# Package 'sequoia'

May 3, 2021

**Type** Package

**Title** Pedigree Inference from SNPs

**Version** 2.3.3

**Date** 2021-04-30

**Author** Jisca Huisman [aut, cre]

**Maintainer** Jisca Huisman <jisca.huisman@gmail.com>

**Description** Multi-generational pedigree inference from incomplete data on hundreds of SNPs, including parentage assignment and sibship clustering. See Huisman (2017) (<DOI:10.1111/1755-0998.12665>) for more information.

**License** GPL-2

**URL** https://jiscah.github.io/

**LazyData** TRUE

**Imports** plyr (>= 1.8.0), stats, utils, graphics

**RoxygenNote** 7.1.1

**Suggests** xlsx, knitr, rmarkdown, bookdown, kinship2, lattice, R.rsp

**VignetteBuilder** knitr, R.rsp

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2021-05-03 00:20:02 UTC

## R topics documented:

---

`CalcBYprobs` *Birth year probabilities*

---

## Description

Estimate the probability that an individual with unknown birth year is born in year `y`, based on the (estimated) birth years of its parents and offspring and the age distribution of other parent-offspring pairs.

## Usage

```
CalcBYprobs(Pedigree = NULL, LifeHistData = NULL, AgePrior = NULL)
```

## Arguments

| | |
|---|---|
| `Pedigree` | dataframe with columns id-dam-sire. |
| `LifeHistData` | dataframe with 3 columns (optionally 5):<br>**ID** max. 30 characters long<br>**Sex** 1 = female, 2 = male, 3 = unknown, 4 = hermaphrodite, other numbers or NA = unknown<br>**BirthYear** birth or hatching year, integer, with missing values as NA or any negative value.<br>**BY.min** minimum birth year, only used if BirthYear is missing<br>**BY.max** maximum birth year, only used if BirthYear is missing<br>If the species has multiple generations per year, use an integer coding such that the candidate parents' 'Birth year' is at least one smaller than their putative offspring's. Column names are ignored, so ensure column order is ID - sex - birth year (- BY.min - BY.max). Individuals do not need to be in the same order as in 'Pedigree', nor do all individuals in the pedigree need to be included. |
| `AgePrior` | a matrix with probability ratios for individuals with age difference A to have relationship R, as generated by `MakeAgePrior`. If `NULL`, `MakeAgePrior` is called using its default values. |

## Details

This function assists in estimating birth years of individuals for which these are unknown, provided they have at least one parent or one offspring in the pedigree. It is not a substitute for field-based estimates of age, only a method to summarise the pedigree + birth year based information.

No distinction is made between genotyped and non-genotyped or dummy individuals.

## Value

A matrix with for each individual (rows) in the pedigree that has a missing birth year in `LifeHistData`, or that is not included in `LifeHistData`, the probability that it is born in `y` (columns). Probabilities are rounded to 3 decimal points and may therefore not sum exactly to 1.

## WARNING

Any errors in the pedigree or lifehistory data will cause errors in the birth year probabilities of their parents and offspring, and putatively also of more distant ancestors and descendants. If the ageprior is based on the same erroneous pedigree and lifehistory data, all birth year probabilities will be affected.

## See Also

`MakeAgePrior` to estimate effect of age on relationships.

## Examples

```
data(SeqOUT_griffin, package="sequoia")
BYprobs <- CalcBYprobs(Pedigree = SeqOUT_griffin$Pedigree,
                       LifeHistData = SeqOUT_griffin$LifeHist)

lattice::levelplot(t(BYprobs), aspect="fill", col.regions=hcl.colors)
```

---

# CalcMaxMismatch *Maximum Number of Mismatches*

---

## Description

Calculate the maximum expected number of mismatches for duplicate samples, parent-offspring pairs, and parent-parent-offspring trios.

## Usage

```
CalcMaxMismatch(Err, MAF, ErrFlavour = "version2.0", qntl = 1 - 1e-05)
```

## Arguments

| | |
|---|---|
| Err | estimated genotyping error rate, as a single number or 3x3 matrix (averaged value(s) across SNPs), or a vector with the same length as MAF, or a nSnp x 3 x 3 array. If a matrix, this should be the probability of observed genotype (columns) conditional on actual genotype (rows). Each row must therefore sum to 1. If an array, each 3x3 slice should abide this rule. |
| MAF | vector with minor allele frequency at each SNP. |
| ErrFlavour | function that takes Err as input, and returns a 3x3 matrix of observed (columns) conditional on actual (rows) genotypes, or choose from inbuilt ones as used in sequoia 'version2.0', 'version1.3', or 'version1.1'. Ignored if Err is a matrix. See ErrToM. |
| qntl | quantile of binomial distribution to be used as the maximum, of individual-level probability. For a desired dataset-level probability quantile $Q$, use $qntl = Q^(1/N)$, where $N$ is the number of individuals. |

## Details

The thresholds for maximum number of mismatches calculated here aim to minimise false negatives, i.e. to minimise the chance that any true duplicates or true parent-offspring pairs are already excluded during the filtering steps where these MaxMismatch values are used. Consequently, there is a high probability of false positives, i.e. it is likely that some sample pairs with fewer mismatches than the MaxMismatch threshold, are in fact not duplicate samples or parent-offspring pairs. Use of these MaxMismatch thresholds is therefore only the first step of pedigree reconstruction by sequoia.

## Value

A vector with three integers:

| | |
|---|---|
| DUP | Maximum number of differences between 2 samples from the same individual |
| OH | Maximum number of Opposing Homozygous SNPs between a true parent-offspring pair |
| ME | Maximum number of Mendelian Errors among a true parent-parent- offspring trio |

.

## See Also

SnpStats.

## Examples

```
CalcMaxMismatch(Err = 0.05, MAF = runif(n=100, min=0.3, max=0.5))
## Not run:
CalcMaxMismatch(Err = 0.02, MAF = SnpStats(MyGenoMatrix, Plot=FALSE)[,"AF"])

## End(Not run)
```

---

# CalcOHLLR *Calculate OH and LLR*

---

## Description

Count opposite homozygous (OH) loci between parent-offspring pairs and Mendelian errors (ME) between parent-parent-offspring trios, and calculate the parental log-likelihood ratios (LLR). Also returns estimate of actual genotypes.

## Usage

```
CalcOHLLR(
  Pedigree = NULL,
  GenoM = NULL,
  CalcLLR = TRUE,
  LifeHistData = NULL,
  AgePrior = FALSE,
  SeqList = NULL,
  Err = 1e-04,
  ErrFlavour = "version2.0",
  Tassign = 0.5,
  Tfilter = -2,
  Complex = "full",
  Herm = "no",
  quiet = FALSE
)
```

## Arguments

| | |
|---|---|
| Pedigree | dataframe with columns id-dam-sire. May include non-genotyped individuals, which will be treated as dummy individuals. If provided, any pedigree in SeqList is ignored. |
| GenoM | numeric matrix with genotype data: One row per individual, and one column per SNP, coded as 0, 1, 2 or -9 (missing). See also GenoConvert. |
| CalcLLR | calculate log-likelihood ratios for all assigned parents (genotyped + dummy/non-genotyped; parent vs. otherwise related). If FALSE, only number of mismatching SNPs are counted (OH & ME), and parameters LifeHistData, AgePrior, Err, Tassign, and Complex are **ignored**. Note also that calculating likelihood ratios is much more time consuming than counting OH & ME. |
| LifeHistData | dataframe with 3 columns (optionally 5):<br>**ID** max. 30 characters long<br>**Sex** 1 = female, 2 = male, 3 = unknown, 4 = hermaphrodite, other numbers or NA = unknown<br>**BirthYear** birth or hatching year, integer, with missing values as NA or any negative value.<br>**BY.min** minimum birth year, only used if BirthYear is missing<br>**BY.max** maximum birth year, only used if BirthYear is missing<br>If the species has multiple generations per year, use an integer coding such that the candidate parents' 'Birth year' is at least one smaller than their putative offspring's. Column names are ignored, so ensure column order is ID - sex - birth year (- BY.min - BY.max). Individuals do not need to be in the same order as in 'GenoM', nor do all genotyped individuals need to be included. |
| AgePrior | logical (TRUE/FALSE) whether to estimate the ageprior from Pedigree and LifeHistData, or a matrix as generated by MakeAgePrior and included in the sequoia output. The AgePrior affects which relationships are considered possible: only those where $P(A\|R)/P(A) > 0$. When TRUE, MakeAgePrior is called using its default values. When FALSE, all relationships are considered possible for all age differences, except that parent-offspring pairs cannot have age difference zero, and grand-parental pairs have an age difference of at least two. |
| SeqList | list with output from sequoia. If input parameter Pedigree=NULL, SeqList$Pedigree will be used if present, and SeqList$PedigreePar otherwise. If SeqList$Specs is present, input parameters with the same name as its items are ignored, except 'CalcLLR' and 'AgePriors=FALSE'. The list elements 'LifeHist', 'AgePriors', and 'ErrM' are also used if present, and override the corresponding input parameters. |
| Err | estimated genotyping error rate, as a single number or 3x3 matrix. Details below. The error rate is presumed constant across SNPs, and missingness is presumed random with respect to actual genotype. |
| ErrFlavour | function that takes Err (single number) as input, and returns a 3x3 matrix of observed (columns) conditional on actual (rows) genotypes, or choose from in-built options 'version2.0', 'version1.3', or 'version1.1', referring to the sequoia version in which they were the default. Ignored if Err is a matrix. See ErrToM. |

| | |
|---|---|
| Tassign | minimum LLR required for acceptance of proposed relationship, relative to next most likely relationship. Higher values result in more conservative assignments. Must be zero or positive. |
| Tfilter | threshold log10-likelihood ratio (LLR) between a proposed relationship versus unrelated, to select candidate relatives. Typically a negative value, related to the fact that unconditional likelihoods are calculated during the filtering steps. More negative values may decrease non-assignment, but will increase computational time. |
| Complex | Breeding system complexity. Either "full" (default), "simp" (simplified, no explicit consideration of inbred relationships), "mono" (monogamous). |
| Herm | Hermaphrodites, either "no", "A" (distinguish between dam and sire role, default if at least 1 individual with sex=4), or "B" (no distinction between dam and sire role). Both of the latter deal with selfing. |
| quiet | logical, suppress messages |

## Details

Any individual in Pedigree that does not occur in GenoM is substituted by a dummy individual; these can be recognised by the value 0' in columns 'SNPd.id.dam' and 'SNPd.id.sire' in the output. For non-genotyped individuals the parental log-likelihood ratio can be calculated if they have at least one genotyped offspring (see also getAssignCat).

The birth years in LifeHistData and the AgePrior are not used in the calculation and do not affect the value of the likelihoods for the various relationships, but they _are_ used during some filtering steps, and may therefore affect the likelihood _ratio_. The default (AgePrior=FALSE) assumes all age-relationship combinations are possible, which may mean that some additional alternatives are considered compared to the sequoia default, resulting in somewhat lower LLR values.

A negative LLR for A's parent B indicates either that B is not truely the parent of A, or that B's parents are incorrect. The latter may cause B's presumed true, unobserved genotype to divert from its observed genotype, with downstream consequences for its offspring. In rare cases it may also be due to 'weird', non-implemented double or triple relationships between A and B.

## Value

The Pedigree dataframe with additional columns:

| | |
|---|---|
| LLRdam | Log10-Likelihood Ratio (LLR) of this female being the mother, versus the next most likely relationship between the focal individual and this female (see Details for relationships considered) |
| LLRsire | idem, for male parent |
| LLRpair | LLR for the parental pair, versus the next most likely configuration between the three individuals (with one or neither parent assigned) |
| OHdam | Number of loci at which the offspring and mother are opposite homozygotes |
| OHsire | idem, for father |
| MEpair | Number of Mendelian errors between the offspring and the parent pair, includes OH as well as e.g. parents being opposing homozygotes, but the offspring not being a heterozygote. The offspring being OH with both parents is counted as 2 errors. |

| | |
|---|---|
| SNPd.id | Number of SNPs scored (non-missing) for the focal individual |
| SNPd.id.dam | Number of SNPs scored (non-missing) for both individual and dam |
| SNPd.id.sire | Number of SNPs scored for both individual and sire |
| Sexx | Sex in LifeHistData, or inferred Sex when assigned as part of parent-pair |
| BY.est | mode of birth year probability distribution |
| BY.lo | lower limit of 95% highest density region of birth year probability distribution |
| BY.hi | higher limit |

The columns 'LLRdam', 'LLRsire' and 'LLRpair' are only included when CalcLLR=TRUE. When a parent or parent-pair is incompatible with the lifehistory data or presumed genotyping error rate, the error value '777' may be given.

The columns 'Sexx', 'BY.est', 'BY.lo' and 'BY.hi' are only included when LifeHistData is provided, and at least one genotyped individual has an unknown birth year or unknown sex.

## See Also

SummarySeq for visualisation of OH & LLR distributions; CalcPairLL for the likelihoods underlying the LLR, GenoConvert to read in various genotype data formats, CheckGeno; PedPolish to check and 'polish' the pedigree; getAssignCat to find which id-parent pairs are both genotyped or can be substituted by dummy individuals; sequoia for pedigree reconstruction.

## Examples

```
data(Ped_HSg5, SimGeno_example, LH_HSg5, package="sequoia")
# count Mendelian errors in an existing pedigree
Ped.OH <- CalcOHLLR(Pedigree = Ped_HSg5, GenoM = SimGeno_example,
                    CalcLLR = FALSE)
Ped.OH[50:55,]
# view histograms
SummarySeq(Ped.OH, Panels="OH")

# Parent likelihood ratios in an existing pedigree, including for
# non-genotyped parents
Ped.LLR <- CalcOHLLR(Pedigree = Ped_HSg5, GenoM = SimGeno_example,
                    CalcLLR = TRUE, LifeHistData=LH_HSg5, AgePrior=TRUE)
SummarySeq(Ped.LLR, Panels="LLR")


# likelihood ratios change with presumed genotyping error rate:
Ped.LLR.B <- CalcOHLLR(Pedigree = Ped_HSg5, GenoM = SimGeno_example,
                    CalcLLR = TRUE, LifeHistData=LH_HSg5, AgePrior=TRUE,
                    Err = 0.005)
SummarySeq(Ped.LLR.B, Panels="LLR")

# run sequoia with CalcLLR=FALSE, and add OH + LLR later:
data(Ped_griffin, LH_griffin, package="sequoia")
Genotypes <- SimGeno(Ped_griffin, nSnp=400)
SeqOUT <- sequoia(Genotypes, LH_griffin, CalcLLR=FALSE,quiet=TRUE,Plot=FALSE)
PedA <- CalcOHLLR(Pedigree = SeqOUT[["Pedigree"]][, 1:3], GenoM = Genotypes,
```

```
  LifeHistData = LH_griffin, AgePrior = TRUE, Complex = "full")
SummarySeq(PedA, Panels=c("LLR", "OH"))
```

---

CalcPairLL *Calculate Likelihoods for Alternative Relationships*

---

## Description

For each specified pair of individuals, calculate the log10-likelihoods of being PO, FS, HS, GP, FA, HA, U (see Details). Individuals must be genotyped or have at least one genotyped offspring.

**NOTE** values $> 0$ are various NA types, see 'Likelihood special codes' in 'Value' section below.

## Usage

```
CalcPairLL(
  Pairs = NULL,
  GenoM = NULL,
  Pedigree = NULL,
  LifeHistData = NULL,
  AgePrior = TRUE,
  SeqList = NULL,
  Complex = "full",
  Herm = "no",
  Err = 1e-04,
  ErrFlavour = "version2.0",
  Tassign = 0.5,
  Tfilter = -2,
  quiet = FALSE,
  Plot = TRUE
)
```

## Arguments

Pairs — dataframe with columns ID1 and ID2, and optionally

**Sex1** Sex of ID1, 1=female, 2=male, 3=unknown, or NA to take from LifeHistData. The sex of individuals occurring as parent in Pedigree cannot be altered.

**Sex2** Sex of ID2

**AgeDif** Age difference in whole time units, BirthYear1 - BirthYear2 (i.e. positive if ID2 is born before ID1). If NA, calculated from LifeHistData. Use '999' to explicitly specify 'unknown'.

**focal** relationship character abbreviation; PO, FS, HS, GP or U. See Details for its effect and explanation of abbreviations. Default: U

**patmat** 1=maternal relatives, 2=paternal relatives. Only relevant for HS & GP, for which it defaults to Sex1, or 1 if Sex1=3, but is currently only predictably implemented for pairs of two genotyped individuals. Always equal to Sex2 for PO pairs when Sex2 is known.

**dropPar1** Drop the parents of `ID1` before calculating the pair likelihood, rather than conditioning on them; choose from 'none', 'dam', 'sire', or 'both'. See example. If e.g. the pair shares a common mother, 'none' and 'sire' will condition on this shared mother and not calculate the likelihood that they are maternal siblings, while dropPar1='dam' or 'both' will calculate that likelihood, and the other likelihoods as if the mother of ID1 were unknown.

**dropPar2** as `dropPar1`, for `ID2`

`GenoM` numeric matrix with genotype data: One row per individual, and one column per SNP, coded as 0, 1, 2 or -9 (missing). See also `GenoConvert`.

`Pedigree` dataframe with columns id-dam-sire; likelihoods will be calculated conditional on the pedigree. May include non-genotyped individuals, which will be treated as dummy individuals.

