Package: sequoia (via r-universe)

October 5, 2026

Type Package

Title Pedigree Inference from SNPs

Version 3.2.0

Date 2025-12-30

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, cli

RoxygenNote 7.3.3

Suggests openxlsx, knitr, rmarkdown, bookdown, kinship2, R.rsp, hexbin, data.table, vcfR, adegenet

VignetteBuilder knitr, R.rsp

NeedsCompilation yes

Depends R (>= 3.5.0)

SystemRequirements Fortran95

Repository https://cran.r-universe.dev

Date/Publication 2026-01-09 16:40:02 UTC

RemoteUrl https://github.com/cran/sequoia

RemoteRef HEAD

RemoteSha 045c4fde57c00231bd6c2b992f379c9131e453cf

Contents

CalcBYprobs . . . 3
CalcMaxMismatch . . . 4
CalcOHLLR . . . 6
CalcPairLL . . . 10
CalcParentProbs . . . 16
CalcRped . . . 18
CheckGeno . . . 19
ComparePairs . . . 20
Conf_griffin . . . 24
CountOH . . . 25
DyadCompare . . . 26
Err_RADseq . . . 27
ErrToM . . . 28
EstConf . . . 31
FieldMums_griffin . . . 35
FindFamilies . . . 36
Geno_griffin . . . 37
Geno_HSg5 . . . 37
GenoConvert . . . 38
GetAncestors . . . 41
getAssignCat . . . 42
GetDescendants . . . 43
getGenerations . . . 44
GetLLRAge . . . 45
GetMaybeRel . . . 46
GetRelM . . . 50
Inherit_patterns . . . 52
LH_griffin . . . 53
LH_HSg5 . . . 54
LHConvert . . . 54
LLtoProb . . . 56
MakeAgePrior . . . 57
MaybeRel_griffin . . . 61
MkGenoErrors . . . 61
Ped_griffin . . . 63
Ped_HSg5 . . . 63
PedCompare . . . 64
PedPolish . . . 68
PedStripFID . . . 70
PlotAgePrior . . . 71
PlotPairLL . . . 71
PlotPedComp . . . 73
PlotPropAssigned . . . 73
PlotRelPairs . . . 74
PlotSeqSum . . . 76
SeqOUT_griffin . . . 77

SeqOUT_HSg5 77
sequoia 78
SimGeno 86
SimGeno_example 89
SnpStats 90
SummarySeq 91
writeColumns 93
writeSeq 94

Index **96**

`CalcBYprobs` *Birth year probabilities*

Description

Estimate the probability that an individual with unknown birth year is born in year `y`, based on `BirthYears` or `BY.min` and/or `BY.max` of its parents, offspring, and siblings, combined with `AgePrior` (the age distribution of other parent-offspring pairs), and/or `Year.last` of its parents.

Usage

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

Arguments

`Pedigree`	dataframe with columns id-dam-sire.
`LifeHistData`	data.frame with up to 6 columns: **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 number. **BY.min** minimum birth year, only used if BirthYear is missing **BY.max** maximum birth year, only used if BirthYear is missing **Year.last** Last year in which individual could have had offspring. Can e.g. in mammals be the year before death for females, and year after death for males. "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, and only integers are used. Individuals do not need to be in the same order as in 'GenoM', nor do all genotyped individuals 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.

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

```
# exclude 20 random individuals from LH_griffin, then estimate their birth
# years from the birth years of their parents and/or offspring:
BYprobs <- CalcBYprobs(Pedigree = Ped_griffin,
                       LifeHistData = LH_griffin[-sample(1:200, 20),])
utils::head(BYprobs)
# For some individuals, the most-likely birth year will not be the actual
# birth year. But with sufficient quantity and quality of information, the
# actual birth year will be among the plausible ones, and be close to the
# most-likely birth year.
## Not run:
# heatmap
lattice::levelplot(t(BYprobs), aspect="fill", col.regions=hcl.colors)

## End(Not run)
```

`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 = "version2.9",
  qntl = 1 - 1e-05,
  Return = "Counts"
)
```

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.
Return	Either 'Counts' to return the threshold counts (default), or 'Probs' to return the mismatch probabilities from which these counts are calculated.

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))

# in sequoia() qntl depends on the number of genotyped individuals, to get an
# approximately constant false exclusion rate at dataset-level
sts <- SnpStats(Geno_griffin, Plot=FALSE, quiet=TRUE, calc_HWE=FALSE)
MAF <- ifelse(sts[,'AF'] < 0.5, sts[,'AF'], 1-sts[,'AF'])
sequoia::CalcMaxMismatch(Err = 0.001,
                         MAF = MAF,
                         qntl = 0.9999^(1/nrow(Geno_griffin)))
```

CalcOHLLR *Calculate OH and LLR for a pedigree*

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).

