param	Either "kappa" (default) or "delta"; indicating which set of coefficients should be estimated.
acrossComps	A logical indicating if pairs of individuals in different components should be included. Default: TRUE.
markers	A vector with names or indices of markers attached to x, indicating which markers to include. By default, all markers are used.
start	A probability vector (i.e., with nonnegative entries and sum 1) of length 3 (if param = "kappa") or 9 (if param = "delta"), indicating the initial value of for the optimisation. By default, start is set to (1/3, 1/3, 1/3) if param = "kappa" and (1/9, ..., 1/9) if param = "delta".
tol, beta, sigma	Control parameters for the optimisation routine; can usually be left untouched.
contourPlot	A logical. If TRUE, contours of the log-likelihood function are plotted overlaying the IBD triangle.
levels	(Only relevant if contourPlot = TRUE.) A numeric vector of levels at which to draw contour lines. If NULL (default), the levels are chosen automatically.
maxval	A logical. If TRUE, the maximum log-likelihood value is included in the output. Default: FALSE
verbose	A logical.

Details

It should be noted that this procedure estimates the *realised* identity coefficients of each pair, i.e., the actual fractions of the autosomes in each IBD state. These may deviate substantially from the theoretical pedigree coefficients.

Maximum likelihood estimation of relatedness coefficients originates with Thompson (1975). Optimisation of κ is done in the (κ_0, κ_2)-plane and restricted to the triangle defined by

$$\kappa_0 \geq 0, \kappa_2 \geq 0, \kappa_0 + \kappa_2 \leq 1$$

Optimisation of Δ is done in unit simplex of R^8, using the first 8 coefficients.

The implementation optimises the log-likelihood using a projected gradient descent algorithm, combined with a version of Armijo line search.

When param = "kappa", the output may be fed directly to showInTriangle() for visualisation.

Value

An object of class ibdEst, which is basically a data frame with either 6 columns (if param = "kappa") or 12 columns (if param = "delta"). The first three columns are id1 (label of first individual), id2 (label of second individual) and N (the number of markers with no missing alleles). The remaining columns contain the coefficient estimates. If maxval = T, a column named maxloglik is added at the end.

Author(s)

Magnus Dehli Vigeland

References

- E. A. Thompson (1975). *The estimation of pairwise relationships.* Annals of Human Genetics 39.
- E. A. Thompson (2000). *Statistical Inference from Genetic Data on Pedigrees.* NSF-CBMS Regional Conference Series in Probability and Statistics. Volume 6.

See Also

ibdBootstrap()

Examples

```
### Example 1: Siblings

# Create pedigree and simulate 100 markers
x = nuclearPed(2) |> markerSim(N = 100, alleles = 1:4, seed = 123)
x

# Estimate kappa (expectation: (0.25, 0.5, 0.25)
k = ibdEstimate(x, ids = 3:4)
k

# Visualise estimate
showInTriangle(k, labels = TRUE)

# Contour plot of the log-likelihood function
ibdEstimate(x, ids = 3:4, contourPlot = TRUE)


### Example 2: Full sib mating
y = fullSibMating(1) |>
  markerSim(ids = 5:6, N = 1000, alleles = 1:10, seed = 123)

# Estimate the condensed identity coefficients
ibdEstimate(y, param = "delta")

# Exact coefficient by `ribd`:
ribd::condensedIdentity(y, 5:6, simplify = FALSE)
```

---

ibdLoglik *Pairwise IBD likelihood*

Description

Given genotype data from two individuals, computes the log-likelihood of a single set of IBD coefficients, either `kappa` = $(\kappa_0, \kappa_1, \kappa_2)$ or the Jacquard coefficients `delta` = $(\Delta_1, ..., \Delta_9)$. The `ibdLoglikFUN` version returns an efficient *function* for computing such likelihoods, suitable for optimisations such as in ibdEstimate().

Usage

```
ibdLoglik(x = NULL, ids = NULL, kappa = NULL, delta = NULL)

ibdLoglikFUN(x, ids, input = c("kappa", "kappa02", "delta"))
```

Arguments

x	A ped object or a list of such.
ids	A vector of ID labels.
kappa	A probability vector of length 3.
delta	A probability vector of length 9.
input	Either "kappa", "kappa02" or "delta". See Value.

Value

`ibdLoglik()` returns a single number; the total log-likelihood over all markers included.

`ibdLoglikFUN()` returns a function for computing such log-likelihoods. The function takes a single input vector p, whose interpretation depends on the `input` parameter:

- "kappa": p is expected to be a set of kappa coefficients $(\kappa_0, \kappa_1, \kappa_2)$.
- "kappa02": p should be a vector of length 2 containing the coefficients κ_0 and κ_2. This is sometimes a convenient shortcut when working in the IBD triangle.
- "delta": Expects p to be a set of condensed Jacquard coefficients $(\Delta_1, ..., \Delta_9)$.

Examples

```
# Siblings typed with 10 markers
x = nuclearPed(2) |> markerSim(N = 10, alleles = 1:4)

# Calculate log-likelihood at a single point
k = c(0.25, 0.5, 0.25)
ibdLoglik(x, ids = 3:4, kappa = k)

# Or first get a function, and then apply it
llFun = ibdLoglikFUN(x, ids = 3:4, input = "kappa")
llFun(k)
```

kinshipLR *Likelihood ratios for kinship testing*

Description

This function computes likelihood ratios (LRs) for a list of pedigrees. One of the pedigrees (the last one, by default) is designated as 'reference', to be used in the denominator in all LR calculations. To ensure that all pedigrees use the same data set, one of the pedigrees may be chosen as 'source', from which data is transferred to all the other pedigrees.