`LifeHistData` dataframe with 3 columns (optionally 5):

**ID** max. 30 characters long

**Sex** 1 = female, 2 = male, 3 = unknown, 4 = hermaphrodite, other numbers or NA = unknown

**BirthYear** birth or hatching year, integer, with missing values as NA or any negative value.

**BY.min** minimum birth year, only used if BirthYear is missing

**BY.max** maximum birth year, only used if BirthYear is missing

If the species has multiple generations per year, use an integer coding such that the candidate parents' 'Birth year' is at least one smaller than their putative offspring's. Column names are ignored, so ensure column order is ID - sex - birth year (- BY.min - BY.max). Individuals do not need to be in the same order as in 'GenoM', nor do all genotyped individuals need to be included.

`AgePrior` logical (`TRUE/FALSE`) whether to estimate the ageprior from `Pedigree` and `LifeHistData`, or a matrix as generated by `MakeAgePrior` and included in the `sequoia` output. The `AgePrior` affects which relationships are considered possible: only those where $P(A|R)/P(A) > 0$. When `TRUE`, `MakeAgePrior` is called using its default values. When `FALSE`, all relationships are considered possible for all age differences, except that parent-offspring pairs cannot have age difference zero, and grand-parental pairs have an age difference of at least two.

`SeqList` list with output from `sequoia`. If input parameter `Pedigree=NULL`, `SeqList$Pedigree` will be used if present, and `SeqList$PedigreePar` otherwise. If `SeqList$Specs` is present, input parameters with the same name as its items are ignored. The list elements 'LifeHist', 'AgePriors', and 'ErrM' are also used if present, and override the corresponding input parameters.

`Complex` Breeding system complexity. Either "full" (default), "simp" (simplified, no explicit consideration of inbred relationships), "mono" (monogamous).

`Herm` Hermaphrodites, either "no", "A" (distinguish between dam and sire role, default if at least 1 individual with sex=4), or "B" (no distinction between dam and sire role). Both of the latter deal with selfing.

| | |
|---|---|
| Err | estimated genotyping error rate, as a single number or 3x3 matrix. Details below. The error rate is presumed constant across SNPs, and missingness is presumed random with respect to actual genotype. |
| ErrFlavour | function that takes Err (single number) as input, and returns a 3x3 matrix of observed (columns) conditional on actual (rows) genotypes, or choose from in-built options 'version2.0', 'version1.3', or 'version1.1', referring to the sequoia version in which they were the default. Ignored if Err is a matrix. See ErrToM. |
| Tassign | minimum LLR required for acceptance of proposed relationship, relative to next most likely relationship. Higher values result in more conservative assignments. Must be zero or positive. |
| Tfilter | threshold log10-likelihood ratio (LLR) between a proposed relationship versus unrelated, to select candidate relatives. Typically a negative value, related to the fact that unconditional likelihoods are calculated during the filtering steps. More negative values may decrease non-assignment, but will increase computational time. |
| quiet | logical, suppress messages |
| Plot | logical, display scatter plots by PlotPairLL. |

## Details

The same pair may be included multiple times, e.g. with different sex, age difference, or focal relationship, to explore their effect on the likelihoods. Likelihoods are only calculated for relationships that are possible given the age difference, e.g. PO (parent-offspring) is not calculated for pairs with an age difference of 0.

Non-genotyped individuals can be included if they have at least one genotyped offspring and can be turned into a dummy (see getAssignCat); to establish this a pedigree must be provided.

**Warning 1**: There is no check whether the input pedigree is genetically sensible, it is simply conditioned upon. Checking whether a pedigree is compatible with the SNP data can be done with CalcOHLLR.

**Warning 2**: Conditioning on a Pedigree can make computation orders of magnitude slower.

## Value

The Pairs dataframe including all optional columns listed above, plus the additional columns:

| | |
|---|---|
| LL_xx | Log10-Likelihood of this pair having relationship xx, with xx being one of PO, FS, etc. as detailed below. |
| TopRel | Abbreviation of most likely relationship |
| LLR | Likelihood ratio between most-likely and second most likely relationships |

**Relationship abbreviations:**

| | |
|---|---|
| PO | Parent - offspring |
| FS | Full siblings |
| HS | Half siblings |
| GP | Grandparent |

| | |
|---|---|
| FA | Full avuncular |
| HA | Half avuncular and other 3rd degree relationships |
| U | Unrelated |
| 2nd | Unclear which type of 2nd degree relatives (HS, GP, or FA) |
| ?? | Unclear which type of 1st, 2nd or 3rd degree relatives |

**Likelihood special codes:**

| | |
|---|---|
| 222 | Maybe (via) other parent (e.g. focal="GP", but as likely to be maternal as paternal grandparent, and therefore not assignable) |
| 333 | Excluded from comparison (shouldn't occur) |
| 444 | Not implemented (e.g. would create an odd double/triple relationship in combination with the provided pedigree) |
| 777 | Impossible (e.g. cannot be both full sibling and grandparent) |
| 888 | Already assigned in the provided pedigree (see dropPar arguments) |
| 999 | NA |

## Double relationships & focal relationship

Especially when Complex='full', not only the seven relationship alternatives listed above are considered, but a whole range of possible double and even triple relationships. For example, mother A and offspring B (PO) may also be paternal half-siblings (HS, A and A's mother mated with same male), grandmother and grand-offspring (GP, B's father is A's son), or paternal aunt (B's father is a full or half sib of A).

The likelihood reported as 'LL_PO' is the most-likely one of the possible alternatives, among those that are not impossible due to age differences or due to the pedigree (as reconstructed up to that point). Whether e.g. the likelihood to be both PO & HS is counted as PO or as HS, depends on the situation and is determined by the variable 'focal': During parentage assignment, it is counted as PO but not HS, while during sibship clustering, it is counted as HS but not PO – not omitting from the alternative relationship would result in a deadlock.

## See Also

PlotPairLL to plot alternative relationship pairs from the output; CalcOHLLR to calculate LLR for parents & parent-pairs in a pedigree; GetRelM to find all pairwise relatives according to the pedigree; GetMaybeRel to get likely relative pairs not in the pedigree.

## Examples

```
# likelihoods underlying parent LLR in pedigree:
data(LH_HSg5, SimGeno_example)
Seq.HSg5 <- sequoia(SimGeno_example, LH_HSg5, Module="par")
tail(Seq.HSg5$PedigreePar)
# take bottom 3 individuals:
Pairs <- data.frame(ID1 = c("a01190", "b01191", "a01192"),
                    ID2 = rep(c("a00012", "b00007"), each=3),
                    AgeDif = 1, focal = "PO")
```

```
# LLRdam & LLRsire:
CalcPairLL(Pairs, SimGeno_example)
# LLRpair is min. of dam & sire LLR, conditional on co-parent:
CalcPairLL(cbind(Pairs, dropPar1=rep(c("dam", "sire"), each=3)),
           SimGeno_example, SeqList = Seq.HSg5)


# likelihoods underlying LLR in getMaybeRel output:
data(Ped_griffin, SeqOUT_griffin, package="sequoia")
Geno.griffin <- SimGeno(Ped_griffin, nSnp=200, SnpError = 0.01, ParMis=0.4)
MR <- GetMaybeRel(GenoM = Geno.griffin,
                  LifeHistData = SeqOUT_griffin$LifeHist,
                  Module = "par", Err = 0.001)
FivePairs <- MR$MaybePar[1:5, c("ID1", "ID2", "Sex1", "Sex2")]
FivePairs$AgeDif <- NA   # pretend unknown age differences
PairLL <- CalcPairLL(Pairs = rbind( cbind(FivePairs, focal = "PO"),
                                    cbind(FivePairs, focal = "HS"),
                                    cbind(FivePairs, focal = "GP")),
                     GenoM = Geno.griffin,
                     Err = 0.005, Plot=FALSE)
PairLL[c(1, 6, 11), ]
# LL(FS)==222 : HSHA, HSGP, FAHA more likely than FS
# LL(GP) higher when focal=HS: GP via 'other' parent also considered
# LL(FA) higher when focal=PO: FAHA, or FS of 'other' parent
```

---

`CalcRped` *Calculate Pedigree Relatedness*

---

## Description

Morph pedigree into a **kinship2** compatible format and use `kinship` to calculate kinship coefficients; relatedness = 2*kinship.

## Usage

```
CalcRped(Pedigree, OUT = "DF")
```

## Arguments

| | |
|---|---|
| `Pedigree` | dataframe with columns id-dam-sire. |
| `OUT` | desired output format, 'M' for matrix or 'DF' for dataframe with columns IID1 - IID2 - R.ped. |

## Value

A matrix or dataframe.

CheckGeno *Check Genotype Matrix*

## Description

Check that the provided genotype matrix is in the correct format, and check for low call rate samples and SNPs.

## Usage

```
CheckGeno(
  GenoM,
  quiet = FALSE,
  Plot = FALSE,
  Return = "GenoM",
  DumPrefix = c("F0", "M0")
)
```

## Arguments

| | |
|---|---|
| GenoM | the genotype matrix. |
| quiet | suppress messages. |
| Plot | display the plots of SnpStats. |
| Return | either 'GenoM' to return the cleaned-up genotype matrix, or 'excl' to return a list with excluded SNPs and individuals (see Value). |
| DumPrefix | length 2 vector, to check if these don't occur among genotyped individuals. |

## Value

If Return='excl' a list with, if any are found:

| | |
|---|---|
| ExcludedSNPs | SNPs scored for <10 excluded when running sequoia |
| ExcludedSnps-mono | monomorphic (fixed) SNPs; automatically excluded when running sequoia. This includes nearly-fixed SNPs with MAF $= 1/2N$. Column numbers are *after* removal of ExcludedSNPs, if any. |
| ExcludedIndiv | Individuals scored for <5 reliably included during pedigree reconstruction. Individual call rate is calculated after removal of 'Excluded SNPs' |
| Snps-LowCallRate | SNPs scored for 10 recommended to be filtered out |
| Indiv-LowCallRate | individuals scored for <50 recommended to be filtered out |

When Return='excl' the return is invisible, i.e. a check is run and warnings or errors are always displayed, but nothing may be returned.

## Thresholds

Appropriate call rate thresholds for SNPs and individuals depend on the total number of SNPs, distribution of call rates, genotyping errors, and the proportion of candidate parents that are SNPd (sibship clustering is more prone to false positives). Note that filtering first on SNP call rate tends to keep more individuals in.

## See Also

SnpStats to calculate SNP call rates; CalcOHLLR to count the number of SNPs scored in both focal individual and parent.

## Examples

```
data(Ped_HSg5)
GenoM <- SimGeno(Ped_HSg5, nSnp=400, CallRate = runif(400, 0.2, 0.8))
# quick alternative:
GenoM.checked <- CheckGeno(GenoM)

# user supervised alternative:
Excl <- CheckGeno(GenoM, Return = "excl")
GenoM.orig <- GenoM   # make a 'backup' copy
if ("ExcludedSnps" %in% names(Excl))
  GenoM <- GenoM[, -Excl[["ExcludedSnps"]]]
if ("ExcludedInd" %in% names(Excl))
  GenoM <- GenoM[!rownames(GenoM) %in% Excl[["ExcludedInd"]], ]
if ("ExcludedIndiv" %in% names(Excl))
  GenoM <- GenoM[!rownames(GenoM) %in% Excl[["ExcludedIndiv"]], ]

# warning about  SNPs scored for <50% of individuals ?
SnpCallRate <- apply(GenoM, MARGIN=2,
                     FUN = function(x) sum(x!=-9)) / nrow(GenoM)
hist(SnpCallRate, breaks=50, col="grey")
GenoM <- GenoM[, SnpCallRate > 0.6]

# to be on the safe side, filter out low call rate individuals
IndivCallRate <- apply(GenoM, MARGIN=1,
                       FUN = function(x) sum(x!=-9)) / ncol(GenoM)
hist(IndivCallRate, breaks=50, col="grey")
GoodSamples <- rownames(GenoM)[ IndivCallRate > 0.8]
```

---

# ComparePairs *Compare Pairwise Relationships*

---

## Description

Compare, count and identify different types of relative pairs between two pedigrees, or within one pedigree.

## Usage

```
ComparePairs(
  Ped1 = NULL,
  Ped2 = NULL,
  Pairs2 = NULL,
  GenBack = 1,
  patmat = FALSE,
  ExcludeDummies = TRUE,
  DumPrefix = c("F0", "M0"),
  Return = "Counts"
)
```

## Arguments

| | |
|---|---|
| Ped1 | first (e.g. original/reference) pedigree, dataframe with 3 columns: id-dam-sire. |
| Ped2 | optional second (e.g. inferred) pedigree. |
| Pairs2 | optional dataframe with relationships categories between pairs of individuals, instead of or in addition to Ped2, e.g. as returned by GetMaybeRel. First three columns: ID1-ID2-relationship, column names and any additional columns are ignored. |
| GenBack | number of generations back to consider; 1 returns parent-offspring and sibling relationships, 2 also returns grandparental, avuncular and first cousins. GenBack >2 is not implemented. |
| patmat | logical, distinguish between paternal versus maternal relative pairs? |
| ExcludeDummies | logical, exclude dummy IDs from output? Individuals with e.g. the same dummy father will still be counted as paternal halfsibs. No attempt is made to match dummies in one pedigree to individuals in the other pedigree; for that use PedCompare. |
| DumPrefix | character vector with the prefixes identifying dummy individuals. Use 'F0' ('M0') to avoid matching to regular individuals with IDs starting with 'F' ('M'), provided Ped2 has fewer than 999 dummy females (males). |
| Return | return a matrix with Counts or a Summary of the number of identical relationships and mismatches per relationship, or detailed results as a 2xNxN Array or as a Dataframe. All returns a list with all four. |

## Details

If Pairs2 is as returned by GetMaybeRel (identified by the additional column names 'LLR' and 'OH'), these relationship categories are appended with an '?' in the output, to distinguish them from those derived from Ped2.

When Pairs2$TopRel contains values other than the ones listed among the return values for the combination of patmat and GenBack, they are prioritised in decreasing order of factor levels, or in decreasing alphabetical order, and before the default (ped2 derived) levels.

The matrix returned by DyadCompare [Deprecated] is a subset of the matrix returned here using default settings.

## Value

Depending on `Return`, one of the following, or a list with all:

`Counts` (the default), a matrix with counts, with the classification in `Ped1` on rows and that in `Ped2` in columns. Counts for 'symmetrical' pairs ("FS", "HS", "MHS", "PHS", "FC1", "DFC1", "U","X") are divided by two.

`Summary` a matrix with one row per relationship type and four columns , named as if `Ped1` is the true pedigree:

- **n** total number of pairs with that relationship in `Ped1`, and occurring in `Ped2`
- **OK** Number of pairs with same relationship in `Ped2` as in `Ped1`
- **hi** Number of pairs with 'higher' relationship in `Ped2` as in `Ped1` (e.g. FS instead of HS; ranking is the order given below)
- **lo** Number of pairs with 'lower' relationship in `Ped2` as in `Ped1`, but not unrelated in `Ped2`

`Array` a 2xNxN array (if `Ped2` or `Pairs2` is specified) or a NxN matrix , where N is the total number of individuals occurring in `Ped1` and/or `Ped2`.

`Dataframe` a dataframe with $N^2$ rows and four columns:

- **id.A** First individual of the pair
- **id.B** Second individual of the pair
- **RC1** the relationship category in `Ped1`, as a factor with all considered categories as levels, including those with 0 count
- **RC2** the relationship category in `Ped2`

Each pair is listed twice, e.g. once as P and once as O, or twice as FS.

## Relationship abbreviations and ranking

By default (`GenBack=1,patmat=FALSE`) the following 7 relationships are distinguished:

- **S**: Self (not included in `Counts`)
- **MP**: Parent
- **O**: Offspring (not included in `Counts`)
- **FS**: Full sibling
- **HS**: Half sibling
- **U**: Unrelated, or otherwise related
- **X**: Either or both individuals not occurring in both pedigrees

In the array and dataframe, 'MP' indicates that the second (column) individual is the parent of the first (row) individual, and 'O' indicates the reverse.

When `GenBack=1,patmat=TRUE` the categories are (S)-M-P-(O)-FS-MHS-PHS- U-X.

When `GenBack=2,patmat=TRUE`, the following relationships are distinguished:

- **S**: Self (not included in `Counts`)
- **M**: Mother
- **P**: Father

- **O**: Offspring (not included in `Counts`)
- **FS**: Full sibling
- **MHS**: Maternal half-sibling
- **PHS**: Paternal half-sibling
- **MGM**: Maternal grandmother
- **MGF**: Maternal grandfather
- **PGM**: Paternal grandmother
- **PGF**: Paternal grandfather
- **GO**: Grand-offspring (not included in `Counts`)
- **FA**: Full avuncular; maternal or paternal aunt or uncle
- **HA**: Half avuncular
- **FN**: Full nephew/niece (not included in `Counts`)
- **HN**: Half nephew/niece (not included in `Counts`)
- **FC1**: Full first cousin
- **DFC1**: Double full first cousin
- **U**: Unrelated, or otherwise related
- **X**: Either or both individuals not occurring in both pedigrees

Note that for avuncular and cousin relationships no distinction is made between paternal versus maternal, as this may differ between the two individuals and would generate a large number of sub-classes. When a pair is related via multiple paths, the first-listed relationship is returned.