Usage

```
CalcOHLLR(
  Pedigree = NULL,
  GenoM = NULL,
  CalcLLR = TRUE,
  LifeHistData = NULL,
  AgePrior = FALSE,
  SeqList = NULL,
  Err = 1e-04,
  ErrFlavour = "version2.9",
  Tassign = 0.5,
  Tfilter = -2,
  Complex = "full",
  Herm = "no",
  useMaxOH = TRUE,
  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, one column per SNP, coded as 0, 1, 2, missing values as a negative number or NA. Row names must be individual IDs, column names are ignored. You can reformat data with GenoConvert, or use other packages to get it into a genlight object and then use as.matrix.

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 — data.frame with up to 6 columns:

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 number.

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

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

Year.last Last year in which individual could have had offspring. Can e.g. in mammals be the year before death for females, and year after death for males.

"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, and only integers are used. 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 — assumed per-locus genotyping error rate, as a single number, or a length 3 vector with P(hom|hom), P(het|hom), P(hom|het), or a 3x3 matrix. See details below. The error rate is presumed constant across SNPs, and missingness is presumed random with respect to actual genotype. Using Err >5% is not recommended, and Err >10% strongly discouraged. See Err_RADseq to convert per-allele rates at homozygous and heterozygous sites to the required length-3 vector, and ErrToM for further genotyping error details.

ErrFlavour	function that takes Err (single number) as input, and returns a length 3 vector or 3x3 matrix, or choose from inbuilt options 'version2.9', 'version2.0', 'version1.3', or 'version1.1', referring to the sequoia version in which they were the default. Ignored if Err is a vector or matrix.
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.
useMaxOH	when calculating likelihoods, skip any parent-offspring pairs for which the opposite homozygote count exceeds the maximum, which is calculated from the genotyping error rate by CalcMaxMismatch.
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 truly 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

```
# 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. Incorrect parents will have negative LLR.
PedZ <- Ped_HSg5[41:50, ]   # small example subset
PedZ$dam[1] <- PedZ$dam[10]
CalcOHLLR(PedZ, GenoM = SimGeno_example, CalcLLR = TRUE)

## Not run:
# with age data: makes some alternative relationships impossible, and thereby
# changes LLR(parent/not-parent)
```

```
Ped.LLR <- CalcOHLLR(PedZ, 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:
SeqOUT <- sequoia(Geno_griffin, LH_griffin, CalcLLR=FALSE,quiet=TRUE,
                  Plot=FALSE)
PedA <- CalcOHLLR(Pedigree = SeqOUT[["Pedigree"]][, 1:3], GenoM=Geno_griffin,
  LifeHistData = LH_griffin, AgePrior = TRUE, Complex = "full")
SummarySeq(PedA, Panels=c("LLR", "OH"))

## End(Not run)
```

`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.

NOTE 2: Relationship between a dummy/non-genotyped individual and another are expressed as relationships with *that dummies offspring*. So, if e.g. ID1=F0001 and ID2=i003_2001_M, and TopRel=FA, that means that dummy female F0001 is likely a full sibling of i003_2001_M. For further details see below.

Usage

```
CalcPairLL(
  Pairs = NULL,
  GenoM = NULL,
  Pedigree = NULL,
  LifeHistData = NULL,
  AgePrior = TRUE,
  SeqList = NULL,
  Module = "ped",
  Complex = "full",
  Herm = "no",
  InclDup = FALSE,
  Err = 1e-04,
  ErrFlavour = "version2.9",
```

```
  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, one column per SNP, coded as 0, 1, 2, missing values as a negative number or NA. Row names must be individual IDs, column names are ignored. You can reformat data with GenoConvert, or use other packages to get it into a genlight object and then use as.matrix.
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	data.frame with up to 6 columns: **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 number. **BY.min** minimum birth year, only used if BirthYear is missing **BY.max** maximum birth year, only used if BirthYear is missing

Year.last Last year in which individual could have had offspring. Can e.g. in mammals be the year before death for females, and year after death for males.

"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, and only integers are used. 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.