Usage

```
kinshipLR(
  ...,
  ref = NULL,
  source = NULL,
  markers = NULL,
  likArgs = NULL,
  linkageMap = NULL,
  keepMerlin = NULL,
  verbose = FALSE
)
```

Arguments

...	Pedigree alternatives. Each argument should be either a single ped object or a list of such. The pedigrees may be named; otherwise they are assigned names "H1", "H2", ... automatically. It is also possible to pass a single `list` containing all the pedigrees.
ref	An index or name indicating which of the input pedigrees should be used as "reference pedigree", i.e., used in the denominator of each LR. If NULL (the default), the last pedigree is used as reference.
source	An index or name designating one of the input pedigrees as source for marker data. If given, marker data is transferred from this to all the other pedigrees (replacing any existing markers). The default action (`source = NULL`) is as follows: If all pedigree have attached markers, no transfers are done. If exactly one of the pedigrees have attached markers, these are transferred to the others. all other cases give an error.
markers	A vector of marker names or indices indicating which markers should be included. If NULL (the default) all markers are used.
likArgs	An optional list of arguments to be passed to `pedprobr::likelihood()`, e.g. `likArgs = list(special = TRUE)`.
linkageMap	If this is non-NULL, the markers are interpreted as being linked, and likelihoods will be computed by an external call to MERLIN. The supplied object should be either:

- a data frame, whose first three columns must be (i) chromosome (ii) marker name (iii) centiMorgan position, or
- a map object created with `ibdsim2::uniformMap()` or `ibdsim2::loadMap()`. This will internally be applied to the attached markers to produce a suitable data frame as above.

`keepMerlin` Either NULL (default) or the path to an existing folder. If given, MERLIN files are stored here, typically for debugging purposes.

`verbose` A logical.

Details

By default, all markers are assumed to be unlinked. To accommodate linkage, a genetic map may be supplied with the argument `linkageMap`. This requires the software MERLIN to be installed.

Value

A `LRresult` object, which is essentially a list with entries

- `LRtotal` : A vector of length `L`, where `L` is the number of input pedigrees. The i'th entry is the total LR (i.e., the product over all markers) comparing pedigree `i` to the reference pedigree. The entry corresponding to the reference will always be 1.
- `LRperMarker` : A numerical matrix, where the i'th column contains the marker-wise LR values comparing pedigree `i` to the reference. The product of all entries in a column should equal the corresponding entry in `LRtotal`.
- `likelihoodsPerMarker` : A numerical matrix of the same dimensions as `LRperMarker`, but where the entries are likelihood of each pedigree for each marker.
- `time` : Elapsed time

Author(s)

Magnus Dehli Vigeland and Thore Egeland

See Also

`LRpower()`, `pedprobr::likelihood()`, `pedprobr::likelihoodMerlin()`

Examples

```
### Example 1: Full vs half sibs

# Simulate 5 markers for a pair of full sibs
ids = c("A", "B")
sibs = nuclearPed(children = ids)
sibs = simpleSim(sibs, N = 5, alleles = 1:4, ids = ids, seed = 123)

# Create two alternative hypotheses
halfsibs = relabel(halfSibPed(), old = 4:5, new = ids)
unrel = singletons(c("A", "B"))
```

```
# Compute LRs. By default, the last ped is used as reference
kinshipLR(sibs, halfsibs, unrel)

# Input pedigrees can be named, reflected in the output
kinshipLR(S = sibs, H = halfsibs, U = unrel)

# Select non-default reference (by index or name)
kinshipLR(S = sibs, H = halfsibs, U = unrel, ref = "H")

# Alternative syntax: List input
peds = list(S = sibs, H = halfsibs, U = unrel)
kinshipLR(peds, ref = "H", source = "S", verbose = TRUE)

# Detailed results
res = kinshipLR(peds)
res$LRperMarker
res$likelihoodsPerMarker


### Example 2: Separating grandparent/halfsib/uncle-nephew

# Requires ibdsim2 and MERLIN
if(requireNamespace("ibdsim2", quietly = TRUE) && pedprobr::checkMerlin()) {

  # Load recombination map
  map = ibdsim2::loadMap("decode19", uniform = TRUE)   # unif for speed

  # Define pedigrees
  ids = c("A", "B")
  H = relabel(halfSibPed(),   old = c(4,5), new = ids)
  U = relabel(avuncularPed(), old = c(3,6), new = ids)
  G = relabel(linearPed(2),   old = c(1,5), new = ids)

  # Attach FORCE panel of SNPs to G
  G = setSNPs(G, FORCE[1:10, ])  # use all for better results

  # Simulate recombination pattern in G
  ibd = ibdsim2::ibdsim(G, N = 1, ids = ids, map = map)

  # Simulate genotypes conditional on pattern
  G = ibdsim2::profileSimIBD(G, ibdpattern = ibd)

  # Compute LR (genotypes are automatically transferred to H and U)
  kinshipLR(H, U, G, linkageMap = map)
}
```

LRpower *Power simulation for kinship LR*

Description

This function uses simulations to estimate the likelihood ratio (LR) distribution in a given kinship testing scenario. In the most general setting, three pedigrees are involved: the two pedigrees being compared, and the true relationship (which may differ from the other two). A subset of individuals are available for genotyping. Some individuals may already be genotyped; all simulations are then conditional on these.

Usage

```
LRpower(
  numeratorPed,
  denominatorPed,
  truePed = numeratorPed,
  ids,
  markers = NULL,
  source = "true",
  nsim = 1,
  threshold = NULL,
  disableMutations = NA,
  alleles = NULL,
  afreq = NULL,
  Xchrom = FALSE,
  knownGenotypes = NULL,
  plot = FALSE,
  plotMarkers = NULL,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

`numeratorPed`, `denominatorPed`
: ped objects (or lists of such), describing the two relationships under comparison.

`truePed`
: A ped object (or a list of such), describing the true relationship. By default equal to `numeratorPed`.

`ids`
: Individuals available for genotyping.

`markers`
: A vector indicating the names or indices of markers attached to the source pedigree. If NULL (default), then all markers attached to the source pedigree are used. If `alleles` or `afreq` is non-NULL, then this parameter is ignored.

`source`
: Either "true" (default), "numerator" or "denominator", indicating which pedigree is used as source for marker data.

`nsim`
: A positive integer: the number of simulations.

`threshold`
: A numeric vector with one or more positive numbers used as LR thresholds.

`disableMutations`
: Not implemented yet.

`alleles`, `afreq`, `Xchrom`
: If these are given, they are used (together with `knownGenotypes`) to create a marker object on the fly.

`knownGenotypes`
: A list of triplets `(a, b, c)`, indicating that individual `a` has genotype `b/c`. Ignored unless `alleles` or `afreq` is non-NULL.

`plot`
: Either a logical or the character "plotOnly". If the latter, a plot is drawn, but no further computations are done.

`plotMarkers`
: A vector of marker names or indices whose genotypes are to be included in the plot.

`seed`
: An integer seed for the random number generator (optional).

`verbose`
: A logical.

Value

A `LRpowerResult` object, which is essentially a list with the following entries:

- `LRperSim`: A numeric vector of length `nsim` containing the total LR for each simulation.
- `meanLRperMarker`: The mean LR per marker, over all simulations.
- `meanLR`: The mean total LR over all simulations.
- `meanLogLR`: The mean total `log10(LR)` over all simulations.
- `IP`: A named numeric of the same length as `threshold`. For each element of `threshold`, the fraction of simulations resulting in a LR exceeding the given number.
- `time`: The total computation time.
- `params`: A list containing the input parameters `missing`, `markers`, `nsim`, `threshold` and `disableMutations`

Examples