When `GenBack=2,patmat=FALSE`, MGM, MGF, PGM and PGF are combined into GP, with the rest of the categories analogous to the above.

## See Also

`PedCompare` for individual-based comparison; `GetRelM` for a pairwise relationships matrix of a single pedigree; `PlotRelPairs` for visualisation of relationships within each pedigree.

To estimate P(actual relationship (Ped1) | inferred relationship (Ped2)), see examples at `EstConf`.

## Examples

```
data(Ped_griffin, SeqOUT_griffin, package="sequoia")
PairsG <- ComparePairs(Ped_griffin, SeqOUT_griffin[["Pedigree"]],
                       patmat = TRUE, ExcludeDummies = TRUE, Return = "All")
PairsG$Counts

# pairwise correct assignment rate:
PairsG$Summary[,"OK"] / PairsG$Summary[,"n"]

# check specific pair:
PairsG$Array[, "i190_2010_M", "i168_2009_F"]
# or
RelDF <- PairsG$Dataframe   # for brevity
```

```
RelDF[RelDF$id.A=="i190_2010_M" & RelDF$id.B=="i168_2009_F", ]

# Colony-style lists of full sib dyads & half sib dyads:
FullSibDyads <- with(RelDF, RelDF[Ped1 == "FS" & id.A < id.B, ])
HalfSibDyads <- with(RelDF, RelDF[Ped1 == "HS" & id.A < id.B, ])
# Use 'id.A < id.B' because each pair is listed 2x
```

---

DyadCompare *Compare Dyads*

---

## Description

Count the number of half and full sibling pairs correctly and incorrectly assigned. DEPRECATED - PLEASE USE ComparePairs

## Usage

```
DyadCompare(Ped1 = NULL, Ped2 = NULL, na1 = c(NA, "0"))
```

## Arguments

| | |
|---|---|
| Ped1 | original pedigree, dataframe with 3 columns: id-dam-sire. |
| Ped2 | second (inferred) pedigree. |
| na1 | the value for missing parents in Ped1. |

## Value

A 3x3 table with the number of pairs assigned as full siblings (FS), half siblings (HS) or unrelated (U, including otherwise related) in the two pedigrees, with the classification in Ped1 on rows and that in Ped2 in columns.

## See Also

ComparePairs which supersedes this function; PedCompare

## Examples

```
data(Ped_HSg5, SimGeno_example, LH_HSg5, package="sequoia")
SeqOUT <- sequoia(GenoM = SimGeno_example, LifeHistData = LH_HSg5,
                  Module="par", quiet=TRUE, Plot=FALSE)
DyadCompare(Ped1=Ped_HSg5, Ped2=SeqOUT$Pedigree)
```

ErrToM *Generate Genotyping Error Matrix*

## Description

Generate a matrix with the probabilities of observed genotypes (columns) conditional on actual genotypes (rows), or return a function to generate such matrices (using a single value Err as input to that function).

## Usage

```
ErrToM(Err = NA, flavour = "version2.0", Return = "matrix")
```

## Arguments

| | |
|---|---|
| Err | estimated genotyping error rate, as a single number or 3x3 or 4x4 matrix. If a single number, an error model is used that aims to deal with scoring errors typical for SNP arrays. If a matrix, this should be the probability of observed genotype (columns) conditional on actual genotype (rows). Each row must therefore sum to 1. If `Return='function'`, this may be `NA`. |
| flavour | matrix-generating function, or one of 'version2.0', 'version1.3' ('SNPchip'), 'version1.1' ('version111'), referring to the sequoia version in which it was used as default. Ignored if `Err` is a matrix and `Return='matrix'` (in which case the matrix will only be checked for validity). |
| Return | output, 'matrix' (always 3x3) or 'function'. |

## Details

By default (`flavour = "SNPchip"`), `Err` is interpreted as a locus-level error rate (rather than allele-level), and equals the probability that an actual heterozygote is observed as either homozygote (i.e., the probability that it is observed as AA = probability that observed as aa = `Err`/2). The probability that one homozygote is observed as the other is $(\texttt{Err}/2)^2$.

The inbuilt 'flavours' correspond to the presumed and simulated error structures, which have changed with sequoia versions. The most appropriate error structure will depend on the genotyping platform; 'version0.9' and 'version1.1' were inspired by SNP array genotyping while 'version1.3' and 'version2.0' are intended to be more general.

Pr(observed genotype (columns) | actual genotype (rows)):

*version2.0:*

| | **0** | **1** | **2** |
|---|---|---|---|
| **0** | $(1-E/2)^2$ | $E(1-E/2)$ | $(E/2)^2$ |
| **1** | $E/2$ | $1-E$ | $E/2$ |
| **2** | $(E/2)^2$ | $E(1-E/2)$ | $(1-E/2)^2$ |

*version1.3*

| | **0** | **1** | **2** |
|---|---|---|---|
| **0** | $1 - E - (E/2)^2$ | $E$ | $(E/2)^2$ |
| **1** | $E/2$ | $1 - E$ | $E/2$ |
| **2** | $(E/2)^2$ | $E$ | $1 - E - (E/2)^2$ |

*version1.1*

| | **0** | **1** | **2** |
|---|---|---|---|
| **0** | $1 - E$ | $E/2$ | $E/2$ |
| **1** | $E/2$ | $1 - E$ | $E/2$ |
| **2** | $E/2$ | $E/2$ | $1 - E$ |

*version0.9* (not recommended)

| | **0** | **1** | **2** |
|---|---|---|---|
| **0** | $1 - E$ | $E$ | $0$ |
| **1** | $E/2$ | $1 - E$ | $E/2$ |
| **2** | $0$ | $E$ | $1 - E$ |

## Value

Either a 3x3 matrix, or a function generating a 3x3 matrix.

---

`EstConf` *Confidence Probabilities*

---

## Description

Estimate confidence probabilities ('backward') and assignment error rates ('forward') per category (genotyped/dummy) by repeatedly simulating genotype data from a reference pedigree using `SimGeno`, reconstruction a pedigree from this using `sequoia`, and counting the number of mismatches using `PedCompare`.

## Usage

```
EstConf(
  Pedigree = NULL,
  LifeHistData = NULL,
  args.sim = list(nSnp = 400, SnpError = 0.001, ParMis = c(0.4, 0.4)),
  args.seq = list(Module = "ped", Err = 0.001, Tassign = 0.5, CalcLLR = FALSE),
  nSim = 10,
```

```
  nCores = 1,
  quiet = TRUE
)
```

## Arguments

| | |
|---|---|
| Pedigree | reference pedigree from which to simulate, dataframe with columns id-dam-sire. Additional columns are ignored. |
| LifeHistData | dataframe with id, sex (1=female, 2=male, 3=unknown), and birth year. |
| args.sim | list of arguments to pass to SimGeno, such as nSnp (number of SNPs), SnpError (genotyping error rate) and ParMis (proportion of non-genotyped parents). Set to NULL to use all default values. |
| args.seq | list of arguments to pass to sequoia, such as Module ('par' or 'ped'), Err (assumed genotyping error rate), and Complex. May include (part of) SeqList, a list of sequoia output (i.e. as a list-within-a-list). Set to NULL to use all default values. |
| nSim | number of iterations of simulate - reconstruct - compare to perform, i.e. number of simulated datasets. |
| nCores | number of computer cores to use. If >1, package **parallel** is used. Set to NULL to use all but one of the available cores, as detected by parallel::detectCores() (using all cores tends to freeze up your computer). |
| quiet | suppress messages. TRUE runs SimGeno and sequoia quietly, 'very' also suppresses other messages and the iteration counter when nCores=1 (there is no iteration counter when nCores>1). |

## Details

The confidence probability is taken as the number of correct (matching) assignments, divided by all assignments made in the *observed* (inferred-from-simulated) pedigree. In contrast, the false negative & false positive assignment rates are proportions of the number of parents in the *true* (reference) pedigree. Each rate is calculated separatedly for dams & sires, and separately for each category (**G**enotyped/**D**ummy(fiable)/**X** (none)) of individual, parent and co-parent.

This function does not know which individuals in the actual Pedigree are genotyped, so the confidence probabilities need to be added to the Pedigree as shown in the example at the bottom.

A confidence of 1 means all assignments on simulated data were correct for that category-combination. It should be interpreted as (and perhaps modified to) $> 1 - 1/N$, where sample size N is given in the last column of the ConfProb and PedErrors dataframes in the output. The same applies for a false negative/positive rate of 0 (i.e. to be interpreted as $< 1/N$).

## Value

A list, with elements:

| | |
|---|---|
| ConfProb | See below |
| PedErrors | See below |
| Pedigree.reference | the pedigree from which data was simulated |

`LifeHistData`

`Pedigree.inferred`
: a list with for each iteration the inferred pedigree based on the simulated data

`SimSNPd`
: a list with for each iteration the IDs of the individuals simulated to have been genotyped

`PedComp.fwd`
: `Counts` from the 'forward' `PedCompare`, from which `PedErrors` is calculated

`RunParams`
: a list with the call to `EstConf`, as well as the default parameter values for `SimGeno`, and `sequoia`.

`RunTime`
: `sequoia` runtime per simulation in seconds, as measured by `system.time()['elapsed']`.

Dataframe `ConfProb` has 7 columns:

`id.cat, dam.cat, sire.cat`
: Category of the focal individual, dam, and sire, in the pedigree inferred based on the simulated data. Coded as G=genotyped, D=dummy, X=none

`dam.conf`
: Probability that the dam is correct, given the categories of the assigned dam and sire (ignoring whether or not the sire is correct)

`sire.conf`
: as `dam.conf`, for the sire

`pair.conf`
: Probability that both dam and sire are correct, given their categories

`N`
: Number of individuals per category-combination, across all `nSim` iterations

Array `PedErrors` has three dimensions:

`class`
: - `FalseNeg`(atives): could have been assigned but was not (individual + parent both genotyped or dummyfiable; P1only in `PedCompare`).
- `FalsePos`(itives): no parent in reference pedigree, but one was assigned based on the simulated data (P2only)
- `Mismatch`: different parents between the pedigrees

`cat`
: Category of individual + parent, as a two-letter code where the first letter indicates the focal individual and the second the parent; G=Genotyped, D=Dummy, T=Total

`parent`
: dam or sire

## Assumptions

Because the actual true pedigree is (typically) unknown, the provided reference pedigree is used as a stand-in and assumed to be the true pedigree, with unrelated founders. It is also assumed that the probability to be genotyped is equal for all parents; in each iteration, a new random set of parents (proportion set by `ParMis`) is mimicked to be non-genotyped. In addition, SNPs are assumed to segregate independently.

## Object size

The size in Kb of the returned list can become pretty big, as each of the inferred pedigrees is included. When running `EstConf` many times for a range of parameter values, it may be prudent to save the required summary statistics for each run rather than the full output.

## See Also

SimGeno,sequoia,PedCompare.

## Examples

```
data(Ped_HSg5, LH_HSg5, package="sequoia")

## Example A: parentage assignment only
conf.A <- EstConf(Pedigree = Ped_HSg5, LifeHistData = LH_HSg5,
   args.sim = list(nSnp = 100, SnpError = 5e-3, ParMis=c(0.2, 0.5)),
   args.seq = list(Module="par", Err=1e-3, Tassign=0.5), nSim = 3)

# parent-pair confidence, per category:
conf.A$ConfProb

# calculate (correct) assignment rates (ignores co-parent)
1 - apply(conf.A$PedErrors, c(1,3), sum, na.rm=TRUE)

## Example B: with sibship clustering, based on sequoia inferred pedigree
RealGenotypes <- SimGeno(Ped = Ped_HSg5, nSnp = 100,
                         ParMis=c(0.19,0.53), SnpError = 6e-3)
SeqOUT <- sequoia(GenoM = RealGenotypes,
                  LifeHistData = LH_HSg5,
                  Err=5e-3, Module="ped",
                  quiet=TRUE, Plot=FALSE)

conf.B <- EstConf(Pedigree = SeqOUT$Pedigree,
              LifeHistData = LH_HSg5,
               args.sim = list(nSnp = 100, SnpError = 5e-3,
                               ParMis=c(0.2, 0.5)),
              args.seq = list(Err=5e-3, Module="ped"),
              nSim = 2, nCores=2)
conf.B$ConfProb

Ped.withConf <- getAssignCat(Pedigree = SeqOUT$Pedigree,
                             SNPd = rownames(RealGenotypes))
Ped.withConf <- merge(Ped.withConf, conf.B$ConfProb, all.x=TRUE, sort=FALSE)
Ped.withConf <- Ped.withConf[, c("id","dam","sire", "dam.conf", "sire.conf",
                                 "id.cat", "dam.cat", "sire.cat")]
head(Ped.withConf[Ped.withConf$dam.cat=="G", ])
head(Ped.withConf[Ped.withConf$dam.cat=="D", ])


## P(actual FS | inferred as FS) etc.
PairL <- list()
for (i in 1:length(conf.A$Pedigree.inferred)) {  # nSim
  cat(i, "\t")
  PairL[[i]] <- ComparePairs(conf.A$Pedigree.reference,
                             conf.A$Pedigree.inferred[[i]],
                             GenBack=1, patmat=TRUE, ExcludeDummies = TRUE,
                             Return="Counts")
```

```
}
# P(actual relationship (Ped1) | inferred relationship (Ped2))
PairA <- plyr::laply(PairL, function(M) sweep(M, 2, colSums(M), "/"))
PairRel.prop <- apply(PairA, 2:3, mean, na.rm=TRUE)  # mean across simulations
round(PairRel.prop, 2)
#' # or: P(inferred relationship | actual relationship)
PairA2 <- plyr::laply(PairL, function(M) sweep(M, 1, rowSums(M), "/"))
```

---

`FieldMums_griffin` *Example field-observed mothers: griffins*

---

## Description

Example field pedigree used in vignette for `PedCompare` example. Non-genotyped females have IDs 'BlueRed', 'YellowPink', etc.

## Usage

```
data(FieldMums_griffin)
```

## Format

A data frame with 144 rows and 2 variables (id, mum)

## Author(s)

Jisca Huisman, <jisca.huisman@gmail.com>

## See Also

`SeqOUT_griffin` for a sequoia run on simulated genotype data, `Ped_griffin` for the 'true' pedigree.

---

`FindFamilies` *Assign Family IDs*

---

## Description

Add a column with family IDs (FIDs) to a pedigree, with each number denoting a cluster of connected individuals.

## Usage

```
FindFamilies(Ped = NULL, SeqList = NULL, UseMaybeRel = FALSE)
```

### Arguments

| | |
|---|---|
| Ped | dataframe with columns id - parent1 - parent2; only the first 3 columns will be used. |
| SeqList | list as returned by sequoia. If 'Ped' is not provided, the element 'Pedigree' from this list will be used if present, and element 'Pedigreepar' otherwise. |
| UseMaybeRel | use SeqList$MaybeRel, the dataframe with probable but non-assigned relatives, to assign additional family IDs? |

### Details

This function repeatedly finds all ancestors and all descendants of each individual in turn, and ensures they all have the same Family ID. Not all connected individuals are related, e.g. all grandparents of an individual will have the same FID, but will typically be unrelated.

When UseMaybeRel = TRUE, probable relatives are added to existing family clusters, or existing family clusters may be linked together. Currently no additional family clusters are created.

### Value

A dataframe with the provided pedigree, with a column 'FID' added.

---

## GenoConvert *Convert Genotype Data*

---

### Description

Convert genotype data in various formats to sequoia's 1-column-per-marker format or Colony's 2-columns-per-marker format.

### Usage

```
GenoConvert(
  InData = NULL,
  InFile = NULL,
  InFormat = "raw",
  OutFile = NA,
  OutFormat = "seq",
 Missing = c("-9", "??", "?", "NA", "NULL", c("0")[InFormat %in% c("col", "ped")]),
  sep = c(" ", "\t", ",", ";"),
  header = NA,
  IDcol = NA,
  FIDcol = NA,
  FIDsep = "__",
  dropcol = NA,
  quiet = FALSE
)
```

## Arguments

| | |
|---|---|
| InData | dataframe or matrix with genotypes to be converted. |
| InFile | character string with name of genotype file to be converted. |
| InFormat | One of 'single', 'double', 'col', 'ped', 'raw', or 'seq', see Details. |
| OutFile | character string with name of converted file. If NA, return matrix with genotypes in console (default); if NULL, write to 'GenoForSequoia.txt' in current working directory. |
| OutFormat | as InFormat; only 'seq' and 'col' are implemented. |
| Missing | vector with symbols interpreted as missing data. '0' is missing data for InFormats 'col' and 'ped' only. |
| sep | vector with field separator strings that will be tried on InFile. The OutFile separator uses the write.table default, i.e. one blank space. |
| header | a logical value indicating whether the file contains a header as its first line. If NA (default), set to TRUE for 'raw', and FALSE otherwise. |
| IDcol | number giving the column with individual IDs; 0 indicates the rownames (for InData only). If NA (default), set to 2 for InFormat 'raw' and 'ped', and otherwise to 1 for InFile and 0 (rownames) for InData, except when InData has a column labeled 'ID'. |
| FIDcol | column with the family IDs, if any are wished to be used. This is column 1 for InFormat 'raw' and 'seq', but those are by default not used. |
| FIDsep | string used to paste FID and IID together into a composite-ID (value passed to paste's collapse). This joining can be reversed using PedStripFID. |
| dropcol | columns to exclude from the output data, on top of IDcol and FIDcol (which become rownames). When NA, defaults to columns 3-6 for InFormat 'raw' and 'seq'. Can also be used to drop some SNPs, see example below on how to do this for the 2-columns-per-SNP input formats. |
| quiet | suppress messages and warnings. |

## Details

The first two arguments are interchangeable, and can be given unnamed. The first argument is assumed to be a file name if it is of class 'character' and length 1, and to be the genetic data if it is a matrix or dataframe.