`Module` if `ped` (full pedigree), turn any non-genotyped parents in `Pedigree` into dummies to condition upon. If `par`, ignore any non-genotyped parents.

`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.

`InclDup` logical, include the likelihood for the two samples to be duplicates (originating from the same individual) in the output?

`Err` assumed per-locus genotyping error rate, as a single number, or a length 3 vector with P(hom|hom), P(het|hom), P(hom|het), or a 3x3 matrix. See details below. The error rate is presumed constant across SNPs, and missingness is presumed random with respect to actual genotype. Using `Err` >5% is not recommended, and `Err` >10% strongly discouraged. See `Err_RADseq` to convert per-allele rates at homozygous and heterozygous sites to the required length-3 vector, and `ErrToM` for further genotyping error details.

`ErrFlavour` function that takes `Err` (single number) as input, and returns a length 3 vector or 3x3 matrix, or choose from inbuilt options 'version2.9', 'version2.0', 'version1.3', or 'version1.1', referring to the sequoia version in which they were the default. Ignored if `Err` is a vector or matrix.

`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:

xx	Log10-Likelihood of this pair having relationship xx, with xx being the relationship abbreviations listed below.
TopRel	Abbreviation of most likely relationship
LLR	Log10-Likelihood ratio between most-likely and second most likely relationships. Other LLRs, e.g. between most-likely and unrelated, can easily be computed.

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. a male (Sex2=2) cannot be mother (patmat=1))
888	Already assigned in the provided pedigree (see dropPar arguments)
999	NA. If all values for the pair are 999, one or both individuals are not genotyped and not dummifiable.

Why does it say 444 (no can do) for all relationships?

This happens when the pair does not pass the initial check which prevents impossible configurations in combination with Pedigree. Specifically, it happens when either or both individuals are a parent in the pedigree, but

- the sex in Pairs is not consistent with that
- Pairs changes the age differences; it is too complex to check whether or not this still makes all pedigree links valid. Only setting the age difference to 'unknown' via Pairs is possible.

Why does it say 777 (impossible)?

This function uses the same machinery as sequoia, which will to save time not calculate the likelihood when it is quickly obvious that the pair cannot be related in the specified manner.

For PO (putative parent-offspring pairs) this is the case when:

- the sex of the candidate parent, via Pairs$Sex2 or LifeHistData, does not match Pairs$patmat, which defaults to 1 (maternal relatives, i.e. dam)
- a dam is already assigned via Pedigree and Pairs$dropPar1 ='none', and Pairs$patmat = 1
- the age difference is zero or otherwise impossible according to the age prior. It is either calculated from LifeHistData or specified via Pairs$AgeDif. The AgePrior can be specified directly, be taken from SeqList, or calculated automatically by MakeAgePrior when both Pedigree and LifeHistData are provided.

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.

Dummy individuals

For historical reasons, the relationships between a dummy ID1 and ID2 are reported *between the sibship and ID2*. So,

- PO: ID2 replaces dummy ID1; or merge dummy ID2 with dummy ID1
- FS, HS: ID1 parent of ID2
- GP: ID2 parent of ID1
- FA,HA: ID2 FS resp. HS of ID1

If ID1 is genotyped and ID2 is a dummy, the relationships are as when ID2 is genotyped.

If you wish to retrieve likelihoods for a different set of relationships, please contact me at <jisca.huisman@gmail.com> .

See Also

PlotPairLL to plot alternative relationship pairs from the output; LLtoProb to transform likelihoods to probabilities; CalcParentProbs which uses this function to calculate parental probabilities; GetRelM to find all pairwise relatives according to the pedigree; GetMaybeRel to get likely relative pairs based on the genetic data.