```
# Paternity LR of siblings
ids = c("A", "B")
truth = nuclearPed(children = ids)
claim = nuclearPed(fa = "A", mo = "NN", children = "B")
unrel = singletons(ids)

# Simulation parameters
nsim = 10   # increase!
thresh = 1

# Simulation 1:
als = 1:5
afr = runif(5)
afr = afr/sum(afr)

pow1 = LRpower(claim, unrel, truth, ids = ids, nsim = nsim,
               threshold = thresh, alleles = als, afreq = afr,
               seed = 123)
pow1
```

```
# Simulation 2: Same, but using an attached marker
truth = addMarker(truth, alleles = als, afreq = afr)

pow2 = LRpower(claim, unrel, truth, ids = ids, nsim = nsim,
               threshold = thresh, markers = 1, seed = 123)

stopifnot(identical(pow1$LRperSim, pow2$LRperSim))


# True pedigree has inbred founders
truth2 = setFounderInbreeding(truth, value = 0.5)

pow3 = LRpower(claim, unrel, truth2, ids = ids, nsim = nsim,
               threshold = thresh, markers = 1, seed = 123) # plot = TRUE
pow3
```

---

markerSim *Marker simulation*

Description

Simulates marker genotypes conditional on the pedigree structure and known genotypes. Note: This function simulates independent realisations at a single locus. Equivalently, it can be thought of as independent simulations of identical, unlinked markers. For simulating profiles for a set of different markers, see profileSim().

Usage

```
markerSim(
  x,
  N = 1,
  ids = NULL,
  alleles = NULL,
  afreq = NULL,
  mutmod = NULL,
  rate = NULL,
  partialmarker = NULL,
  loopBreakers = NULL,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

x	A ped object or a list of such.
N	A positive integer: the number of markers to be simulated.

ids	A vector indicating the pedigree members whose genotypes should be simulated. Alternatively, a function taking x as input and returning a character vector of ID labels. Default: All individuals.
alleles	(Only if partialmarker is NULL.) A vector with allele labels. If NULL, the following are tried in order: • names(afreq) • 'seq_along(afreq)' • 1:2 (Fallback if both alleles and afreq are NULL.)
afreq	(Only if partialmarker is NULL.) A numeric vector with allele frequencies, possibly named with allele labels.
mutmod, rate	Arguments specifying a mutation model, passed on to pedtools::marker() (see there for explanations).
partialmarker	Either NULL (resulting in unconditional simulation), a marker object (on which the simulation should be conditioned) or the name (or index) of a marker attached to x.
loopBreakers	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 pedtools::breakLoops().
seed	An integer seed for the random number generator (optional).
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 ped object equal to x except its MARKERS 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

profileSim(), simpleSim()

Examples

```
x = nuclearPed(2)

# Unconditional simulation
markerSim(x, N = 2, alleles = 1:3)

# Conditional on one child being homozygous 1/1
x = addMarker(x, "3" = "1/1", alleles = 1:3)
markerSim(x, N = 2, partialmarker = 1)
markerSim(x, N = 1, ids = 4, partialmarker = 1, verbose = FALSE)
```

---

markerSimParametric *Simulate marker data given IBD coefficients*

Description

This function simulates genotypes for two individuals given their IBD distribution, for N identical markers.

Usage

```
markerSimParametric(
  kappa = NULL,
  delta = NULL,
  states = NULL,
  N = 1,
  alleles = NULL,
  afreq = NULL,
  seed = NULL,
  returnValue = c("singletons", "alleles", "genotypes", "internal")
)
```

Arguments

kappa	A probability vector of length 3, giving a set of realised kappa coefficients (between two noninbred individuals).
delta	A probability vector of length 9, giving a set of condensed identity coefficients (Jacquard coefficients).
states	An integer vector of length N, with entries in 1-9. Each entry gives the identity state of the corresponding marker. (See details.)
N	A positive integer: the number of independent markers to be simulated.
alleles	A vector with allele labels. If NULL, the following are tried in order: • names(afreq) • 'seq_along(afreq)' • 1:2 (fallback if both alleles and afreq are NULL)

afreq	A numeric vector with allele frequencies, possibly named with allele labels.
seed	An integer seed for the random number generator (optional).
returnValue	Either "singleton" (default) or "alleles". (see Value).

Details

Exactly one of kappa, delta and states must be given; the other two should remain NULL.

If states is given, it explicitly determines the condensed identity state at each marker. The states are described by integers 1-9, using the tradition order introduced by Jacquard.

If kappa is given, the states are generated by the command states = sample(9:7, size = N, replace = TRUE, prob = kappa). (Note that identity states 9, 8, 7 correspond to IBD status 0, 1, 2, respectively.)

If delta is given, the states are generated by the command states = sample(1:9, size = N, replace = TRUE, prob = delta).

Value

The output depends on the value of the returnValue parameter:

- "singletons": a list of two singletons with the simulated marker data attached.
- "alleles": a list of four vectors of length N, named a, b, c and d. These contain the simulated alleles, where a/b and c/d are the genotypes of the to individuals.
- "genotypes": a list of two vectors of length N, containing the simulated genotypes. Identical to paste(a, b, sep = "/") and paste(c, d, sep = "/"), where a, b, c, d are the vectors returned when returnValue == "alleles".
- "internal": similar to "alleles", but using the index integer of each allele. (This option is mostly for internal use.)