## Value

A genotype matrix in the specified output format. If 'OutFile' is specified, the matrix is written to this file and nothing is returned inside R. When converting to 0/1/2 format, 2 is the homozygote for the minor allele, and 0 the homozygote for the major allele.

## Input formats

The following formats can be specified by InFormat:

**seq** (sequoia) genotypes are coded as 0, 1, 2, missing as $-9$, in 1 column per marker. Column 1 contains IDs, there is no header row.

**raw** (PLINK) genotypes are coded as 0, 1, 2, missing as NA, in 1 column per marker. The first 6 columns are descriptive (1:FID, 2:IID, 3 to 6 ignored), and there is a header row. This is produced by PLINK's option –recodeA

**ped** (PLINK) genotypes are coded as A, C, T, G, missing as 0, in 2 columns per marker. The first 6 columns are descriptive (1:FID, 2:IID, 3 to 6 ignored).

**col** (Colony) genotypes are coded as numeric values, missing as 0, in 2 columns per marker. Column 1 contains IDs.

**single** 1 column per marker, otherwise unspecified

**double** 2 columns per marker, otherwise unspecified

For each `InFormat`, its default values for `Missing`,`header`,`IDcol`,`FIDcol`, and `dropcol` can be overruled by specifying the corresponding input parameters.

## Error messages

Occasionally when reading in a file `GenoConvert` may give an error that 'rows have unequal length'. GenoConvert makes use of `readLines` and `strsplit`, which is much faster than `read.table` for large datafiles, but also more sensitive to unusual line endings, unusual end-of-file characters, or invisible characters (spaces or tabs) after the end of some lines. In these cases, try to read the data from file using read.table or read.csv, and then use `GenoConvert` on this dataframe or matrix, see example.

## Author(s)

Jisca Huisman, <jisca.huisman@gmail.com>

## See Also

`CheckGeno`,`SnpStats`,`LHConvert`.

## Examples

```
## Not run:
# Requires PLINK installed & in system PATH:

# tinker with window size, window overlap and VIF to get a set of
# 400 - 800 markers (100-200 enough for just parentage):
system("cmd", input = "plink --file mydata --indep 50 5 2")
system("cmd", input = "plink --file mydata --extract plink.prune.in
  --recodeA --out PlinkOUT")

GenoM <- GenoConvert(InFile = "PlinkOUT.raw")

# save time on file conversion next time:
write.table(GenoM, file="Geno_sequoia.txt", quote=FALSE, col.names=FALSE)
GenoM <- as.matrix(read.table("Geno_sequoia.txt", row.names=1, header=FALSE))

# drop some SNPs, e.g. after a warning of >2 alleles:
dropSNP <- c(5,68,101,128)
GenoM <- GenoConvert(ColonyFile, InFormat = "col",
```

```
                    dropcol = 1 + c(2*dropSNP-1, 2*dropSNP) )

# circumvent a 'rows have unequal length' error:
GenoTmp <- as.matrix(read.table("mydata.txt", header=TRUE, row.names=1))
GenoM <- GenoConvert(InData=GenoTmp, InFormat="single", IDcol=0)

## End(Not run)
```

---

`getAssignCat` *Assignability of Reference Pedigree*

---

## Description

Identify which individuals are SNP genotyped, and which can potentially be substituted by a dummy individual ('Dummifiable').

## Usage

```
getAssignCat(Pedigree, SNPd, minSibSize = "1sib1GP")
```

## Arguments

| | |
|---|---|
| `Pedigree` | dataframe with columns id-dam-sire. Reference pedigree. |
| `SNPd` | character vector with ids of genotyped individuals. |
| `minSibSize` | minimum requirements to be considered 'dummifiable':<br>• '1sib' : sibship of size 1, i.e. the non-genotyped individual has at least 1 genotyped offspring. If there is no sibship-grandparent this isn't really a sibship, but can be useful in some situations. Used by `CalcOHLLR`.<br>• '1sib1GP': sibship of size 1 with at least 1 genotyped grandparent. The minimum to be potentially assignable by `sequoia`.<br>• '2sib': at least 2 siblings, with or without grandparents. Used by `PedCompare`.<br>. |

## Details

It is assumed that all individuals in `SNPd` have been genotyped for a sufficient number of SNPs. To identify samples with a too-low call rate, use `CheckGeno`. To calculate the call rate for all samples, see the examples below.

Some parents indicated here as assignable may never be assigned by sequoia, for example parent-offspring pairs where it cannot be determined which is the older of the two, or grandparents that are indistinguishable from full avuncular (i.e. genetics inconclusive because the candidate has no parent assigned, and ageprior inconclusive).

## Value

The Pedigree dataframe with 3 additional columns, id.cat, dam.cat and sire.cat, with coding similar to that used by PedCompare:

| | |
|---|---|
| G | Genotyped |
| D | Dummy or 'dummifiable' |
| X | Not genotyped and not dummifiable, or no parent in pedigree |

## Examples

```
data(Ped_HSg5, SimGeno_example, package="sequoia")
PedA <- getAssignCat(Ped_HSg5, rownames(SimGeno_example))
tail(PedA)
table(PedA$dam.cat, PedA$sire.cat, useNA="ifany")

# calculate call rate
## Not run:
CallRates <- apply(MyGenotypes, MARGIN=1,
                   FUN = function(x) sum(x!=-9)) / ncol(MyGenotypes)
hist(CallRates, breaks=50, col="grey")
GoodSamples <- rownames(MyGenotypes)[ CallRates > 0.8]
# threshold depends on total number of SNPs, genotyping errors, proportion
# of candidate parents that are SNPd (sibship clustering is more prone to
# false positives).
PedA <- getAssignCat(MyOldPedigree, rownames(GoodSamples))

## End(Not run)
```

---

# getGenerations *Count Generations*

---

## Description

For each individual in a pedigree, count the number of generations since its most distant pedigree founder.

## Usage

```
getGenerations(Ped, StopIfInvalid = TRUE)
```

## Arguments

| | |
|---|---|
| Ped | dataframe, pedigree with the first three columns being id - dam - sire. Column names are ignored, as are additional columns. |
| StopIfInvalid | if a pedigree loop is detected, stop with an error (TRUE, default) or return the Pedigree, to see where the problem(s) occur. |

## Value

A vector with the generation number for each individual, starting at 0 for founders. NA indicates a pedigree loop where an individual is its own ancestor (or that the pedigree has >1000 generations). Returned invisibly to be a part of QC.

---

`GetLLRAge` *LLR-age from Ageprior Matrix*

---

## Description

Get log10-likelihood ratios for a specific age difference from matrix `AgePriorExtra`.

## Usage

```
GetLLRAge(AgePriorExtra, agedif, patmat)
```

## Arguments

| | |
|---|---|
| `AgePriorExtra` | matrix in `sequoia` output |
| `agedif` | vector with age differences, in whole numbers. Must occur in rownames of `AgePriorExtra`. |
| `patmat` | numeric vector; choose maternal (1), paternal (2) relatives, or for each relationship the most-likely alternative (3). |

## Value

A matrix with `nrow` equal to the length of `agedif`, and 7 columns: PO-FS-HS-GP-FA-HA-U.

## Examples

```
data(SeqOUT_griffin, package="sequoia")
PairsG <- data.frame(ID1="i122_2007_M",
                     ID2 = c("i124_2007_M", "i042_2003_F", "i083_2005_M"),
                     AgeDif = c(0,4,2))
cbind(PairsG,
      GetLLRAge(SeqOUT_griffin$AgePriorExtra,
                agedif = PairsG$AgeDif, patmat=rep(2,3)))
```

`GetMaybeRel` *Find Putative Relatives*

## Description

Identify pairs of individuals likely to be related, but not assigned as such in the provided pedigree.

## Usage

```
GetMaybeRel(
  GenoM = NULL,
  SeqList = NULL,
  Pedigree = NULL,
  LifeHistData = NULL,
  AgePrior = NULL,
  ParSib = NULL,
  Module = "par",
  Complex = "full",
  Herm = "no",
  Err = 1e-04,
  ErrFlavour = "version2.0",
  MaxMismatch = NA,
  Tassign = 0.5,
  Tfilter = -2,
  MaxPairs = 7 * nrow(GenoM),
  quiet = FALSE
)
```

## Arguments

| | |
|---|---|
| `GenoM` | numeric matrix with genotype data: One row per individual, and one column per SNP, coded as 0, 1, 2 or -9 (missing). See also `GenoConvert`. |
| `SeqList` | list with output from `sequoia`. `SeqList$Pedigree` is used if present, and `SeqList$PedigreePar` otherwise, and overrides the input parameter `Pedigree`. If 'Specs' is present, its elements override all input parameters with the same name. The list elements 'LifeHist', 'AgePriors', and 'ErrM' are also used if present, and similarly override the corresponding input parameters. |
| `Pedigree` | dataframe with id - dam - sire in columns 1-3. May include non-genotyped individuals, which will be treated as dummy individuals. When provided, all likelihoods (and thus all maybe-relatives) are conditional on this pedigree. Note: `SeqList$Pedigree` or `SeqList$PedigreePar` take precedent (for this function only). |
| `LifeHistData` | dataframe with 3 columns (optionally 5):<br>**ID** max. 30 characters long<br>**Sex** 1 = female, 2 = male, 3 = unknown, 4 = hermaphrodite, other numbers or NA = unknown |

**BirthYear** birth or hatching year, integer, with missing values as NA or any negative value.

**BY.min** minimum birth year, only used if BirthYear is missing

**BY.max** maximum birth year, only used if BirthYear is missing

If the species has multiple generations per year, use an integer coding such that the candidate parents' 'Birth year' is at least one smaller than their putative offspring's. Column names are ignored, so ensure column order is ID - sex - birth year (- BY.min - BY.max). Individuals do not need to be in the same order as in 'GenoM', nor do all genotyped individuals need to be included.

`AgePrior` Agepriors matrix, as generated by `MakeAgePrior` and included in the `sequoia` output. Affects which relationships are considered possible (only those where $P(A|R)/P(A) > 0$).

`ParSib` either 'par' to check for putative parent-offspring pairs only, or 'sib' to check for all types of first and second degree relatives. This argument will be deprecated, please use `Module`.

`Module` type of relatives to check for. One of

**par** parent - offspring pairs

**ped** all first and second degree relatives

When 'par', all pairs are returned that are more likely parent-offspring than unrelated, potentially including pairs that are even more likely to be otherwise related.

`Complex` Breeding system complexity. Either "full" (default), "simp" (simplified, no explicit consideration of inbred relationships), "mono" (monogamous).

`Herm` Hermaphrodites, either "no", "A" (distinguish between dam and sire role, default if at least 1 individual with sex=4), or "B" (no distinction between dam and sire role). Both of the latter deal with selfing.

`Err` estimated genotyping error rate, as a single number or 3x3 matrix. Details below. The error rate is presumed constant across SNPs, and missingness is presumed random with respect to actual genotype.

`ErrFlavour` function that takes `Err` (single number) as input, and returns a 3x3 matrix of observed (columns) conditional on actual (rows) genotypes, or choose from in-built options 'version2.0', 'version1.3', or 'version1.1', referring to the sequoia version in which they were the default. Ignored if `Err` is a matrix. See `ErrToM`.

`MaxMismatch` **DEPRECATED AND IGNORED**. Now calculated automatically using `CalcMaxMismatch`.

`Tassign` minimum LLR required for acceptance of proposed relationship, relative to next most likely relationship. Higher values result in more conservative assignments. Must be zero or positive.

`Tfilter` threshold log10-likelihood ratio (LLR) between a proposed relationship versus unrelated, to select candidate relatives. Typically a negative value, related to the fact that unconditional likelihoods are calculated during the filtering steps. More negative values may decrease non-assignment, but will increase computational time.

`MaxPairs` the maximum number of putative pairs to return.

`quiet` logical, suppress messages.

## Details

When `Module="par"`, the age difference of the putative pair is temporarily set to NA so that genetic parent-offspring pairs declared to be born in the same year may be discovered. When `Module="ped"`, only relationships possible given the age difference, if known from the LifeHistData, are considered.

## Value

A list with

`MaybePar` or `MaybeRel`
: A dataframe with non-assigned likely relatives, with columns ID1 - ID2 - TopRel - LLR - OH - BirthYear1 - BirthYear2 - AgeDif - Sex1 - Sex2 - SNPdBoth

`MaybeTrio`
: A dataframe with non-assigned parent-parent-offspring trios, with columns id - parent1 - parent2 - LLRparent1 - LLRparent2 - LLRpair - OHparent1 - OHparent2 - MEpair - SNPd.id.parent1 - SNPd.id.parent2

The following categories are used in column 'TopRel', indicating the most likely relationship category:

| | |
|---|---|
| `PO` | Parent-Offspring |
| `FS` | Full Siblings |
| `HS` | Half Siblings |
| `GP` | GrandParent - grand-offspring |
| `FA` | Full Avuncular (aunt/uncle) |
| `2nd` | 2nd degree relatives, not enough information to distinguish between HS,GP and FA |
| `Q` | Unclear, but probably 1st, 2nd or 3rd degree relatives |

## See Also

`sequoia` to identify likely pairs of duplicate genotypes and for pedigree reconstruction; `GetRelM` to identify all pairs of relatives in a pedigree; `CalcPairLL` for the likelihoods underlying the LLR.

## Examples

```
SeqOUT <- sequoia(GenoM = SimGeno_example,
                  LifeHistData = LH_HSg5,
                  Module = "par",
                  quiet=TRUE, Plot=FALSE)
MaybePO <- GetMaybeRel(GenoM = SimGeno_example,
                       SeqList = SeqOUT)
head(MaybePO$MaybePar)

# instead of providing the entire SeqList, one may
# specify the relevant elements separately
Maybe <- GetMaybeRel(GenoM = SimGeno_example,
                     Pedigree = SeqOUT$PedigreePar,
```

```
                    LifeHistData = LH_HSg5,
                    Err=0.0001, Complex = "full",
                    Module = "ped")
head(Maybe$MaybeRel)

# visualise results, turn dataframe into matrix first:
MaybeM <- GetRelM(Pairs=Maybe$MaybeRel)
PlotRelPairs(MaybeM)
```

---

GetRelM *Matrix with Pairwise Relationships*

---

## Description

Generate a matrix or 3D array with all pairwise relationships from a pedigree or dataframe with pairs.

## Usage

```
GetRelM(
  Pedigree = NULL,
  Pairs = NULL,
  GenBack = 1,
  patmat = FALSE,
  Return = "Matrix"
)
```

## Arguments

| | |
|---|---|
| Pedigree | dataframe with columns id - dam - sire. |
| Pairs | dataframe with columns ID1 - ID2 - Rel. |
| GenBack | number of generations back to consider; 1 returns parent-offspring and sibling relationships, 2 also returns grand-parental, avuncular and first cousins. |
| patmat | logical, distinguish between paternal versus maternal relative pairs? For avuncular pairs, the distinction is never made. |
| Return | 'Matrix' or 'Array'. The former returns an N x N matrix with the closest relationship between each pair, the latter an N x N x R array with for each of the R considered relationships whether it exists between the pair (1) or not (0). See Details below. |

## Details

Double relationships are ignored when `Return='Matrix'`, but not when `Return='Array'`. For example, when A and B are both mother-offspring and paternal siblings (A mated with her father to produce B), only the mother-offspring relationship will be indicated when `Return='Matrix'`.

Note that full siblings are the exception to this rule: in the `Array` they will be indicated as 'FS' only, and not as 'MHS' or 'PHS'. Similarly, full avuncular pairs are not indicated as 'HA'.

When `Pairs` is provided, `GenBack` and `patmat` are ignored, and no check is performed if the abbreviations are compatible with other functions.

## Value

If `Return='Matrix'`, an N x N square matrix, with N equal to the number of rows in `Pedigree` (after running `PedPolish`) or the number of unique individuals in `Pairs`. If `Return='Array'`, an N x N x R array is returned, with R, the number of different relationships, determined by `GenBack` and `patmat`.

The following abbreviations are used within the returned `Matrix`, or as names of the 3rd dimension in the `Array`:

| | |
|---|---|
| `S` | Self |
| `M` | Mother |
| `P` | Father |
| `MP` | Mother or Father (`patmat=FALSE`) |
| `O` | Offspring |
| `FS` | Full sibling |
| `MHS` | Maternal half-sibling |
| `PHS` | Paternal half-sibling |
| `XHS` | other half-sibling (hermaphrodites) |
| `HS` | half-sibling (`patmat=FALSE`) |
| `MGM` | Maternal grandmother |
| `MGF` | Maternal grandfather |
| `PGM` | Paternal grandmother |
| `PGF` | Paternal grandfather |
| `GP` | Grandparent (`patmat=FALSE`) |
| `GO` | Grand-offspring |
| `FA` | Full avuncular; maternal or paternal aunt or uncle |
| `HA` | Half avuncular |
| `FN` | Full nephew/niece |
| `HN` | Half nephew/niece |
| `FC1` | Full first cousin |
| `DFC1` | Double full first cousin |
| `U` | Unrelated (or otherwise related) |

## See Also

ComparePairs for comparing pairwise relationships between two pedigrees; PlotRelPairs.