Examples

```
# Likelihoods depend on the presumed genotyping error rate:
CalcPairLL(Pairs = data.frame(ID1='i042_2003_F', ID2='i015_2001_F'),
           GenoM = Geno_griffin, Err = 1e-7, Plot=FALSE)
CalcPairLL(Pairs = data.frame(ID1='i042_2003_F', ID2='i015_2001_F'),
           GenoM = Geno_griffin, Err = 1e-3, Plot=FALSE)


## likelihoods underlying parent LLR in pedigree:
# Example: dams for bottom 3 individuals
tail(SeqOUT_griffin$PedigreePar, n=3)
# set up dataframe with these pairs. LLRdam & LLRsire ignore any co-parent
Pairs_d <- data.frame(ID1 = SeqOUT_griffin$PedigreePar$id[140:142],
                      ID2 = SeqOUT_griffin$PedigreePar$dam[140:142],
                      focal = "PO",
                      dropPar1 = 'both')

# Calculate LL's, conditional on the rest of the pedigree + age differences
CalcPairLL(Pairs_d, GenoM = Geno_griffin, Err = 1e-04,
           LifeHistData = LH_griffin, Pedigree = SeqOUT_griffin$PedigreePar)

# LLR changes when ignoring age and/or pedigree, as different relationships
# become (im)possible
CalcPairLL(Pairs_d, GenoM = Geno_griffin, Err = 1e-04)

# LLRpair is calculated conditional on co-parent, and min. of dam & sire LLR
Pairs_d$dropPar1 <- 'dam'
Pairs_s <- data.frame(ID1 = SeqOUT_griffin$PedigreePar$id[141:142],
                      ID2 = SeqOUT_griffin$PedigreePar$sire[141:142],
```

```
                      focal = "PO",
                      dropPar1 = 'sire')
CalcPairLL(rbind(Pairs_d, Pairs_s), GenoM = Geno_griffin, Err = 1e-04,
           LifeHistData = LH_griffin, Pedigree = SeqOUT_griffin$PedigreePar)


## likelihoods underlying LLR in getMaybeRel output:
MaybeRel_griffin$MaybePar[1:5, ]
FivePairs <- MaybeRel_griffin$MaybePar[1:5, c("ID1", "ID2", "Sex1", "Sex2")]
PairLL <- CalcPairLL(Pairs = rbind( cbind(FivePairs, focal = "PO"),
                                    cbind(FivePairs, focal = "HS"),
                                    cbind(FivePairs, focal = "GP")),
                     GenoM = Geno_griffin, Plot=FALSE)
PairLL[PairLL$ID1=="i121_2007_M", ]
# LL(FS)==222 : HSHA, HSGP, and/or 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
```

`CalcParentProbs` *Calculate assignment probabilities*

Description

For each assigned offspring-parent pair, calculate the probability they are parent-offspring vs otherwise related. Probabilities are scaled to sum to one across all possible* relationships between the pair or trio; see Details.

Usage

```
CalcParentProbs(Pedigree = NULL, GenoM = NULL, quiet = FALSE, nCores = 1, ...)
```

Arguments

`Pedigree`	dataframe with columns id-dam-sire. By default, any non-genotyped individuals are 'dummified'; use `Module='par'` to ignore them.
`GenoM`	numeric matrix with genotype data: One row per individual, one column per SNP, coded as 0, 1, 2, missing values as a negative number or NA. You can reformat data with `GenoConvert`, or use other packages to get it into a genlight object and then use `as.matrix`.
`quiet`	logical, suppress messages. No progress is printed when >1 core is used.
`nCores`	number of computer cores to use. If `2` or `4`, package **parallel** is used (other values are not applicable).
`...`	Additional arguments passed to `CalcPairLL`, such as the genotyping error rate `Err`, age information in `LifeHistData` and `AgePrior`, or `InclDup` to include the probability that the two samples are duplicates.

Details

The returned probabilities are calculated from the likelihoods used throughout the rest of this package, by scaling them to sum to one across all possible relationships. For `Complex='simp'` these are PO=parent-offspring, FS=full siblings, HS=half siblings, GP=grand-parental, FA=full avuncular, HA=third degree relatives (incl half avuncular), and U=unrelated. For `Complex='full'` there are numerous double relationship considered (PO & HS, HS & HA, etc), making both numerator and denominator in the scaling step less unambiguous, and the returned probabilities an approximation.

The likelihoods are calculated by calling `CalcPairLL` once or twice for each id-dam and id-sire pair: once not conditioning on the co-parent, and once conditional on the co-parent, if any. For genotyped individuals this is done with `focal='PO'`, and for dummy individuals with `focal='GP'`.

For relationships between a genotyped and a dummy individual, it may only be possible to determine that the genotyped individual is a second degree relative (GP, HS, or FA) to the dummy's offspring. This then results in a probability of at most 0.33, even when the two are indeed parent and offspring.

See `CalcPairLL` and the vignettes for further details.

Note that for large pedigrees this function can be fairly slow, especially when using `CalcPairLL`'s default `Module='ped'` and `Complex='full'`.

Subsetting the genotype data may give different results, as the likelihoods and thus the probabilities depend on the allele frequencies in the sample.