Examples

```
# MZ twins
markerSimParametric(kappa = c(0,0,1), N = 5, alleles = 1:10)

# Equal distribution of states 1 and 2
markerSimParametric(delta = c(.5,.5,0,0,0,0,0,0,0), N = 5, alleles = 1:10)

# Force a specific sequence of states
markerSimParametric(states = c(1,2,7,8,9), N = 5, alleles = 1:10)
```

`missingPersonEP` *Exclusion power for missing person cases*

Description

This is a special case of `exclusionPower()` for use in missing person cases. The function computes the probability that a random person is genetically incompatible with the typed relatives of the missing person.

Usage

```
missingPersonEP(
  reference,
  missing,
  markers = NULL,
  disableMutations = NA,
  verbose = TRUE
)
```

Arguments

`reference`	A ped object with attached markers.
`missing`	The ID label of the missing pedigree member.
`markers`	A vector indicating the names or indices of markers attached to the source pedigree. If NULL (default), then all markers attached to the source pedigree are used. If `alleles` or `afreq` is non-NULL, then this parameter is ignored.
`disableMutations`	This parameter determines how mutation models are treated. Possible values are as follows: • `NA` (the default): Mutations are disabled only for those markers whose known genotypes are consistent with the pedigree. This is determined by temporarily removing all mutation models and checking which markers have nonzero likelihood. • `TRUE`: Mutations are disabled for all markers. This will result in an error if any markers are inconsistent. • `FALSE`: No action is done to disable mutations. • A vector containing the names or indices of those markers for which mutations should be disabled.
`verbose`	A logical.

Details

This function is identical to `randomPersonEP()`, but with different argument names. This makes it consistent with `missingPersonIP()` and the other 'missing person' functions.

Value

The `EPresult` object returned by `exclusionPower()`.

See Also

`randomPersonEP()`, `exclusionPower()`

Examples

```
# Four siblings; the fourth is missing
x = nuclearPed(4)

# Remaining sibs typed with 4 triallelic markers
x = markerSim(x, N = 4, ids = 3:5, alleles = 1:3, seed = 577, verbose = FALSE)

# Add marker with inconsistency in reference genotypes
# (by default this is ignored by `missingPersonEP()`)
x = addMarker(x, "3" = "1/1", "4" = "2/2", "5" = "3/3")

# Compute exclusion power statistics
missingPersonEP(x, missing = 6)
```

---

missingPersonIP *Inclusion power for missing person cases*

Description

This function simulates the LR distribution for the true missing person in a reference family. The output contains both the total and marker-wise LR of each simulation, as well as various summary statistics. If a specific LR threshold is given, the *inclusion power* is computed as the probability that LR exceeds the threshold.

Usage

```
missingPersonIP(
  reference,
  missing,
  markers,
  nsim = 1,
  threshold = NULL,
  disableMutations = NA,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

reference	A ped object with attached markers.
missing	The ID label of the missing pedigree member.
markers	A vector indicating the names or indices of markers attached to the source pedigree. If NULL (default), then all markers attached to the source pedigree are used. If alleles or afreq is non-NULL, then this parameter is ignored.
nsim	A positive integer: the number of simulations
threshold	A numeric vector with one or more positive numbers used as the likelihood ratio thresholds for inclusion
disableMutations	This parameter determines how mutation models are treated. Possible values are as follows: • NA (the default): Mutations are disabled only for those markers whose known genotypes are consistent with the pedigree. This is determined by temporarily removing all mutation models and checking which markers have nonzero likelihood. • TRUE: Mutations are disabled for all markers. This will result in an error if any markers are inconsistent. • FALSE: No action is done to disable mutations. • A vector containing the names or indices of those markers for which mutations should be disabled.
seed	An integer seed for the random number generator (optional).
verbose	A logical.

Value

A mpIP object, which is essentially a list with the following entries:

- LRperSim: A numeric vector of length nsim containing the total LR for each simulation.
- meanLRperMarker: The mean LR per marker, over all simulations.
- meanLR: The mean total LR over all simulations.
- meanLogLR: The mean total log10(LR) over all simulations.
- IP: A named numeric of the same length as threshold. For each element of threshold, the fraction of simulations resulting in a LR exceeding the given number.
- time: The total computation time.
- params: A list containing the input parameters missing, markers, nsim, threshold and disableMutations

Examples

```
# Four siblings; the fourth is missing
x = nuclearPed(4)

# Remaining sibs typed with 5 triallelic markers
x = markerSim(x, N = 5, ids = 3:5, alleles = 1:3, seed = 123, verbose = FALSE)
```

```
# Compute inclusion power statistics
ip = missingPersonIP(x, missing = 6, nsim = 5, threshold = c(10, 100))
ip

# LRs from each simulation
ip$LRperSim
```

`missingPersonLR` *Likelihood ratio calculation for missing person identification*

Description

This is a wrapper function for `kinshipLR()` for the special case of missing person identification. A person of interest (POI) is matched against a reference dataset containing genotypes of relatives of the missing person.

Usage

```
missingPersonLR(reference, missing, poi = NULL, verbose = TRUE, ...)
```

Arguments

`reference`	A ped object with attached markers.
`missing`	The ID label of the missing member of `reference`.
`poi`	A `singleton` object, or NULL. If NULL, and `missing` is genotyped, this data is extracted and used as `poi`.
`verbose`	A logical.
`...`	Optional parameters to be passed on to `kinshipLR()`.

Details

Note that this function accepts two forms of input:

1. With `poi` a typed singleton. This is the typical use case, when you want to compute the LR for some person of interest.
2. With `poi = NULL`, but `missing` being genotyped. The data for `missing` is then extracted as a singleton POI. This is especially useful in simulation procedures, e.g., for simulating the LR distribution of the true missing person.

See Examples for illustrations of both cases.

Value

The `LRresult` object returned by `kinshipLR()`, but without the trivial `H2:H2` comparison.

Examples

```
#-----------------------------------------------
# Example: Identification of a missing grandchild
#-----------------------------------------------