## Examples

```
data(Ped_griffin)
Rel.griffin <- GetRelM(Ped_griffin, patmat=TRUE, GenBack=2)
table(c(Rel.griffin))
# turning matrix into vector first makes table() much faster
PlotRelPairs(Rel.griffin)
```

---

# Inherit *Inheritance patterns*

---

## Description

Inheritance patterns used by SimGeno for non-autosomal SNPs, identical to those in Inherit.xlsx

## Usage

```
data(Inherit)
```

## Format

An array with the following dimensions:

**d1** type: autosomal, x-chromosome, y-chromosome, or mtDNA

**d2** offspring sex: female, male, or unknown

**d3** offspring genotype: aa (0), aA (1), Aa (1), or AA (2)

**d4** mother genotype

**d5** father genotype

## Author(s)

Jisca Huisman, <jisca.huisman@gmail.com>

## See Also

SimGeno

LHConvert *Extract Sex and Birth Year from PLINK File*

## Description

Convert the first six columns of a PLINK .fam, .ped or .raw file into a three-column lifehistory file for sequoia. Optionally FID and IID are combined.

## Usage

```
LHConvert(
  PlinkFile = NULL,
  UseFID = FALSE,
  SwapSex = TRUE,
  FIDsep = "__",
  LifeHistData = NULL
)
```

## Arguments

| | |
|---|---|
| PlinkFile | character string with name of genotype file to be converted. |
| UseFID | use the family ID column. The resulting ids (rownames of GenoM) will be in the form FID__IID. |
| SwapSex | change the coding from PLINK default (1=male, 2=female) to sequoia default (1=female, 2=male); any other numbers are set to NA. |
| FIDsep | characters inbetween FID and IID in composite-ID. By default a double underscore is used, to avoid problems when some IIDs contain an underscore. Only used when UseFID=TRUE. |
| LifeHistData | dataframe with additional sex and birth year info. In case of conflicts, LifeHistData takes priority, with a warning. If UseFID=TRUE, IDs in LifeHistData are assumed to be already as FID__IID. |

## Details

The first 6 columns of PLINK .fam, .ped and .raw files are by default FID - IID - father ID (ignored) - mother ID (ignored) - sex - phenotype.

## Value

A dataframe with id, sex and birth year, which can be used as input for sequoia.

## See Also

GenoConvert, PedStripFID to reverse UseFID.

## Examples

```
## Not run:
# combine FID and IID in dataframe with additional sex & birth years
ExtraLH$FID_IID <- paste(ExtraLH$FID, ExtraLH$IID, sep = "__")
LH.new <- LHConvert(PlinkFile, UseFID = TRUE, FIDsep = "__",
                    LifeHistData = ExtraLH)

## End(Not run)
```

---

LH_griffin *Example lifehistory data: griffins*

---

## Description

Example lifehistory data for griffin pedigree

## Usage

```
data(LH_griffin)
```

## Format

A data frame with 200 rows and 3 variables (ID, Sex, BirthYear)

## Author(s)

Jisca Huisman, <jisca.huisman@gmail.com>

## See Also

Ped_griffin, SeqOUT_griffin

---

LH_HSg5 *Example life history file*

---

## Description

This is the lifehistory file associated with Ped_HSg5, which is **Pedigree II** in the paper.

## Usage

```
data(LH_HSg5)
```

## Format

A data frame with 1000 rows and 3 variables:

**ID** Female IDs start with 'a', males with 'b'; the next 2 numbers give the generation number (00 – 05), the last 3 numbers the individual ID number (runs continuously across all generations)

**Sex** 1 = female, 2 = male

**BirthYear** from 2000 (generation 0, founders) to 2005

## Author(s)

Jisca Huisman, <jisca.huisman@gmail.com>

## References

Huisman, J. (2017) Pedigree reconstruction from SNP data: Parentage assignment, sibship clustering, and beyond. Molecular Ecology Resources 17:1009–1024.

## See Also

Ped_HSg5 sequoia

---

# MakeAgePrior *Age Priors*

---

## Description

Estimate probability ratios $P(R|A)/P(R)$ for age differences A and five categories of parent-offspring and sibling relationships R.

## Usage

```
MakeAgePrior(
  Pedigree = NULL,
  LifeHistData = NULL,
  MaxAgeParent = NULL,
  Discrete = NULL,
  Flatten = NULL,
  lambdaNW = -log(0.5)/100,
  Smooth = TRUE,
  Plot = TRUE,
  Return = "LR",
  quiet = FALSE
)
```

## Arguments

| | |
|---|---|
| `Pedigree` | dataframe with id - dam - sire in columns 1-3, and optional column with birth years. Other columns are ignored. |
| `LifeHistData` | dataframe with 3 or 5 columns: id - sex (not used) - birth year (- BY.min - BY.max), with unknown birth years coded as negative numbers or NA. Column names are ignored, so the column order is important. "Birth year" may be in any arbitrary discrete time unit relevant to the species (day, month, decade), as long as parents are never born in the same time unit as their offspring. It may include individuals not in the pedigree, and not all individuals in the pedigree need to be in LifeHistData. |
| `MaxAgeParent` | maximum age of a parent, a single number (max across dams and sires) or a vector of length two (dams, sires). If NULL, it will be estimated from the pedigree. See details below. |
| `Discrete` | discrete generations? By default (NULL), discrete generations are assumed if all parent-offspring pairs have an age difference of 1, and all siblings an age difference of 0, and there are at least 20 pairs of each category (mother, father, maternal sibling, paternal sibling). Otherwise, overlapping generations are presumed. When `Discrete=TRUE` (explicitly or deduced), `Smooth` and `Flatten` are always automatically set to `FALSE`. Use `Discrete=FALSE` to enforce (potential for) overlapping generations. |
| `Flatten` | logical. To deal with small sample sizes for some or all relationships, calculate weighed average between the observed age difference distribution among relatives and a flat (0/1) distribution. When `Flatten=NULL` (the default) automatically set to TRUE when there are fewer than 20 parents with known age of either sex assigned, or fewer than 20 maternal or paternal siblings with known age difference. Also advisable if the sampled relative pairs with known age difference are non-typical of the pedigree as a whole. |
| `lambdaNW` | control weighing factors when `Flatten=TRUE`. Weights are calculated as $W(R) = 1 - exp(-lambdaNW * N(R))$, where $N(R)$ is the number of pairs with relationship R for which the age difference is known. Large values (>0.2) put strong emphasis on the pedigree, small values (<0.0001) cause the pedigree to be ignored. Default results in $W = 0.5$ for $N = 100$. |
| `Smooth` | smooth the tails of and any dips in the distribution? Sets dips (<10% of average of neighbouring ages) to the average of the neighbouring ages, sets the age after the end (oldest observed age) to LR(end)/2, and assigns a small value (0.001) to the ages before the front (youngest observed age) and after the new end. Peaks are not smoothed out, as these are less likely to cause problems than dips, and are more likely to be genuine characteristics of the species. Is set to `FALSE` when generations do not overlap (`Discrete=TRUE`). |
| `Plot` | plot a heatmap of the results? |
| `Return` | return only a matrix with the likelihood-ratio $P(A\|R)/P(A)$ (`"LR"`) or a list including also various intermediate statistics (`"all"`) ? |
| `quiet` | suppress messages. |

## Details

$\alpha_{A,R}$ is the ratio between the observed counts of pairs with age difference A and relationship R ($N_{A,R}$), and the expected counts if age and relationship were independent ($N_{.,.} * p_A * p_R$).

During pedigree reconstruction, $\alpha_{A,R}$ are multiplied by the genetic-only $P(R|G)$ to obtain a probability that the pair are relatives of type R conditional on both their age difference and their genotypes.

The age-difference prior is used for pairs of genotyped individuals, as well as for dummy individuals. This assumes that the propensity for a pair with a given age difference to both be sampled does not depend on their relationship, so that the ratio $P(A|R)/P(A)$ does not differ between sampled and unsampled pairs.

For further details, see the vignette.

## Value

A matrix with the probability ratio of the age difference between two individuals conditional on them being a certain type of relative ($P(A|R)$) versus being a random draw from the sample ($P(A)$). Assuming conditional independence, this equals the probability ratio of being a certain type of relative conditional on the age difference, versus being a random draw.

The matrix has one row per age difference (0 - nAgeClasses) and five columns, one for each relationship type, with abbreviations:

| | |
|---|---|
| `M` | Mothers |
| `P` | Fathers |
| `FS` | Full siblings |
| `MS` | Maternal half-siblings |
| `PS` | Paternal half-siblings |

When `Return`='all', a list is returned with the following elements:

| | |
|---|---|
| `BirthYearRange` | vector length 2 |
| `MaxAgeParent` | vector length 2, see details |
| `tblA.R` | matrix with the counts per age difference (rows) / relationship (columns) combination, plus a column 'X' with age differences across all pairs of individuals |
| `PA.R` | Proportions, i.e. `tblA.R` divided by its `colSums`, with full-sibling correction applied if necessary (see vignette). |
| `LR.RU.A.raw` | Proportions `PA.R` standardised by global age difference distribution (column 'X'); `LR.RU.A` prior to flattening and smoothing |
| `Weights` | vector length 4, the weights used to flatten the distributions |
| `LR.RU.A` | the ageprior, flattend and/or smoothed |
| `Specs.AP` | the names of the input `Pedigree` and `LifeHistData` (or `NULL`), `lambdaNW`, and the 'effective' settings (i.e. after any automatic update) of `Discrete`, `Smooth`, and `Flatten`. |

## CAUTION

The small sample correction with `Smooth` and/or `Flatten` prevents errors in one dataset, but may introduce errors in another; a single solution that fits to the wide variety of life histories and datasets is impossible. Please do inspect the matrix, e.g. with `PlotAgePrior`, and adjust the input parameters and/or the output matrix as necessary.

## Single cohort

When all individuals in `LifeHistData` have the same birth year, it is assumed that `Discrete=TRUE` and `MaxAgeParent=1`. Consequently, it is assumed there are no avuncular pairs present in the sample; cousins are considered as alternative. To enforce overlapping generations, and thereby the consideration of full- and half- avuncular relationships, set `MaxAgeParent` to some value greater than 1.

When no birth year information is given at all, a single cohort is assumed, and the same rules apply.

## Other time units

"Birth year" may be in any arbitrary time unit relevant to the species (day, month, decade), as long as parents are always born before their putative offspring, and never in the same time unit (e.g. parent's BirthYear= 1 (or 2001) and offspring BirthYear=5 (or 2005)). Negative numbers and NA's are interpreted as unknown, and fractional numbers are not allowed.

## MaxAgeParent

The maximum parental age for each sex equals the maximum of:

- the maximum age of parents in `Pedigree`,
- the input parameter `MaxAgeParent`,
- the maximum range of birth years in `LifeHistData` (including BY.min and BY.max). Only used if both of the previous are `NA`, or if there are fewer than 20 parents of either sex assigned.
- 1, if `Discrete=TRUE` or the previous three are all `NA`

If the age distribution of assigned parents does not capture the maximum possible age of parents, it is advised to specify `MaxAgeParent` for one or both sexes. Not doing so may hinder subsequent assignment of both dummy parents and grandparents.

@section grandparents & avuncular The agepriors for grand-parental and avuncular pairs is calculated from these by `sequoia`, and included in its output as 'AgePriorExtra'.

## See Also

`sequoia` and its argument `args.AP`, `PlotAgePrior` for visualisation. The age vignette gives further details, mathematical justification, and some examples.

## Examples

```
# without pedigree or lifehistdata:
MakeAgePrior()
MakeAgePrior(MaxAgeParent = c(2,3))
```

```
MakeAgePrior(Discrete=TRUE)

# single cohort:
MakeAgePrior(LifeHistData = data.frame(ID = letters[1:5], Sex=3,
  BirthYear=1984))

# overlapping generations:
data(Ped_griffin, SeqOUT_griffin, package="sequoia")
# without pedigree: MaxAgeParent = max age difference between any pair +1
MakeAgePrior(LifeHistData = SeqOUT_griffin$LifeHist)
# with pedigree:
MakeAgePrior(Pedigree=Ped_griffin,
             LifeHistData=SeqOUT_griffin$LifeHist,
             Smooth=FALSE, Flatten=FALSE)
# with small-sample correction:
MakeAgePrior(Pedigree=Ped_griffin,
             LifeHistData=SeqOUT_griffin$LifeHist,
             Smooth=TRUE, Flatten=TRUE)
```

---

`MkGenoErrors` *Simulate Genotyping Errors*

---

## Description

Generate errors and missing values in a (simulated) genotype matrix.

## Usage

```
MkGenoErrors(
  SGeno,
  CallRate = 0.99,
  SnpError = 5e-04,
  ErrorFM = function(E) {    matrix(c(1 - E - (E/2)^2, E, (E/2)^2, E/2, 1 - E, E/2,
    (E/2)^2, E, 1 - E - (E/2)^2), 3, 3, byrow = TRUE) },
  Error.shape = 0.5,
  CallRate.shape = 1
)
```

## Arguments

| | |
|---|---|
| `SGeno` | matrix with genotype data in Sequoia's format: 1 row per individual, 1 column per SNP, and genotypes coded as 0/1/2. |
| `CallRate` | either a single number for the mean call rate (genotyping success), OR a vector with the call rate at each SNP, OR a named vector with the call rate for each individual. In the third case, ParMis is ignored, and individuals in the pedigree (as id or parent) not included in this vector are presumed non-genotyped. |

| | |
|---|---|
| SnpError | mean per-locus genotyping error rate across SNPs, and a beta-distribution will be used to simulate the number of missing cases per SNP, OR a vector with the genotyping error for each SNP. |
| ErrorFM | function taking the error rate (scalar) as argument and returning a 4x4 or 3x3 matrix with probabilities that actual genotype i (rows) is observed as genotype j (columns). |
| Error.shape | first shape parameter (alpha) of beta-distribution of per-SNP error rates. A higher value results in a flatter distribution. |
| CallRate.shape | as Error.shape, for per-SNP call rates. |

## Value

The input genotype matrix, with some genotypes replaced, and some set to missing (-9).

## Examples

```
data(Ped_HSg5)
GenoM <- SimGeno(Ped = Ped_HSg5, nSnp = 100, ParMis = 0.2,
                 SnpError=0, CallRate=1)
GenoM.actual <- GenoM
LowQ <- sample.int(nrow(GenoM), 42)  # low-quality samples
GenoM[LowQ, ] <- MkGenoErrors(GenoM[LowQ, ], SnpError = 0.05)
GenoM[-LowQ, ] <- MkGenoErrors(GenoM[-LowQ, ], SnpError = 0.001)
ErrorCount <- sapply(1:nrow(GenoM), function(i) {
  sum(GenoM.actual[i,] != GenoM[i,] & GenoM[i,] != -9) } )
mean(ErrorCount[LowQ])
mean(ErrorCount[-LowQ])
```

---

# PedCompare *Compare Two Pedigrees*

---

## Description

Compare an inferred pedigree (Ped2) to a previous or simulated pedigree (Ped1), including comparison of sibship clusters and sibship grandparents.

## Usage

```
PedCompare(
  Ped1 = NULL,
  Ped2 = NULL,
  DumPrefix = c("F0", "M0"),
  SNPd = NULL,
  Symmetrical = TRUE,
  minSibSize = "2sib",
  Plot = TRUE
)
```

## Arguments

| | |
|---|---|
| Ped1 | first (e.g. original) pedigree, dataframe with columns id-dam-sire; only the first 3 columns will be used. |
| Ped2 | second pedigree, e.g. newly inferred SeqOUT$Pedigree or SeqOUT$PedigreePar, with columns id-dam-sire. |
| DumPrefix | character vector with the prefixes identifying dummy individuals in Ped2. Use 'F0' ('M0') to avoid matching to regular individuals with IDs starting with 'F' ('M'), provided Ped2 has fewer than 999 dummy females (males). |
| SNPd | character vector with IDs of genotyped individuals. If NULL, defaults to the IDs occurring in both Ped1 and Ped2 and not starting with any of the prefixes in DumPrefix. |
| Symmetrical | when determining the category of individuals (Genotyped/Dummy/X), use the 'highest' category across the two pedigrees (TRUE, default) or only consider Ped1 (Symmetrical = FALSE). |
| minSibSize | minimum requirements to be considered 'dummifiable', passed to getAssignCat:<br>• '1sib' : sibship of size 1, with or without grandparents. The latter aren't really a sibship, but can be useful in some situations.<br>• '1sib1GP': sibship of size 1 with at least 1 grandparent<br>• '2sib': at least 2 siblings, with or without grandparents (default) |
| Plot | show square Venn diagrams of counts? |

## Details

The comparison is divided into different classes of 'assignable' parents (getAssignCat). This includes cases where the focal individual and parent according to Ped1 are both Genotyped (G-G), as well as cases where the non-genotyped parent according to Ped1 can be lined up with a sibship Dummy parent in Ped2 (G-D), or where the non-genotyped focal individual in Ped1 can be matched to a dummy individual in Ped2 (D-G and D-D). If SNPd is NULL (the default), and DumPrefix is set to NULL, the intersect between the IDs in Pedigrees 1 and 2 is taken as the vector of genotyped individuals.