Value

the `Pedigree` dataframe with the three applicable columns renamed to id-dam-sire, and 7 additional columns:

`Probdam`	Probability that individual in dam column is the maternal parent, rather than otherwise related (LL(PO)/sum(LL))
`Probsire`	Analogous for sire
`Probpair`	Probability for id-dam-sire trio. Approximated as the minimum of dam conditional on sire and sire conditional on dam, thus not including e.g. both being siblings (those other configurations are considered by sequoia during pedigree reconstruction, but can (currently) not be accessed directly)
`dam_alt`, `sire_alt`	Most likely alternative (not PO) relationship between id-dam and id-sire, respectively
`Probdam_alt`, `Probsire_alt`	Probability of most likely alternative relationship

Warning

The probabilities will be less reliable with close inbreeding and double relationships. This function has not been tested yet with hermaphrodites, and is unlikely to give reliable results without further code updates.

See Also

`CalcPairLL`, `LLtoProb`

Examples

```
test_ped <- Ped_griffin[21:25,]
# add an incorrect sire to illustrate
test_ped$sire <- as.character(test_ped$sire)
test_ped$sire[5] <- 'i057_2003_M'
Ped_with_probs <- CalcParentProbs(test_ped, Geno_griffin)
print(Ped_with_probs, digits=2)
# Any non-genotyped non-'dummifiable' individuals are automatically skipped

# To get likelihoods for 'all' relationships, not just probabilities for
# PO & (next-)most-likely:
LL_sire_single <- CalcPairLL(
  Pairs = data.frame(id1=test_ped$id,
                     id2=test_ped$sire,
                     dropPar1='both', # drop both -> id2 as single parent
                     focal='PO'),
  Pedigree = Ped_griffin,   # pedigree to condition on
  GenoM = Geno_griffin, Plot=FALSE)
```

`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",
  Strict = TRUE,
  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).
Strict	Exclude any individuals genotyped for <5 genotyped for <5 up to version 2.4.1. Otherwise only excluded are (very nearly) monomorphic SNPs, SNPs scored for fewer than 2 individuals, and individuals scored for fewer than 2 SNPs.
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

```
GenoM <- SimGeno(Ped_HSg5, nSnp=400, CallRate = runif(400, 0.2, 0.8))
# the quick way:
GenoM.checked <- CheckGeno(GenoM, Return="GenoM")

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

# warning about  SNPs scored for <50% of individuals ?
# note: this is not necessarily a problem, and sometimes unavoidable.
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 filter out low call rate individuals: (also not necessarily a problem)
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",
  Pairs_suffix = "?"
)
```

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 as first three columns: ID1-ID2- relationship, e.g. as returned by GetMaybeRel. Column names and any additional columns are ignored. May be provided in addition to, or instead of Ped2.
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.
Pairs_suffix	symbol added to the relationship abbreviations derived from Pairs2, when both Ped2 and Pairs2 are provided. Can be an empty string.

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 subclasses. When a pair is related via multiple paths, the first-listed relationship is returned. To get all the different paths between a pair, use `GetRelM` with `Return='Array'`.

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

```
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
```

Conf_griffin *Example output from estimating confidence probabilities: griffins*

Description

Example output of EstConf, with the inferred pedigree in SeqOUT_griffin used as reference pedigree.

Usage

```
data(Conf_griffin)
```

Format

a list, see sequoia

Author(s)

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

See Also

Ped_griffin, Geno_griffin,

Examples

```
## Not run:
Conf_griffin <- EstConf(Pedigree = SeqOUT_griffin$Pedigree,
                        LifeHistData = LH_griffin,
                        args.sim = list(nSnp = 400, SnpError = 0.001,
                                        ParMis=0.4),
                        args.seq = list(Module = 'ped', Err=0.001),
                        nSim = 20,
                        nCores = 5,
                        quiet = TRUE)

## End(Not run)
```

`CountOH` *Count opposing homozygous SNPs between pairs of individuals*

Description

Quick identification of likely parents, as the number of opposing homozygous (OH) SNPs is expected to be zero for parent- offspring pairs in absence of genotyping errors, and greater than zero for all other pairs.