# Database with 5 STR markers (increase to make more realistic)
db = NorwegianFrequencies[1:5]

# Pedigree with missing person (MP); grandmother is genotyped
x = linearPed(2) |>
  relabel(old = 5, new = "MP") |>
  profileSim(markers = db, ids = "2", seed = 123)


### Scenario 1: Unrelated POI --------------------

# Generate random unrelated profile
poi = singleton("POI") |>
  profileSim(markers = db, seed = 1234)

# Compute LR
lr = missingPersonLR(x, missing = "MP", poi = poi)
lr
lr$LRperMarker


### Scenario 2: POI is the missing person --------
# A small simulation example

# Simulate profiles for MP conditional on the grandmother
N = 10
y = profileSim(x, N = N, ids = "MP", seed = 12345)

# Compute LRs for each sim
LRsims = lapply(y, missingPersonLR, missing = "MP", verbose = FALSE)

# Plot distribution
LRtotal = sapply(LRsims, function(a) a$LRtotal)
plot(density(LRtotal))

# LRs for each marker
LRperMarker = sapply(LRsims, function(a) a$LRperMarker)
LRperMarker

# Overlaying marker-wise density plots (requires tidyverse)
# library(tidyverse)
# t(LRperMarker) |> as_tibble() |> pivot_longer(everything()) |>
#   ggplot() + geom_density(aes(value, fill = name), alpha = 0.6)
```

`missingPersonPlot` *Missing person plot*

Description

Visualises the competing hypotheses of a family reunion case. A plot with two panels is generated. The left panel shows a pedigree in which the *person of interest* (POI) is identical to the *missing person* (MP). The right panel shows the situation where these two are unrelated. See Details for further explanations.

Usage

```
missingPersonPlot(
  reference,
  missing,
  labs = labels(reference),
  marker = NULL,
  hatched = typedMembers(reference),
  MP.label = "MP",
  POI.label = "POI",
  MP.col = "#FF9999",
  POI.col = "lightgreen",
  POI.sex = getSex(reference, missing),
  POI.hatched = NULL,
 titles = c(expression(H[1] * ": POI = MP"), expression(H[2] * ": POI unrelated")),
  width = NULL,
  cex = 1.2,
  ...
)
```

Arguments

`reference`	A `pedtools::ped()` object.
`missing`	The ID label of the missing pedigree member.
`labs`	A character vector with labels for the pedigree members. See `pedtools::plot.ped()`.
`marker`	Optional vector of marker indices to be included in the plot.
`hatched`	A vector of ID labels indicating who should appear with hatched symbols in the plot. By default, all typed members.
`MP.label`, `POI.label`	Custom labels of the missing person and the POI. Default: "MP" and "POI".
`MP.col`, `POI.col`	Fill colours for MP and POI.
`POI.sex`	The sex of POI. This defaults to that of the missing person, but may be set explicitly. This is particularly useful when the missing person has unknown sex.
`POI.hatched`	Deprecated (ignored).

titles	A character of length 2, with subtitles for the two frames.
width	A positive number controlling the width of the plot. More specifically this number is the relative width of the reference pedigree, compared to a singleton.
cex	Expansion factor for pedigree symbols and font size.
...	Extra parameters passed on to pedtools::plotPedList().

Details

A standard family reunification case involves the following ingredients:

- A reference family with a single missing person ("MP").
- Some of the family members have been genotyped
- A person of interest ("POI") is to be matched against the reference family

After genotyping of POI, the genetic evidence is typically assessed by computing the likelihood ratio of the following hypotheses:

- H1: POI is MP
- H2: POI is unrelated to the family

The goal of this function is to illustrate the above hypotheses, using labels, colours and shading to visualise the different aspects of the situation.

This function cannot handle cases with more complicated hypotheses (e.g. multiple missing persons, or where H2 specifies a different relationship). However, as it is basically a wrapper of pedtools::plotPedList(), an interested user should be able to extend the source code to such cases without too much trouble.

Value

None

Examples

```
x = nuclearPed(father = "fa", mother = "mo", children = c("b1", "b2"))

# Default plot
missingPersonPlot(x, missing = "b2")

# Open in separate window; explore various options
missingPersonPlot(x,
                  missing = "b2",
                  hatched = "b1",
                  deceased = c("fa", "mo"),
                  cex = 1.5,      # larger symbols and labels (see ?par())
                  cex.main = 1.3, # larger frame titles (see ?par())
                  dev.width = 7,  # device width (see ?plotPedList())
                  dev.height = 3  # device height (see ?plotPedList())
                  )
```

`MPPsims` *Missing person power simulations*

Description

Estimate the exclusion/inclusion power for various selections of available individuals.

Usage

```
MPPsims(
  reference,
  missing = "MP",
  selections,
  ep = TRUE,
  ip = TRUE,
  addBaseline = TRUE,
  nProfiles = 1,
  lrSims = 1,
  thresholdIP = NULL,
  disableMutations = NA,
  numCores = 1,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

`reference`	A connected `ped` object, or a list of pedigrees. In the latter case, the list must have the same length as `selections`.
`missing`	The ID label of the missing pedigree member.
`selections`	A list of pedigree member subsets. In the special case that all subsets consist of a single individual, `selections` can be given as a simple vector.
`ep`	A logical: Estimate the exclusion power? (Default: TRUE)
`ip`	A logical: Estimate the inclusion power? (Default: TRUE)
`addBaseline`	A logical. If TRUE (default) an *empty* selection, named "Baseline", is added as the first element of `selection`.
`nProfiles`	The number of profile simulations for each selection.
`lrSims, thresholdIP`	Parameters passed onto `missingPersonIP()`.
`disableMutations`	This parameter determines how mutation models are treated. Possible values are as follows: • `NA` (the default): Mutations are disabled only for those markers whose known genotypes are consistent with the pedigree. This is determined by temporarily removing all mutation models and checking which markers have nonzero likelihood.

- TRUE: Mutations are disabled for all markers. This will result in an error if any markers are inconsistent.
- FALSE: No action is done to disable mutations.
- A vector containing the names or indices of those markers for which mutations should be disabled.

numCores	The number of cores used for parallelisation, by default 1.
seed	An integer seed for the random number generator (optional).
verbose	A logical.