## Value

A list with

| | |
|---|---|
| Counts | A 7 x 5 x 2 named numeric array with the number of matches and mismatches, see below |
| Counts.detail | a large numeric array with number of matches and mismatches, with more detail for all possible combination of categories |
| MergedPed | A dataframe with side-by-side comparison of the two pedigrees |
| ConsensusPed | A consensus pedigree, with Pedigree 2 taking priority over Pedigree 1 |
| DummyMatch | Dataframe with all dummy IDs in Pedigree 2 (id.2), and the best-matching individual in Pedigree 1 (id.1). Also includes the class of the dam & sire, as well as counts of offspring per outcome class (off.Match, off.Mismatch, etc.) |

`Mismatch` A subset of MergedPed with mismatches between Ped1 and Ped2, as defined below

`Ped1only` as Mismatches, with parents in Ped1 that were not assigned in Ped2

`Ped2only` as Mismatches, with parents in Ped2 that were missing in Ped1

'MergedPed', 'Mismatch', 'Ped1only' and 'Ped2only' provide the following columns:

`id` All ids in both Pedigree 1 and 2. For dummy individuals, this is the id *in pedigree 2*

`dam.1, sire.1` parents in Pedigree 1

`dam.2, sire.2` parents in Pedigree 2

`id.r, dam.r, sire.r`
The *real* id of dummy individuals or parents in Pedigree 2, i.e. the best-matching non-genotyped individual in Pedigree 1, or "nomatch". If a sibship in Pedigree 1 is divided over 2 sibships in Pedigree 2, the smaller one will be denoted as "nomatch"

`id.dam.cat, id.sire.cat`
the category of the individual (first letter) and *highest category* of the dam (sire) in Pedigree 1 or 2: G=Genotyped, D=(potential) dummy, X=none. Individual, one-letter categories are generated by `getAssignCat`. Using the 'best' category from both pedigrees makes comparison between two inferred pedigrees symmetrical and more intuitive.

`dam.class, sire.class`
classification of dam and sire: Match, Mismatch, P1only, P2only, or '_' when no parent is assigned in either pedigree

The first dimension of `Counts` denotes the following categories:

`GG` Genotyped individual, assigned a genotyped parent in either pedigree

`GD` Genotyped individual, assigned a dummy parent, or at least 1 genotyped sibling or a genotyped grandparent in Pedigree 1)

`GT` Genotyped individual, total

`DG` Dummy individual, assigned a genotyped parent (i.e., grandparent of the sibship in Pedigree 2)

`DD` Dummy individual, assigned a dummy parent (i.e., avuncular relationship between sibships in Pedigree 2)

`DT` Dummy total

`TT` Total total, includes all genotyped individuals, plus non-genotyped individuals in Pedigree 1, plus non-replaced dummy individuals (see below) in Pedigree 2

The second dimension of `Counts` gives the outcomes:

`Total` The total number of individuals with a parent assigned in either or both pedigrees

`Match` The same parent is assigned in both pedigrees (non-missing). For dummy parents, it is considered a match if the inferred sibship which contains the most offspring of a non-genotyped parent, consists for more than half of this individual's offspring.

| | |
|---|---|
| Mismatch | Different parents assigned in the two pedigrees. When a sibship according to Pedigree 1 is split over two sibships in Pedigree 2, the smaller fraction is included in the count here. |
| P1only | Parent in Pedigree 1 but not 2; includes non-assignable parents (e.g. not genotyped and no genotyped offspring). |
| P2only | Parent in Pedigree 2 but not 1. |

The third dimension Counts separates between maternal and paternal assignments, where e.g. paternal 'DT' is the assignment of fathers to both maternal and paternal sibships (i.e., to dummies of both sexes).

In 'ConsensusPed', the priority used is parent.r (if not "nomatch") > parent.2 > parent.1. The columns 'id.cat', dam.cat' and 'sire.cat' have two additional levels compared to 'MergedPed':

| | |
|---|---|
| G | Genotyped |
| D | Dummy individual (in Pedigree 2) |
| R | Dummy individual in pedigree 2 replaced by best matching non-genotyped individual in pedigree 1 |
| U | Ungenotyped, Unconfirmed (parent in Pedigree 1, with no dummy match in Pedigree 2) |
| X | No parent in either pedigree |

### Assignable

Note that 'assignable' may be overly optimistic. Some parents from Ped1 indicated as assignable may never be assigned by sequoia, for example parent-offspring pairs where it cannot be determined which is the older of the two, or grandparents that are indistinguishable from full avuncular (i.e. genetics inconclusive because the candidate has no parent assigned, and ageprior inconclusive).

### Dummifiable

Considered as potential dummy individuals are all non-genotyped individuals in Pedigree 1 who have, according to either pedigree, at least 2 genotyped offspring, or at least one genotyped offspring and a genotyped parent.

### Mismatches

Perhaps unexpectedly, cases where all siblings are correct but a dummy parent rather than the genotyped Ped1-parent are assigned, are classified as a mismatch (for each of the siblings). These are typically due to a too low assumed genotyping error rate, a wrong parental birth year, or some other issue that requires user inspection. To identify these cases, CompairePairs may be of help.

### Genotyped 'mystery samples'

If Pedigree 2 includes samples for which the ID is unknown, the behaviour of PedCompare depends on whether the temporary IDs for these samples are included in SNPd. If they are included, matching (actual) IDs in Pedigree 1 will be flagged as mismatches (because the IDs differ). If they are not included in SNPd, or SNPd is not explicitly provided, matches are accepted, as the situation is indistinguishable from comparing dummy parents across pedigrees.

This is of course all conditional on relatives of the mystery sample being assigned in Pedigree 2.

## Author(s)

Jisca Huisman, <jisca.huisman@gmail.com>

## See Also

ComparePairs for comparison of all pairwise relationships in 2 pedigrees, EstConf for repeated simulate-reconstruct-compare, sequoia for the main pedigree reconstruction function, getAssignCat for all parents in the reference pedigree that could have been assigned.

## Examples

```
data(Ped_HSg5, SimGeno_example, LH_HSg5, package="sequoia")
SeqOUT <- sequoia(GenoM = SimGeno_example, LifeHistData = LH_HSg5,
  Err=0.0001, quiet=TRUE, Plot=FALSE)
# (Performance is better when using Err=0.001, but this makes for a more
# interesting example)

compare <- PedCompare(Ped1=Ped_HSg5, Ped2=SeqOUT$Pedigree)
compare$Counts["TT",,]  # totals only
compare$Counts[,,"dam"]  # dams only
# 2 mismatch & 3+1 non-assigned, due to simulated genotyping errors

# inspect 'assignable but non-assigned in Ped2', id + dam both genotyped:
compare$P1only[compare$P1only$id.dam.cat=="GG", ]
# further inspection:
compare$MergedPed[which(compare$MergedPed$dam.1=="a00013"), ]

# overview of all non-genotyped -- dummy matches
head(compare$DummyMatch)

# success of paternity assignment, if genotyped mother correctly assigned
dimnames(compare$Counts.detail)
compare$Counts.detail["G","G",,"Match",]
```

---

PedPolish *Fix Pedigree*

---

## Description

Ensure all parents & all genotyped individuals are included, remove duplicates, rename columns, and replace 0 by NA or v.v..

## Usage

```
PedPolish(
  Pedigree,
  gID = NULL,
```

```
  ZeroToNA = TRUE,
  NAToZero = FALSE,
  DropNonSNPd = TRUE,
  FillParents = FALSE,
  NullOK = FALSE,
  LoopCheck = TRUE,
  StopIfInvalid = TRUE
)
```

## Arguments

| | |
|---|---|
| Pedigree | dataframe where the first 3 columns are id, dam, sire. |
| gID | character vector with ids of genotyped individuals (rownames of genotype matrix). |
| ZeroToNA | logical, replace 0's for missing values by NA's (defaults to TRUE). |
| NAToZero | logical, replace NA's for missing values by 0's. If TRUE, ZeroToNA is automatically set to FALSE. |
| DropNonSNPd | logical, remove any non-genotyped individuals (but keep non-genotyped parents), & sort pedigree in order of gID. |
| FillParents | logical, for individuals with only 1 parent assigned, set the other parent to a dummy (without assigning siblings or grandparents). Makes the pedigree compatible with R packages and software that requires individuals to have either 2 or 0 parents, such as kinship. |
| NullOK | logical, is it OK for Ped to be NULL? Then NULL will be returned. |
| LoopCheck | logical, check for invalid pedigree loops by calling getGenerations. |
| StopIfInvalid | if a pedigree loop is detected, stop with an error (TRUE, default). |

## Details

Recognized column names are any that contain:

**dam** "dam", "mother", "mot", "mom", "mum", "mat"

**sire** "sire", "father", "fat", "dad", "pat"

sequoia requires the column order id - dam - sire; columns 2 and 3 are swapped if necessary.

## Examples

```
## Not run:
# To get the output pedigree into kinship2 compatible format:
PedP <- sequoia::PedPolish(SeqOUT$Pedigree, DropNonSNPd=FALSE,
                           FillParents = TRUE)
PedP$Sex <- with(PedP, ifelse(id %in% dam, "female",  "male"))
# default to 'male' to avoid warning: "More than 25% of the gender values are
#  'unknown'"

Ped.fix <- with(PedP, kinship2::fixParents(id=id, dadid=sire, momid=dam,
                                           sex=Sex))
```

```
Ped.k <- with(Ped.fix, kinship2::pedigree(id, dadid, momid, sex, missid=0))

## End(Not run)
```

---

PedStripFID *Back-transform IDs*

---

## Description

Reverse the joining of FID and IID in GenoConvert and LHConvert

## Usage

```
PedStripFID(Ped, FIDsep = "__")
```

## Arguments

| | |
|---|---|
| Ped | pedigree as returned by sequoia (e.g. SeqOUT$Pedigree). |
| FIDsep | characters inbetween FID and IID in composite-ID. |

## Details

Note that the family IDs are the ones provided, and not automatically updated. New, numeric ones can be obtained with FindFamilies.

## Value

A pedigree with 6 columns

| | |
|---|---|
| FID | family ID of focal individual (offspring). |
| id | within-family of focal individual |
| dam.FID | original family ID of assigned dam |
| dam | within-family of dam |
| sire.FID | original family ID of assigned sire |
| sire | within-family of sire |

## Ped_griffin *Example pedigree: griffins*

### Description

Example pedigree used in the ageprior vignette, with overlapping generations.

### Usage

```
data(Ped_griffin)
```

### Format

A data frame with 200 rows and 4 variables (id, dam, sire, birthyear)

### Author(s)

Jisca Huisman, <jisca.huisman@gmail.com>

### See Also

LH_griffin; SeqOUT_griffin for a sequoia run on simulated genotype data based on this pedigree; Ped_HSg5 for another pedigree, sequoia

## Ped_HSg5 *Example pedigree*

### Description

This is **Pedigree II** in the paper, with discrete generations and considerable inbreeding

### Usage

```
data(Ped_HSg5)
```

### Format

A data frame with 1000 rows and 3 variables (id, dam, sire)

### Author(s)

Jisca Huisman, <jisca.huisman@gmail.com>

### References

Huisman, J. (2017) Pedigree reconstruction from SNP data: Parentage assignment, sibship clustering, and beyond. Molecular Ecology Resources 17:1009–1024.

## See Also

LH_HSg5 SimGeno_example sequoia

---

PlotAgePrior *Plot Age Priors*

---

## Description

Visualise the age-difference based prior probability ratios as a heatmap.

## Usage

```
PlotAgePrior(AP = NULL, legend = TRUE)
```

## Arguments

| | |
|---|---|
| AP | matrix with age priors ($P(A\|R)/P(A)$) with age differences in rows and relationships in columns; by default M: maternal parent (mother), P: paternal parent (father), FS: full siblings, MS: maternal siblings (full + half), PS: paternal siblings. |
| legend | if TRUE, a new plotting window is started and layout is used to plot a legend next to the main plot. Set to FALSE if you want to add it as panel to an existing plot (e.g. with par(mfcol=c(2,2))). |

## Value

A heatmap.

## See Also

MakeAgePrior, SummarySeq.

## Examples

```
data(SeqOUT_griffin, package="sequoia")
PlotAgePrior(SeqOUT_griffin$AgePriors)
PlotAgePrior(SeqOUT_griffin$AgePriorExtra)
```

PlotPairLL *Plot Pair Log10-Likelihoods*

## Description

Colour-coded scatter plots of e.g. LLR(PO/U) against LLR(FS/U), for various relationship combinations.

## Usage

```
PlotPairLL(
  PairLL,
  combo = list(c("FS", "PO"), c("HS", "FS"), c("GP", "HS"), c("FA", "HS")),
  nrows = NULL,
  ncols = NULL,
  bgcol = TRUE,
  Tassign = 0.5,
  Tfilter = -2
)
```

## Arguments

| | |
|---|---|
| PairLL | dataframe, output from CalcPairLL. |
| combo | list with length-2 character vectors, specifying which likelihoods to plot against each other. Choose from 'PO', 'FS', 'HS', 'GP', 'FA', and 'HA'. The first one gets plotted on the x-axis, the second on the y-axis. Subsequent figures will be drawn row-wise. |
| nrows | number of rows in the figure layout. If NULL, set to ceiling(length(combo)/ncols). |
| ncols | number of columns in the figure layout. If both nrows and ncols are NULL, ncols is set to ceiling(sqrt(length(combo))), and nrows will be equal to ncols or one less. |
| bgcol | logical, colour the upper and lower triangle background of each figure to match the specified relationship combo. |
| Tassign | assignment threshold, shown as grey square in bottom-left corner and a band along the diagonal. |
| Tfilter | filter threshold, shown as dark grey square in bottom-left. |

## Details

The colour of each point is determined by columns focal (outer circle) and TopRel (inner filling) of PairLL.

Impossible relationships (LL > 0 in PairLL) are shown as -Inf on the axes, if any are present.

## See Also

CalcPairLL.

## Examples

```
data(SimGeno_example)
Pairs <- data.frame(ID1 = "a01005",
                    ID2 = c("a00013", "a00008", "a00011", "b00001",
                            "b01006", "b01007", "b01013", "b01014"),
                    focal = rep(c("PO", "HS"), each=4))
PLL <- CalcPairLL(Pairs, GenoM=SimGeno_example, Plot=FALSE)
PlotPairLL(PLL,
           combo = list(c("FS", "PO"), c("HS", "FS"), c("GP", "HS"),
                        c("FA", "HS"), c("HA", "FA"), c("FA", "GP")),
           nrows = 3)
```

---

PlotPedComp *Visualise PedCompare Output*

---

## Description

square Venn diagrams with PedCompare Counts.

## Usage

```
PlotPedComp(Counts, sameSize = FALSE)
```

## Arguments

| | |
|---|---|
| Counts | a 7x5x2 array with counts of matches and mismatches per category (genotyped vs dummy), as returned by PedCompare. |
| sameSize | logical, make all per-category Venn diagrams the same size TRUE, or make their size proportional to the counts (FALSE, the default). If TRUE, a warning is printed at the bottom. |

## See Also

PedCompare

## Examples

```
data(SeqOUT_griffin, FieldMums_griffin, package="sequoia")
PC.g <- PedCompare(Ped1 = cbind(FieldMums_griffin, sire=NA),
                   Ped2 = SeqOUT_griffin$Pedigree)
PlotPedComp(PC.g$Counts)
```

PlotRelPairs *Plot Pairwise Relationships*

## Description

Plot pairwise 1st and 2nd degree relationships between individuals, similar to Colony's dyad plot.

## Usage

```
PlotRelPairs(
  RelM = NULL,
  subset.x = NULL,
  subset.y = NULL,
  drop.U = TRUE,
  pch.symbols = FALSE,
  cex.axis = 0.7,
  mar = c(5, 5, 1, 8)
)
```

## Arguments

| | |
|---|---|
| RelM | square matrix with relationships between all pairs of individuals, as generated by GetRelM. Row and column names should be individual IDs. |
| subset.x | vector with IDs to show on the x-axis; the y-axis will include all siblings, parents and grandparents of these individuals. |
| subset.y | vector with IDs to show on the y-axis; the x-axis will include all siblings, offspring and grandoffspring of these individuals. Specify either subset.x or subset.y (or neither), not both. |
| drop.U | logical: omit individuals without relatives from the plot, and omit individuals without parents from the x-axis. Ignored if subset.x or subset.y is specified. |
| pch.symbols | logical: use different symbols for the different relationships (TRUE) or only colours in a heatmap-like fashion (FALSE). Question marks in the plot indicate that one or more of the symbols are not supported on your machine. |
| cex.axis | the magnification to be used for axis annotation. Decrease this value if R is dropping axis labels to prevent them from overlapping. |
| mar | A numerical vector of the form c(bottom, left, top, right) which gives the number of lines of margin to be specified on the four sides of the plot. |

## Details

Parents are shown above the diagonal (y-axis is parent of x-axis), siblings below the diagonal. If present, grandparents and full aunts/uncles are also shown above the diagonal. Individuals are sorted by dam ID and sire ID so that siblings are grouped together, and then by generation (getGenerations) so that later generations are closer to the origin.