Value

An object of class "MPPsim", which is basically a list with one entry for each element of selections. Each entry has elements ep and ip, each of which is a list of length nProfiles.

The output object has various attributes reflecting the input. Note that reference and selection may differ slightly from the original input, since they may be modified during the function run. (For instance, a "Baseline" entry is added to selection if addBaseline is TRUE.) The crucial point is that the output attributes correspond exactly to the output data.

- reference (always a list, of the same length as the selections attribute
- selections
- nProfiles,lrSims,thresholdIP,seed (as in the input)
- totalTime (the total time used)

Examples

```
x = nuclearPed(fa = "Gf", mo = "Gm", children = c("Uncle", "Mother"), sex = 1:2)
x = addChildren(x, fa = "Father", mo = "Mother", nch = 3, sex = c(1,2,1),
                id = c("S1", "S2", "MP"))
x = addSon(x, "Father", id = "HS")

# Brother S1 is already genotyped with a marker with 4 alleles
x = addMarker(x, S1 = "1/2", alleles = 1:4)

# Alternatives for additional genotyping
sel = list("Father", "S2", "HS", c("Gm", "Uncle"))

plot(x, marker = 1, hatched = sel)

# Simulate
simData = MPPsims(x, selections = sel, nProfiles = 2, lrSims = 2)

# Power plot
powerPlot(simData, type = 3)


### With  mutations
# Add inconsistent marker
x = addMarker(x, S1 = "1/2", Father = "3/3", alleles = 1:4)
```

```
# Set mutation models for both
mutmod(x, 1:2) = list("equal", rate = 0.1)

# By default mutations are disabled for consistent markers
MPPsims(x, selections = "Father", addBaseline = FALSE)

# Don't disable anything
MPPsims(x, selections = "Father", addBaseline = FALSE,
        disableMutations = FALSE)


# Disable all mutation models. SHOULD GIVE ERROR FOR SECOND MARKER
# MPPsims(x, selections = "Father", addBaseline = FALSE,
#         disableMutations = TRUE)
```

`NorwegianFrequencies` *Norwegian STR frequencies*

Description

A database of Norwegian allele frequencies for 35 STR markers.

Usage

```
NorwegianFrequencies
```

Format

A list of length 35. Each entry is a numerical vector summing to 1, named with allele labels.

The following markers are included:

- `D3S1358`: 12 alleles
- `TH01`: 10 alleles
- `D21S11`: 26 alleles
- `D18S51` : 23 alleles
- `PENTA_E`: 21 alleles
- `D5S818`: 9 alleles
- `D13S317`: 9 alleles
- `D7S820`: 19 alleles
- `D16S539`: 9 alleles
- `CSF1PO`: 11 alleles
- `PENTA_D`: 24 alleles

- VWA: 12 alleles
- D8S1179: 12 alleles
- TPOX: 9 alleles
- FGA: 25 alleles
- D19S433: 17 alleles
- D2S1338: 13 alleles
- D10S1248: 9 alleles
- D1S1656: 17 alleles
- D22S1045: 9 alleles
- D2S441: 13 alleles
- D12S391: 23 alleles
- SE33: 55 alleles
- D7S1517: 11 alleles
- D3S1744: 8 alleles
- D2S1360: 10 alleles
- D6S474: 6 alleles
- D4S2366: 7 alleles
- D8S1132: 12 alleles
- D5S2500: 8 alleles
- D21S2055: 18 alleles
- D10S2325: 10 alleles
- D17S906: 78 alleles
- APOAI1: 41 alleles
- D11S554: 51 alleles

Source

Dupuy et al. (2013): *Frequency data for 35 autosomal STR markers in a Norwegian, an East African, an East Asian and Middle Asian population and simulation of adequate database size*. Forensic Science International: Genetics Supplement Series, Volume 4 (1).

powerPlot *Exclusion/inclusion power plots*

Description

This function offers four different visualisations of exclusion/inclusion powers, particularly for missing person cases. Output from `MPPsims()` may be fed directly as input to this function. The actual plotting is done with `ggplot2`.

Usage

```
powerPlot(
  ep,
  ip = NULL,
  type = 1,
  majorpoints = TRUE,
  minorpoints = TRUE,
  ellipse = FALSE,
  col = NULL,
  labs = NULL,
  jitter = FALSE,
  alpha = 1,
  stroke = 1.5,
  shape = "circle",
  size = 1,
  hline = NULL,
  vline = NULL,
  xlim = NULL,
  ylim = NULL,
  xlab = NULL,
  ylab = NULL
)
```

Arguments

`ep`, `ip`	Lists of equal length, with outputs from one or more runs of `missingPersonEP()` and `missingPersonIP()` respectively. Alternatively, `ep` can be a single output from `MPPsims()`, in which case `ip` should be NULL. See Examples.
`type`	Plot type; either 1, 2, 3 or 4.
`majorpoints`	A logical indicating whether "major" points should be drawn (see Details).
`minorpoints`	A logical indicating whether "minor" points should be drawn (see Details).
`ellipse`	A logical. If TRUE, data ellipsis are drawn for each group containing more than 1 element. NB: This fails with a warning if all points in a group fall on a line.
`col`	A colour vector, recycle to match the top level length of `ep`.
`labs`	A character of the same length as `ep`. If NULL, the names of `ep` are used, if present.

jitter	A logical (default: FALSE). If TRUE, a small jitter is added to the major points.
alpha	Transparency for minor points (see Details).
stroke	Border width for major points (see Details).
shape	Either "circle", "square", "diamond", "triangleUp" or "triangleDown", determining the shapes of both minor and major points.
size	Point size.
hline, vline	Single numbers indicating positions for horizontal/vertical "threshold" lines. If NULL (default), no lines are drawn.
xlim, ylim	Axis limits; automatically chosen if NULL.
xlab, ylab	Axis labels; automatically chosen if NULL.

Details

The plot types are as follows:

type = 1: x = Exclusion power; y = Inclusion power

type = 2: x = Exclusion odds ratio; y = Inclusion odds ratio

type = 3: x = Expected number of exclusions; y = average log(LR)

type = 4: x = Exclusion power; y = average LR

In the most general case ep (and similarly for ip) can be a list of lists of EPresult objects. We refer to the inner lists as "groups". A group may consist of a single output, or several (typically many simulations of the same situation). Points within the same group are always drawn with the same colour and shape.

When plotting several groups, two sets of points are drawn by default:

- Major points: Group means.
- Minor points: Individual points in groups with more than one element.

The parameters majorpoints and minorpoints control which of the above points are included.

Value

A ggplot2 plot object.

See Also

MPPsims(), missingPersonEP(), missingPersonEP()

Examples

```
### Example 1: Comparing the power of 3 reference families ###

# Frequencies for 2 STR markers
db = NorwegianFrequencies[1:2]  # Increase!

# Define pedigrees and simulate data
PAR = nuclearPed(1, child = "MP") |>
```

```
  profileSim(markers = db, ids = 1)
SIB = nuclearPed(2) |> relabel(old = 4, new = "MP") |>
  profileSim(markers = db, ids = 3)
GRA = linearPed(2) |> relabel(old = 5, new = "MP") |>
  profileSim(markers = db, ids = 1)

# Collect in list and plot
peds = list(PAR = PAR, SIB = SIB, GRA = GRA)
plotPedList(peds, marker = 1, hatched = typedMembers, frames = FALSE,
            col = list(red = "MP"))

# Compute exclusion/inclusion powers:
ep = lapply(peds, function(y)
  missingPersonEP(y, missing = "MP", verbose = FALSE))

ip = lapply(peds, function(y)    # increase nsim!
  missingPersonIP(y, missing = "MP", nsim = 5, threshold = 10, verbose = FALSE))

# Plot
powerPlot(ep, ip, size = 2)

# Different plot type, not dependent of `threshold`
powerPlot(ep, ip, size = 2, type = 3)



### Example 2: Exploring powers for different sets of available relatives

# Create trio pedigree
ref = nuclearPed(father = "fa", mother = "mo", child = "MP")

# Add empty marker with 5 alleles
ref = addMarker(ref, alleles = 1:5)

# Alternatives for genotyping
sel = list("fa", c("fa", "mo"))

# Simulate power for each selection
simData = MPPsims(ref, selections = sel, nProfiles = 3, lrSims = 5,
                  thresholdIP = 2, seed = 123, numCores = 1)

# Power plot 1: EP vs IP
powerPlot(simData, type = 1)
powerPlot(simData, type = 1, minorpoints = FALSE, hline = 0.8)




# Change shape, and modify legend order
powerPlot(simData[3:1], type = 1, shape = c("ci", "sq", "di"))

# Zoom in, and add threshold lines
powerPlot(simData, type = 1, xlim = c(0.2, 1), ylim = c(0.5, 1),
```

```
          hline = 0.8, vline = 0.8)

# Power plot 3: Expected number of exclusions vs E[log LR]
powerPlot(simData, type = 3)

# With horizontal/vertical lines
powerPlot(simData, type = 3, hline = log10(2), vline = 1)

# Plot 4: Illustrating the general inequality ELR > 1/(1-EP)
powerPlot(simData, type = 4)
```

`profileSim` *Simulation of complete DNA profiles*

Description

Simulation of DNA profiles for specified pedigree members. Some pedigree members may already be genotyped; in that case the simulation is conditional on these. The main work of this function is done by `markerSim()`.

Usage

```
profileSim(
  x,
  N = 1,
  ids = NULL,
  markers = NULL,
  seed = NULL,
  numCores = 1,
  simplify1 = TRUE,
  verbose = TRUE,
  ...
)
```

Arguments

`x`	A ped object or a list of such.
`N`	The number of complete simulations to be performed.
`ids`	A character (or coercible to character) of ID labels indicating whose genotypes should be simulated. Alternatively, a function taking `x` as input and returning a character vector of ID labels.
`markers`	Either a vector indicating a subset of markers attached to `x`, or a named list of frequency vectors. By default (`NULL`), all attached markers are used. If a frequency list is given, marker objects are created and attached to `x`. Simulations are conditional on the locus attributes (allele frequencies, mutation models, etc) and any existing genotypes in the indicated markers.

seed	An integer seed for the random number generator (optional).
numCores	The number of cores to be used. The default is 1, i.e., no parallelisation.
simplify1	A logical, by default TRUE, removing the outer list layer when N = 1. See Value.
verbose	A logical, by default TRUE.
...	Further arguments passed on to markerSim().

Value

A list of N objects similar to x, but with simulated genotypes. Any previously attached markers are replaced by the simulated profiles. If the indicated markers contained genotypes for some pedigree members, these are still present in the simulated profiles.

If N = 1 and simplify1 = TRUE, the outer list layer is removed, i.e., profileSim(..., N = 1, simplify1 = T) is equivalent to profileSim(..., N = 1, simplify1 = F)[[1]]. This is usually the desired object in interactive use, and works well with piping.

When using profileSim() in other functions, it is recommended to add simplify1 = FALSE to safeguard against issues with N = 1.

Examples

```
# Example pedigree with two brothers
x = nuclearPed(children = c("B1", "B2"))

### Simulate profiles using built-in freq database
profileSim(x, markers = NorwegianFrequencies[1:3])

### Conditioning on known genotypes for one brother

# Attach two SNP markers with genotypes for B1
y = x |>
  addMarker(B1 = "1/2", alleles = 1:2) |>
  addMarker(B1 = "1",   alleles = 1:2, chrom = "X")

# Simulate 2 profiles of B2 conditional on the above
profileSim(y, N = 2, ids = "B2", seed = 123)
```

---

profileSimParametric *Simulate complete DNA profiles given IBD coefficients*

Description

This function generalises markerSimParametric() in the same way that profileSim() generalises markerSim().

Usage

```
profileSimParametric(
  kappa = NULL,
  delta = NULL,
  states = NULL,
  N = 1,
  freqList = NULL,
  seed = NULL,
  returnValue = c("singletons", "alleles", "genotypes", "internal")
)
```

Arguments

kappa	A probability vector of length 3, giving a set of realised kappa coefficients (between two noninbred individuals).
delta	A probability vector of length 9, giving a set of condensed identity coefficients (Jacquard coefficients).
states	An integer vector of length N, with entries in 1-9. Each entry gives the identity state of the corresponding marker. (See details.)
N	A positive integer: the number of complete profiles to be simulated
freqList	A list of numeric vectors. Each vector is the allele frequencies of a marker.
seed	An integer seed for the random number generator (optional).
returnValue	Either "singleton" (default) or "alleles". (see Value).