If RelM is based on a dataframe with pairs rather than a pedigree, parents and grandparents are similarly only displayed above the diagonal, but the order of individuals is arbitrary and the ID on the x-axis is as likely to be the grandparent of the one on the y-axis as vice versa. Second degree relatives of unknown classification ('2nd', may be HS, GP or FA) are only shown below the diagonal. The switch between pedigree-based versus pairs-based is made on whether parent-offspring pairs are coded as 'M','P', 'MP', 'O' (unidirectional, from pedigree) or as 'PO' (bidirectional, from pairs).

Note that half-avuncular and (double) full cousin pairs are ignored.

## Value

The subsetted, rearranged RelM is returned invisible.

The numbers of unique pairs of each relationship type are given in the figure legend. The number of 'self' pairs refers to the number of individuals on the x-axis, not all of whom may occur on the y-axis when drop.U=TRUE or a subset is specified.

## See Also

GetRelM; SummarySeq for individual-wise graphical pedigree summaries.

## Examples

```
data(Ped_griffin, package="sequoia")
Rel.griffin <- GetRelM(Ped_griffin, patmat=TRUE, GenBack=2)
PlotRelPairs(Rel.griffin)


PlotRelPairs(Rel.griffin, pch.symbols = TRUE)
# plot with unicode symbols not supported on all platforms


# parents & grandparents of 2008 cohort:
PlotRelPairs(Rel.griffin,
             subset.x = Ped_griffin$id[Ped_griffin$birthyear ==2008])
# offspring & grand-offspring of 2002 cohort:
PlotRelPairs(Rel.griffin,
             subset.y = Ped_griffin$id[Ped_griffin$birthyear ==2002])
```

---

# SeqOUT_griffin *Example sequoia output (griffins)*

---

## Description

Example output of a sequoia run including sibship clustering, based on the griffin pedigree.

## Usage

```
data(SeqOUT_griffin)
```

## Format

a list, see sequoia

## Author(s)

Jisca Huisman, <jisca.huisman@gmail.com>

## See Also

Ped_griffin,sequoia

## Examples

```
## Not run:
GenoS <- SimGeno(Ped.griffin, nSnp=400, ParMis=0.4)
griffin.sex <- sapply(as.character(Ped_griffin$id), function(x)
                      substr(x, start=nchar(x), stop=nchar(x)))
LH.griffin <- data.frame(ID = Ped_griffin$id,
                         Sex = ifelse(griffin.sex=="F", 1, 2),
                         BirthYear = Ped_griffin$birthyear)
SeqOUT.GX <- sequoia(GenoS, LH.griffin,
                     Module = "ped",
                     args.AP = list(Smooth = FALSE))

## End(Not run)
```

---

sequoia *Pedigree Reconstruction*

---

## Description

Perform pedigree reconstruction based on SNP data, including parentage assignment and sibship clustering.

## Usage

```
sequoia(
  GenoM = NULL,
  LifeHistData = NULL,
  SeqList = NULL,
  Module = "ped",
  MaxSibIter = 42,
  Err = 1e-04,
  ErrFlavour = "version2.0",
  MaxMismatch = NA,
  Tfilter = -2,
  Tassign = 0.5,
  MaxSibshipSize = 100,
```

```
  DummyPrefix = c("F", "M"),
  Complex = "full",
  Herm = "no",
  UseAge = "yes",
  args.AP = list(Flatten = NULL, Smooth = TRUE),
  FindMaybeRel = FALSE,
  CalcLLR = TRUE,
  quiet = FALSE,
  Plot = NULL
)
```

## Arguments

| | |
|---|---|
| GenoM | numeric matrix with genotype data: One row per individual, and one column per SNP, coded as 0, 1, 2 or -9 (missing). See also GenoConvert. |
| LifeHistData | dataframe with 3 columns (optionally 5):<br>**ID** max. 30 characters long<br>**Sex** 1 = female, 2 = male, 3 = unknown, 4 = hermaphrodite, other numbers or NA = unknown<br>**BirthYear** birth or hatching year, integer, with missing values as NA or any negative value.<br>**BY.min** minimum birth year, only used if BirthYear is missing<br>**BY.max** maximum birth year, only used if BirthYear is missing<br>If the species has multiple generations per year, use an integer coding such that the candidate parents' 'Birth year' is at least one smaller than their putative offspring's. Column names are ignored, so ensure column order is ID - sex - birth year (- BY.min - BY.max). Individuals do not need to be in the same order as in 'GenoM', nor do all genotyped individuals need to be included. |
| SeqList | list with output from a previous run, to be re-used in the current run. Used are elements 'PedigreePar', 'LifeHist', 'AgePriors', 'Specs', and 'ErrM', and these override the corresponding input parameters. Not all of these elements need to be present, and all other elements are ignored. If SeqList$Specs is provided, all input parameters with the same name as its items are ignored, except Module/MaxSibIter. |
| Module | one of<br>**pre** Only input check, return SeqList$Specs<br>**dup** Also check for duplicate genotypes<br>**par** Also perform parentage assignment (genotyped parents to genotyped offspring)<br>**ped** (Also) perform full pedigree reconstruction, including sibship clustering and grandparent assignment. By far the most time consuming, and may take several hours for large datasets.<br>NOTE: *Until 'MaxSibIter' is fully deprecated: if 'MaxSibIter' differs from the default (42), and 'Module' equals the default ('ped'), MaxSibIter overrides 'Module'.* |

| | |
|---|---|
| MaxSibIter | **[will be deprecated]** number of iterations of sibship clustering, including assignment of grandparents to sibships and avuncular relationships between sibships. Clustering continues until convergence or until MaxSibIter is reached. Set to 0 for parentage assignment only. |
| Err | estimated genotyping error rate, as a single number or 3x3 matrix. Details below. The error rate is presumed constant across SNPs, and missingness is presumed random with respect to actual genotype. |
| ErrFlavour | function that takes Err (single number) as input, and returns a 3x3 matrix of observed (columns) conditional on actual (rows) genotypes, or choose from inbuilt options 'version2.0', 'version1.3', or 'version1.1', referring to the sequoia version in which they were the default. Ignored if Err is a matrix. See ErrToM. |
| MaxMismatch | **DEPRECATED AND IGNORED**. Now calculated automatically using CalcMaxMismatch. |
| Tfilter | threshold log10-likelihood ratio (LLR) between a proposed relationship versus unrelated, to select candidate relatives. Typically a negative value, related to the fact that unconditional likelihoods are calculated during the filtering steps. More negative values may decrease non-assignment, but will increase computational time. |
| Tassign | minimum LLR required for acceptance of proposed relationship, relative to next most likely relationship. Higher values result in more conservative assignments. Must be zero or positive. |
| MaxSibshipSize | maximum number of offspring for a single individual (a generous safety margin is advised). |
| DummyPrefix | character vector of length 2 with prefixes for dummy dams (mothers) and sires (fathers); maximum 20 characters each. Length 3 vector in case of hermaphrodites (or default prefix 'H'). |
| Complex | Breeding system complexity. Either "full" (default), "simp" (simplified, no explicit consideration of inbred relationships), "mono" (monogamous). |
| Herm | Hermaphrodites, either "no", "A" (distinguish between dam and sire role, default if at least 1 individual with sex=4), or "B" (no distinction between dam and sire role). Both of the latter deal with selfing. |
| UseAge | either "yes" (default), "no", or "extra" (additional rounds with extra reliance on ageprior, may boost assignments but increased risk of erroneous assignments); used during full reconstruction only. |
| args.AP | list with arguments to be passed on to MakeAgePrior. |
| FindMaybeRel | **DEPRECATED AND IGNORED**, advised to run GetMaybeRel separately. TRUE/FALSE to identify pairs of non-assigned likely relatives after pedigree reconstruction. Can be time-consuming in large datasets. |
| CalcLLR | TRUE/FALSE; calculate log-likelihood ratios for all assigned parents (genotyped + dummy; parent vs. otherwise related). Time-consuming in large datasets. Can be done separately with CalcOHLLR. |
| quiet | suppress messages: TRUE/FALSE/"verbose". |
| Plot | display plots from SnpStats, MakeAgePrior, and SummarySeq. Defaults (NULL) to TRUE when quiet=FALSE or "verbose", and FALSE when quiet=TRUE. If you get error 'figure margins too large', enlarge the plotting area (drag with mouse). Error 'invalid graphics state' can be dealt with by clearing the plotting area with dev.off(). |

## Details

For each pair of candidate relatives, the likelihoods are calculated of them being parent-offspring (PO), full siblings (FS), half siblings (HS), grandparent-grandoffspring (GG), full avuncular (niece/nephew - aunt/uncle; FA), half avuncular/great-grandparental/cousins (HA), or unrelated (U). Assignments are made if the likelihood ratio (LLR) between the focal relationship and the most likely alternative exceed the threshold Tassign.

Dummy parents of sibships are denoted by F0001, F0002, ... (mothers) and M0001, M0002, ... (fathers), are appended to the bottom of the pedigree, and may have been assigned real or dummy parents themselves (i.e. sibship-grandparents). A dummy parent is not assigned to singletons.

The genotyping error rate 'Err' is by default at locus level, not allele level: the probability to observe true homozygote *aa* as heterozygote *Aa* is $\approx E$, and as alternate homozygote *AA* $(E/2)^2$; the probability to observe a true heterozygote as *aa* = the probability to observe it as *AA* $= E/2$. This error structure can be fully customised by providing a 3x3 matrix of observed genotype (columns) conditional on actual genotype (rows) instead.

Full explanation of the various options and interpretation of the output is provided in the vignette.

## Value

A list with some or all of the following components:

| | |
|---|---|
| `AgePriors` | Matrix with age-difference based probability ratios for each relationship, used for full pedigree reconstruction; see `MakeAgePrior` for details. When running only parentage assignment (`Module="par"`) the returned AgePriors has been updated to incorporate the information of the assigned parents, and is ready for use during full pedigree reconstruction. |
| `DummyIDs` | Dataframe with pedigree for dummy individuals, as well as their sex, estimated birth year (point estimate, upper and lower bound of 95% confidence interval; see also `CalcBYprobs`), number of offspring, and offspring IDs. From version 2.1 onwards, this includes dummy offspring. |
| `DupGenotype` | Dataframe, duplicated genotypes (with different IDs, duplicate IDs are not allowed). The specified number of maximum mismatches is used here too. Note that this dataframe may include pairs of closely related individuals, and monozygotic twins. |
| `DupLifeHistID` | Dataframe, row numbers of duplicated IDs in life history dataframe. For convenience only, but may signal a problem. The first entry is used. |
| `ErrM` | Error matrix; probability of observed genotype (columns) conditional on actual genotype (rows) |
| `ExcludedInd` | Individuals in GenoM which were excluded because of a too low genotyping success rate (<50%). |
| `ExcludedSNPs` | Column numbers of SNPs in GenoM which were excluded because of a too low genotyping success rate (<10%). |
| `LifeHist` | Provided dataframe with sex and birth year data. |
| `LifeHistPar` | LifeHist with additional columns 'Sexx' (inferred Sex when assigned as part of parent-pair), 'BY.est' (mode of birth year probability distribution), 'BY.lo' (lower limit of 95% highest density region), 'BY.hi' (higher limit), inferred after |

parentage assignment. 'BY.est' is NA when the probability distribution is flat between 'BY.lo' and 'BY.hi'.

`LifeHistSib` as LifeHistPar, but estimated after full pedigree reconstruction

`MaybeParent` Dataframe with pairs of individuals who are more likely parent-offspring than unrelated, but which could not be phased due to unknown age difference or sex, or for whom LLR did not pass Tassign.

`MaybeRel` Dataframe with pairs of individuals who are more likely to be first or second degree relatives than unrelated, but which could not be assigned.

`MaybeTrio` Dataframe with non-assigned parent-parent-offspring trios (both parents are of unknown sex), with similar columns as the pedigree

`NoLH` Vector, IDs in genotype data for which no life history data is provided.

`Pedigree` Dataframe with assigned genotyped and dummy parents from Sibship step; entries for dummy individuals are added at the bottom.

`PedigreePar` Dataframe with assigned parents from Parentage step.

`Specs` Named vector with parameter values.

`TotLikParents` Numeric vector, Total likelihood of the genotype data at initiation and after each iteration during Parentage.

`TotLikSib` Numeric vector, Total likelihood of the genotype data at initiation and after each iteration during Sibship clustering.

`AgePriorExtra` As AgePriors, but including columns for grandparents and avuncular pairs. NOT updated after parentage assignment, but returned as used during the run.

`DummyClones` Hermaphrodites only: female-male dummy ID pairs that refer to the same non-genotyped individual

List elements PedigreePar and Pedigree both have the following columns:

`id` Individual ID

`dam` Assigned mother, or NA

`sire` Assigned father, or NA

`LLRdam` Log10-Likelihood Ratio (LLR) of this female being the mother, versus the next most likely relationship between the focal individual and this female. See Details below for relationships considered, and see `CalcPairLL` for underlying likelihood values and further details)

`LLRsire` idem, for male parent

`LLRpair` LLR for the parental pair, versus the next most likely configuration between the three individuals (with one or neither parent assigned)

`OHdam` Number of loci at which the offspring and mother are opposite homozygotes

`OHsire` idem, for father

`MEpair` Number of Mendelian errors between the offspring and the parent pair, includes OH as well as e.g. parents being opposing homozygotes, but the offspring not being a heterozygote. The offspring being OH with both parents is counted as 2 errors.

## Disclaimer

While every effort has been made to ensure that sequoia provides what it claims to do, there is absolutely no guarantee that the results provided are correct. Use of sequoia is entirely at your own risk.

## Author(s)

Jisca Huisman, <jisca.huisman@gmail.com>

## References

Huisman, J. (2017) Pedigree reconstruction from SNP data: Parentage assignment, sibship clustering, and beyond. Molecular Ecology Resources 17:1009–1024.

## See Also

- GenoConvert to read in various data formats,
- CheckGeno, SnpStats to calculate missingness and allele frequencies,
- SimGeno to simulate SNP data from a pedigree
- MakeAgePrior to estimate effect of age on relationships,
- GetMaybeRel to find pairs of potential relatives,
- SummarySeq and PlotAgePrior to visualise results,
- GetRelM to turn a pedigree into pairwise relationships,
- CalcOHLLR to calculate Mendelian errors and LLR for any pedigree,
- CalcPairLL for likelihoods of various relationships between specific pairs,
- CalcBYprobs to estimate birth years,
- PedCompare and ComparePairs to compare to two pedigrees,
- EstConf to estimate assignment errors,
- writeSeq to save results,
- vignette("sequoia") for detailed manual & FAQ.

## Examples

```
# ===  EXAMPLE 1: simulated data  ===
data(SimGeno_example, LH_HSg5, package="sequoia")
head(SimGeno_example[,1:10])
head(LH_HSg5)
# parentage assignment:
SeqOUT <- sequoia(GenoM = SimGeno_example, Err = 0.005,
                  LifeHistData = LH_HSg5, Module="par", Plot=TRUE)
names(SeqOUT)
SeqOUT$PedigreePar[34:42, ]

# compare to true (or old) pedigree:
PC <- PedCompare(Ped_HSg5, SeqOUT$PedigreePar)
PC$Counts["GG",,]
```

```
# parentage assignment + full pedigree reconstruction:
SeqOUT2 <- sequoia(GenoM = SimGeno_example, Err = 0.005,
                   LifeHistData = LH_HSg5, Module="ped", quiet="verbose")
SeqOUT2$Pedigree[34:42, ]

PC2 <- PedCompare(Ped_HSg5, SeqOUT2$Pedigree)
PC2$Counts["GT",,]
PC2$Counts[,,"dam"]

# different kind of pedigree comparison:
ComparePairs(Ped1=Ped_HSg5, Ped2=SeqOUT$PedigreePar, patmat=TRUE)

# results overview:
SummarySeq(SeqOUT2)

# important to run with approx. correct genotyping error rate:
SeqOUT2.b <- sequoia(GenoM = SimGeno_example, #  Err = 1e-4 by default
                   LifeHistData = LH_HSg5, Module="ped", Plot=FALSE)
PC2.b <- PedCompare(Ped_HSg5, SeqOUT2.b$Pedigree)
PC2.b$Counts["GT",,]

## Not run:
# ===  EXAMPLE 2: real data  ===
# ideally, select 400-700 SNPs: high MAF & low LD
# save in 0/1/2/NA format (PLINK's --recodeA)
GenoM <- GenoConvert(InFile = "inputfile_for_sequoia.raw",
                     InFormat = "raw")  # can also do Colony format
SNPSTATS <- SnpStats(GenoM)
# perhaps after some data-cleaning:
write.table(GenoM, file="MyGenoData.txt", row.names=T, col.names=F)

# later:
GenoM <- as.matrix(read.table("MyGenoData.txt", row.names=1, header=F))
LHdata <- read.table("LifeHistoryData.txt", header=T) # ID-Sex-birthyear
SeqOUT <- sequoia(GenoM, LHdata, Err=0.005)
SummarySeq(SeqOUT)

writeSeq(SeqOUT, folder="sequoia_output")  # several text files

# runtime:
SeqOUT$Specs$TimeEnd - SeqOUT$Specs$TimeStart

## End(Not run)
```

---

SimGeno *Simulate Genotypes*

---

## Description

Simulate SNP genotype data from a pedigree, with optional missingess and errors.