Value

A list of length N, whose entries are determined by returnValue, as explained in markerSimParametric().

Examples

```
# A single profile with 9 markers, each with forced identity state
profileSimParametric(states = 1:9, freqList = NorwegianFrequencies[1:9])
```

---

quickLR *LR calculations for paternity and sibship*

Description

A thin wrapper around kinshipLR() for the common scenario of testing a pair of individuals for paternity and/or sibship, against being unrelated.

Usage

```
quickLR(x, ids = typedMembers(x), test = c("pat", "sib", "half"))
```

Arguments

x	A ped object or a list of such.
ids	A vector of two typed members of x. If not given, the typed members of x are selected by default, but note that this gives an error if the number of such individuals is not 2.
test	The hypotheses to be tested (against 'unrelatedness'). Allowed values are "pat" (=paternity), "sib" (=full siblings), and "half" (=half siblings). By default, all three are included.

Value

A (slightly simplified) LRresult object, as described in kinshipLR().

Examples

```
# Simulate 100 markers for half siblings
x = halfSibPed() |> markerSim(N = 100, ids = 4:5, alleles = 1:3, seed = 1)

# Test paternity, full sib, half sib
quickLR(x)
```

---

randomPersonEP *Random person exclusion power*

Description

This is a special case of exclusionPower(), computing the power to exclude a random person as a given pedigree member. More specifically, the function computes the probability of observing, in an individual unrelated to the family individual, a genotype incompatible with the typed family members.

Usage

```
randomPersonEP(x, id, markers = NULL, disableMutations = NA, verbose = TRUE)
```

Arguments

x	A ped object with attached markers.
id	The ID label of a single pedigree member.
markers	A vector indicating the names or indices of markers attached to the source pedigree. If NULL (default), then all markers attached to the source pedigree are used. If alleles or afreq is non-NULL, then this parameter is ignored.
disableMutations	This parameter determines how mutation models are treated. Possible values are as follows:

- NA (the default): Mutations are disabled only for those markers whose known genotypes are consistent with the pedigree. This is determined by temporarily removing all mutation models and checking which markers have nonzero likelihood.
- TRUE: Mutations are disabled for all markers. This will result in an error if any markers are inconsistent.
- FALSE: No action is done to disable mutations.
- A vector containing the names or indices of those markers for which mutations should be disabled.

verbose A logical.

Value

The EPresult object returned by exclusionPower().

Examples

```
# Four siblings:
x = nuclearPed(4)

# First 3 sibs typed with 4 triallelic markers
x = markerSim(x, N = 4, ids = 3:5, alleles = 1:3, seed = 577, verbose = FALSE)

# Probability that a random man is excluded as the fourth sibling
randomPersonEP(x, id = 6)
```

---

rankProfiles *Find the most likely profiles of a pedigree member*

Description

Identify and rank the most likely DNA profiles of a pedigree member. For each marker, the possible genotypes of the indicated person are ranked by likelihood.

Usage

```
rankProfiles(x, id, markers = NULL, maxPerMarker = Inf, verbose = FALSE)
```

Arguments

x	A ped object with attached markers.
id	The name of a single (typically untyped) pedigree member.
markers	Names or indices of the markers to be included. Default: all.
maxPerMarker	A single number, limiting the number of top genotypes considered for each marker. Default: Inf (no restriction).
verbose	A logical, by default FALSE.

Details

Note that this function assumes that all markers are independent.

If the marker data includes mutation models, it may be wise to try first with `maxPerMarker = 1` to limit computation time.

Value

A list with the following components (`N` denotes the number of markers):

- `profiles`: A data frame with `N+1` columns, containing the possible profiles, ranked by likelihood.
- `marginal1`: A numeric of length `N`, giving the marginal probability of the most likely genotype for each marker.
- `marginal2`: A numeric of length `N`, with marginals for the *second* most likely genotype for each marker, or `NA` if there is no second.
- `best`: A character of length `N` containing the most likely profile. This is the same as `names(marginal1)`, and also as `profiles[1, 1:N]`.

Examples

```
x = nuclearPed(nch = 4) |>
   markerSim(N = 4, alleles = c("a", "b", "c"), seed = 1, verbose = FALSE)
x
# Remove data for father
y = setAlleles(x, ids = 1, alleles = 0)

# Most likely profiles of father
rankProfiles(y, id = 1)

# Compare with truth
getGenotypes(x, ids = 1)

# Same example with mutations allowed
z = setMutmod(y, model = "equal", rate = 0.01)
rankProfiles(z, id = 1)
```

---

showInTriangle *Add points to the IBD triangle*

Description

This function is re-exported from the `ribd` package. For documentation see `ribd::showInTriangle()`.

simpleSim *Unconditional marker simulation*

Description

Unconditional simulation of unlinked markers

Usage

```
simpleSim(
  x,
  N,
  alleles,
  afreq,
  ids,
  Xchrom = FALSE,
  mutmod = NULL,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

x	A ped object.
N	A positive integer: the number of markers to be simulated.
alleles	A vector with allele labels.
afreq	A numeric vector of allele frequencies. If missing, the alleles are assumed to be equifrequent.
ids	A vector containing ID labels of those pedigree members whose genotypes should be simulated.
Xchrom	A logical: X linked markers or not?
mutmod	A list of mutation matrices named 'female' and 'male'.
seed	An integer seed for the random number generator (optional).
verbose	A logical.

Details

Simple genotype simulation, performed by first distributing alleles randomly to all founders, followed by Mendelian *gene dropping* down throughout the pedigree (i.e., for each non-founder a random allele is selected from each of the parents). Finally, genotypes of individuals not included in ids are removed.

Value

A ped object equal to x except its MARKERS entry, which consists of the N simulated markers.

See Also

markerSim()

Examples

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

y = cousinPed(1, child = TRUE)
simpleSim(y, N = 3, alleles = LETTERS[1:10])
```

Index