## Usage

```
SimGeno(
  Pedigree,
  nSnp = 400,
  ParMis = 0.4,
  MAF = 0.3,
  CallRate = 0.99,
  SnpError = 5e-04,
  ErrorFM = "version2.0",
  ReturnStats = FALSE,
  OutFile = NA,
  Inherit = "autosomal",
  InheritFile = NA,
  quiet = FALSE
)
```

## Arguments

| | |
|---|---|
| Pedigree | dataframe, pedigree with the first three columns being id - dam - sire. Column names are ignored, as are additional columns, with the exception of a 'Sex' column when Inherit is not 'autosomal'. |
| nSnp | number of SNPs to simulate. |
| ParMis | single number or vector length two with proportion of parents with fully missing genotype. Ignored if CallRate is a named vector. |
| MAF | minimum minor allele frequency, and allele frequencies will be sampled uniformly between this minimum and 0.5, OR a vector with minor allele frequency at each locus. In both cases, this is the MAF among pedigree founders, the MAF in the sample will deviate due to drift. |
| CallRate | either a single number for the mean call rate (genotyping success), OR a vector with the call rate at each SNP, OR a named vector with the call rate for each individual. In the third case, ParMis is ignored, and individuals in the pedigree (as id or parent) not included in this vector are presumed non-genotyped. |
| SnpError | mean per-locus genotyping error rate across SNPs, and a beta-distribution will be used to simulate the number of missing cases per SNP, OR a vector with the genotyping error for each SNP. |
| ErrorFM | function taking the error rate (scalar) as argument and returning a 3x3 matrix with probabilities that actual genotype i (rows) is observed as genotype j (columns). Inbuilt ones are as used in sequoia 'version2.0', 'version1.3', or 'version1.1'. See details. |
| ReturnStats | in addition to the genotype matrix, return the input parameters and mean & quantiles of MAF, error rate and call rates. |
| OutFile | file name for simulated genotypes. If NA (default), return results within R. |

| | |
|---|---|
| Inherit | inheritance pattern, scalar or vector of length nSnp, Defaults to 'autosomal'. An excel file included in the package has inheritance patterns for the X and Y chromosome and mtDNA, and allows custom inheritance patterns. Note that these are experimental, and NOT currently supported by the pedigree reconstruction with sequoia ! |
| InheritFile | file name of file with inheritance patterns, with extension csv, txt, xls or xlsx (the latter two require library xlsx). |
| quiet | suppress messages. |

## Details

Please ensure the pedigree is a valid pedigree, for example by first running PedPolish. For founders, i.e. individuals with no known parents, genotypes are drawn according to the provided MAF and assuming Hardy-Weinberg equilibrium. Offspring genotypes are generated following Mendelian inheritance, assuming all loci are completely independent. Individuals with one known parent are allowed: at each locus, one allele is inherited from the known parent, and the other drawn from the genepool according to the provided MAF.

Genotyping errors are generated following a user-definable 3x3 matrix with probabilities that actual genotype $i$ (rows) is observed as genotype $j$ (columns). This is specified as ErrorFM, which is a function of SnpError. By default (ErrorFM = "version2.0"), SnpError is interpreted as a locus-level error rate (rather than allele-level), and equals the probability that a homozygote is observed as heterozygote, and the probability that a heterozygote is observed as either homozygote (i.e., the probability that it is observed as AA = probability that observed as aa = SnpError/2). The probability that one homozygote is observed as the other is $(\text{SnpError}/2)^2$.

Note that this differs from versions up to 1.1.1, where a proportion of SnpError*3/2 of genotypes were replaced with random genotypes. This corresponds to ErrorFM = "Version111".

Error rates differ between SNPs, but the same error pattern is used across all SNPs, even when inheritance patterns vary. When two or more different error patterns are required, SimGeno should be run on the different SNP subsets separately, and results combined.

Variation in call rates is assumed to follow a highly skewed (beta) distribution, with many samples having call rates close to 1, and a narrowing tail of lower call rates. The first shape parameter defaults to 1 (but see MkGenoErrors), and the second shape parameter is defined via the mean as CallRate. For 99.9% of SNPs to have a call rate of 0.8 (0.9; 0.95) or higher, use a mean call rate of 0.969 (0.985; 0.993).

Variation in call rate between samples can be specified by providing a named vector to CallRate, which supersedes PropLQ in versions up to 1.1.1. Otherwise, variation in call rate and error rate between samples occurs only as side-effect of the random nature of which individuals are hit by per-SNP errors and drop-outs. Finer control is possible by first generating an error-free genotype matrix, and then calling MkGenoErrors directly on subsets of the matrix.

## Value

If ReturnStats=FALSE (the default), a matrix with genotype data in sequoia's input format, encoded as 0/1/2/-9.

If ReturnStats=TRUE, a named list with three elements: list 'ParamsIN', matrix 'SGeno', and list 'StatsOUT':

| | |
|---|---|
| AF | Frequency in 'observed' genotypes of '1' allele |
| AF.act | Allele frequency in 'actual' (without genotyping errors & missingness) |
| SnpError | Error rate per SNP (actual /= observed AND observed /= missing) |
| SnpCallRate | Non-missing per SNP |
| IndivError | Error rate per individual |
| IndivCallRate | Non-missing per individual |

## Disclaimer

This simulation is highly simplistic and assumes that all SNPs segregate completely independently, that the SNPs are in Hardy-Weinberg equilibrium in the pedigree founders. It assumes that genotyping errors are not due to heritable mutations of the SNPs, and that missingness is random and not e.g. due to heritable mutations of SNP flanking regions. Results based on this simulated data will provide an minimum estimate of the number of SNPs required, and an optimistic estimate of pedigree reconstruction performance.

## Author(s)

Jisca Huisman, <jisca.huisman@gmail.com>

## See Also

The wrapper EstConf for repeated simulation and pedigree reconstruction; MkGenoErrors for fine control over the distribution of genotyping errors in simulated data.

## Examples

```
data(Ped_HSg5)
GenoM <- SimGeno(Pedigree = Ped_HSg5, nSnp = 100, ParMis = c(0.2, 0.7))

## Not run:
# Alternative genotyping error model
EFM <- function(E) {   # Whalen, Gorjanc & Hickey 2018
 matrix(c(1-E*3/4, E/4, E/4,
          E/4, 1/2-E/4, 1/2-E/4, E/4,
          E/4, E/4, 1-E*3/4),
          3,3, byrow=TRUE)  }
EFM(0.01)
GenoM <- SimGeno(Pedigree = Ped_HSg5, nSnp = 100, ParMis = 0.2,
 SnpError = 5e-3, ErrorFM = EFM)

# combination of high & low quality SNPs
Geno.HQ <- SimGeno(Ped_HSg5, nSnp=50, MAF=0.3, CallRate=runif(50, 0.7, 1))
Geno.LQ <- SimGeno(Ped_HSg5, nSnp=20, MAF=0.1, CallRate=runif(20, 0.1, 5))
Geno.HQLQ <- merge(Geno.HQ, Geno.LQ, by="row.names")

## End(Not run)
```

## SimGeno_example *Example genotype file*

### Description

Simulated genotype data for cohorts 1+2 in Pedigree Ped_HSg5

### Usage

```
data(SimGeno_example)
```

### Format

A genotype matrix with 214 rows (ids) and 200 columns (SNPs). Each SNP is coded as 0/1/2 copies of the reference allele, with -9 for missing values. Ids are stored as rownames.

### Author(s)

Jisca Huisman, <jisca.huisman@gmail.com>

### See Also

Ped_HSg5, SimGeno

## SnpStats *SNP Summary Statistics*

### Description

Estimate allele frequency (AF), missingness and Mendelian errors per SNP.

### Usage

```
SnpStats(GenoM, Pedigree = NULL, ErrFlavour = "version2.0", Plot = TRUE)
```

### Arguments

| | |
|---|---|
| GenoM | genotype matrix, in sequoia's format: 1 column per SNP, 1 row per individual, genotypes coded as 0/1/2/-9, and rownames giving individual IDs. |
| Pedigree | dataframe with 3 columns: ID - parent1 - parent2. Additional columns and non-genotyped individuals are ignored. Used to estimate the error rate. |
| ErrFlavour | function that takes the genotyping error rate Err as input, and returns a 3x3 matrix of observed (columns) conditional on actual (rows) genotypes, or choose from inbuilt ones as used in sequoia 'version2.0', 'version1.3', or 'version1.1'. See ErrToM. |
| Plot | show histograms of the results? |

## Details

Calculation of these summary statistics can be done in PLINK, and SNPs with low minor allele frequency or high missingness should be filtered out prior to pedigree reconstruction. This function is provided as an aid to inspect the relationship between AF, missingness and genotyping error to find a suitable combination of SNP filtering thresholds to use.

For pedigree reconstruction, SNPs with zero or one copies of the alternate allele in the dataset (MAF $\leq 1/2N$) are considered fixed, and excluded.

## Value

A matrix with a number of rows equal to the number of SNPs (=number of columns of GenoM), and when no Pedigree is provided 2 columns:

| | |
|---|---|
| `AF` | Allele frequency of the 'second allele' (the one for which the homozygote is coded 2) |
| `Mis` | Proportion of missing calls |

When a Pedigree is provided, there are 7 additional columns:

| | |
|---|---|
| `n.dam, n.sire, n.pair` | Number of dams, sires, parent-pairs succesfully genotyped for the SNP |
| `OHdam, OHsire` | Count of number of opposing homozygous cases |
| `MEpair` | Count of Mendelian errors, includes opposing homozygous cases |
| `Err.hat` | Error rate, as estimated from the joined offspring-parent (-parent) genotypes and the presumed error structure (`ErrFlavour`) |

## Estimated genotyping error

The error rate is estimated from the number of opposing homozygous cases (OH, parent is AA and offspring is aa) Mendelian errors (ME, e.g. parents AA and aa, but offspring not Aa) in parent-parent-offspring trios, and OH cases for offspring with a single genotyped parent.

The estimated error rates will not be as accurate as from duplicate samples. A single error in an individual with many offspring will be counted as many times, potentially resulting in non-sensical values of 'Err.hat' close to 1. On the other hand, errors in individuals without parents or offspring will not be counted at all. Moreover, a high error rate may interfere with pedigree reconstruction, and successful assignment will be biased towards parents with lower error count. Nonetheless, it may provide a ballpark estimate for the average error rate, which can be useful for subsequent (rerun of) pedigree reconstruction.

## See Also

`GenoConvert` to convert from various data formats; `CheckGeno` to check the data is in valid format for sequoia and exclude monomorphic SNPs etc., `CalcOHLLR` to calculate OH & ME per individual.

## Examples

```
data(Ped_HSg5)
Genotypes <- SimGeno(Ped_HSg5, nSnp=400, CallRate = runif(400, 0.2, 0.8),
  SnpError = 0.05)
SNPstats <- SnpStats(Genotypes, Pedigree=Ped_HSg5)
```

---

SummarySeq *Summarise Sequoia Output or Pedigree*

---

## Description

Number of assigned parents and grandparents and sibship sizes, split by genotyped, dummy, and 'observed'.

## Usage

```
SummarySeq(
  SeqList = NULL,
  Pedigree = NULL,
  DumPrefix = c("F0", "M0"),
  SNPd = NULL,
  Plot = TRUE,
  Panels = "all"
)
```

## Arguments

| | |
|---|---|
| SeqList | the list returned by sequoia. Only elements 'Pedigree' or 'PedigreePar' and 'AgePriors' are used. |
| Pedigree | dataframe, pedigree with the first three columns being id - dam - sire. Column names are ignored, as are additional columns. |
| DumPrefix | character vector of length 2 with prefixes for dummy dams (mothers) and sires (fathers). Will be read from SeqList's 'Specs' if provided. Used to distinguish between dummies and non-dummies. Length 3 in case of hermaphrodites. |
| SNPd | character vector with ids of SNP genotyped individuals. Only when Pedigree is provided instead of SeqList, then used to distinguish between genetically assigned parents and 'observed' parents (e.g. observed in the field, or assigned previously using microsatellites). If SeqList's 'PedigreePar' is provided, all ids in that dataframe will be presumed genotyped. |
| Plot | show barplots and histograms of the results, as well as of the parental LLRs, Mendelian errors, and agepriors, if present. |
| Panels | character vector with panel(s) to plot. Choose from 'all', 'G.parents' (parents of genotyped individuals), 'D.parents' (parents of dummy individuals), 'sibships' (distribution of sibship sizes), 'LLR' (log10-likelihood ratio parent/otherwise related), 'OH' (count of opposite homozygote SNPs). |

## Value

A list with the following elements:

| | |
|---|---|
| PedSummary | a 2-column matrix with basic summary statistics, similar to what used to be returned by **Pedantics**' pedStatSummary (now archived on CRAN). First column refers to the complete pedigree, second column to SNP-genotyped individuals only. Maternal siblings sharing a dummy parent are counted in the 2nd column if both sibs are genotyped, but not if one of the sibs is a dummy individual. |
| ParentCount | a 2x3x2x4 array with the number of assigned parents, split by D1: genotyped vs dummy individuals; D2: female, male and unknown-sex individuals; D3: dams vs sires; D4: genotyped, dummy, observed vs no parent |
| GPCount | a 2x4x4 array with the number of assigned grandparents, split by D1: genotyped vs dummy individuals; D2 Maternal grandmother (MGM), maternal grandfather (MGF), paternal grandmother (PGM), paternal grandfather (PGF); D3: genotyped, dummy, observed vs no grandparent |
| SibSize | a list with as first element a table of maternal sibship sizes, and as second element a table of paternal sibship sizes. Each table is a matrix with a number of rows equal to the maximum sibship size, and 3 columns, splitting by the type of parent: genotyped, dummy, or observed. |

## See Also

sequoia for pedigree reconstruction and links to other functions.

## Examples

```
data(SimGeno_example, LH_HSg5, package="sequoia")
SeqOUT <- sequoia(GenoM = SimGeno_example,
                  LifeHistData = LH_HSg5, Module="ped")
Ped_example <- SeqOUT[["Pedigree"]]
Ped_example$dam[1:20] <- paste0("Mum", 1:20)  # some fake field mums
PedSum1 <- SummarySeq(SeqOUT, Pedigree=Ped_example, Panels="G.parents")
summary(PedSum1)
PedSum1$PedSummary
```

---

# tryCatch.W.E *tryCatch both warnings (with value) and errors*

---

## Description

Catch *and* save both errors and warnings, and in the case of a warning, also keep the computed result.

### Usage

```
tryCatch.W.E(expr)
```

### Arguments

| | |
|---|---|
| expr | an R expression to evaluate |

### Value

a list with 'value' and 'warning', where 'value' may be an error caught.

### Author(s)

Martin Maechler; Copyright (C) 2010-2012 The R Core Team

---

## writeColumns *Write Data to a File Column-wise*

---

### Description

Write data.frame or matrix to a text file, using white space padding to keep columns aligned as in `print`.

### Usage

```
writeColumns(x, file = "", row.names = TRUE, col.names = TRUE)
```

### Arguments

| | |
|---|---|
| x | the object to be written, preferably a matrix or data frame. If not, it is attempted to coerce x to a matrix. |
| file | a character string naming a file. |
| row.names | a logical value indicating whether the row names of x are to be written along with x. |
| col.names | a logical value indicating whether the column names of x are to be written along with x. |

writeSeq *Write Sequoia Output to File*

## Description

The various list elements returned by `sequoia` are each written to text files in the specified folder, or to separate sheets in a single excel file (requires library **xlsx**).

## Usage

```
writeSeq(
  SeqList,
  GenoM = NULL,
  MaybeRel = NULL,
  PedComp = NULL,
  OutFormat = "txt",
  folder = "Sequoia-OUT",
  file = "Sequoia-OUT.xlsx",
  ForVersion = 2,
  quiet = FALSE
)
```

## Arguments

| | |
|---|---|
| `SeqList` | list returned by `sequoia`, to be written out. |
| `GenoM` | matrix with genetic data (optional). Ignored if OutFormat='xls', as the resulting file could become too large for excel. |
| `MaybeRel` | list with results from `GetMaybeRel` (optional). |
| `PedComp` | list with results from `PedCompare` (optional). `SeqList$DummyIDs` is combined with `PedComp$DummyMatch` if both are provided. |
| `OutFormat` | 'xls' or 'txt'. |
| `folder` | the directory where the text files will be written; will be created if it does not already exists. Relative to the current working directory, or NULL for current working directory. Ignored if `OutFormat='xls'`. |
| `file` | the name of the excel file to write to, ignored if `OutFormat='txt'`. |
| `ForVersion` | choose '1' for back-compatibility with stand-alone sequoia versions 1.x |
| `quiet` | suppress messages. |

## Details

The text files can be used as input for the stand-alone Fortran version of sequoia, e.g. when the genotype data is too large for R. See `vignette('sequoia')` for further details.

## See Also

`writeColumns` to write to a text file, using white space padding to keep columns aligned.

## Examples

```
## Not run:
writeSeq(SeqList, OutFormat="xls", file="MyFile.xlsx")

# add additional sheets to the excel file:
library(xlsx)
write.xlsx(MyData, file = "MyFile.xlsx", sheetName="ExtraData",
      col.names=TRUE, row.names=FALSE, append=TRUE, showNA=FALSE)

## End(Not run)
```

# Index