Usage

```
CountOH(x = NULL, ID2 = NULL, GenoM = NULL, max_OH = -1, quiet = FALSE)
```

Arguments

`x`	Either a matrix, dataframe or similar where the first two columns are individual IDs, or a vector with IDs. In the second case, you may provide `ID2`, and the output will be an ID1 x ID2 matrix; else the output will be an ID1 x ID1 matrix. Non-genotyped individuals are included in the results with all `NA`'s, and a warning.
`ID2`	optional second vector with IDs
`GenoM`	numeric matrix with genotype data: One row per individual, one column per SNP, coded as 0, 1, 2, missing values as a negative number or NA. Row names must be individual IDs, column names are ignored. You can reformat data with `GenoConvert`, or use other packages to get it into a genlight object and then use `as.matrix`.
`max_OH`	stop counting OH's for a pair if this value is reached, to reduce computation time. Ignored if negative value or equal to total number of SNPs.
`quiet`	suppress messages

Details

Counting the number of opposing homozygous (OH) SNPs is much faster than calculating likelihoods, and does not rely on an estimated genotyping error rate. It can therefore be useful during quality control, or to help figure out problems when assignment rate with `sequoia` is lower than expected.

See Also

`CalcPairLL` to calculate likelihoods for pairs, `CalcOHLLR` to calculate OH for a pedigree, `CalcMaxMismatch` for calculation of the maximum OH used by `sequoia` to filter potential parent-offspring pairs.

Examples

```
offspring_ids <- with(LH_HSg5, ID[BirthYear==2001])
candidate_father_ids <- with(LH_HSg5, ID[BirthYear==2000 & Sex==2])

OH_matrix <- CountOH(offspring_ids, candidate_father_ids, GenoM=Geno_HSg5)

hist(c(OH_matrix), breaks=c(0:40)-.5)
# with high quality SNP data, there is often a clear separation in OH counts
# between parent-offspring pairs (here: OH<3) and others (here: OH>4).
# BUT: non-PO close relatives may have very low OH counts by chance,
# and true PO pairs may have fairly high OH counts due to genotyping errors.
```

DyadCompare *Compare Dyads (DEPRECATED)*

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

```
## Not run:
DyadCompare(Ped1=Ped_HSg5, Ped2=SeqOUT_HSg5$Pedigree)

## End(Not run)
```

Err_RADseq *Convert Genotyping Error Rates from per-allele to per-locus*

Description

Convert per-allele genotyping rates at homozygous (E0) and heterozygous (E1) sites to a length-3 vector with per-locus error rates hom|hom, het|hom, hom|het.

Usage

```
Err_RADseq(E0 = 0.005, E1 = 0.05, Return = "vector")
```

Arguments

E0	per-allele genotyping rates at homozygous sites
E1	per-allele genotyping rates at heterozygous sites
Return	output format, 'vector' (default) or 'matrix'

Details

Estimation of per-allele genotyping rates is described in Bresadola et al (2020) - 'Estimating and accounting for genotyping errors in RAD-seq experiments', MER. The error model implemented here is identical to that in Table 1 of that paper, and the default values are also taken from that paper.

For further information on how the sequoia package handles genotyping errors, see ErrToM.

Value

Depending on Return, either:

- 'vector': a length 3 vector, with the probabilities (observed given actual) hom|other hom, het|hom, and hom|het.
- 'matrix': a 3x3 matrix, with probabilities of observed genotypes (columns) conditional on actual (rows)

Examples

```
# Compare with default error pattern (SNP chip based) :
Err_RADseq(E0=0.001, E1=0.05)
ErrToM(0.05*(1-0.05)*2, Return='vector')

# usage in sequoia() and other functions:
Err_low <- Err_RADseq(E0=0.002, E1=0.05)
Err_high <- Err_RADseq(E0=0.01, E1=0.15)
## Not run:
 SeqOUT_lowErr <- sequoia(GenoM, LHdata, Err=Err_low)
 SeqOUT_highErr <- sequoia(GenoM, LHdata, Err=Err_high)

# also usable for confidence estimates, and to explore potential consequences
# of the actual genotyping error rate being much higher/lower than assumed
EC <- EstConf(best_pedigree, LHdata, args.sim=list(SnpError=Err_high),
         args.seq=list(Err=Err_low))

## End(Not run)
```

`ErrToM` *Generate Genotyping Error Matrix*

Description

Make a vector or matrix specifying the genotyping error pattern, or a function to generate such a vector/matrix from a single value Err.

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 = "version2.9", Return = "matrix")
```

Arguments

`Err`	estimated genotyping error rate, as a single number, or 3x3 or 4x4 matrix, or length 3 vector. 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`. If a vector, these are the probabilities (observed given actual) hom|other hom, het|hom, and hom|het.
`flavour`	vector-generating or matrix-generating function, or one of 'version2.9', 'version2.0', 'version1.3' (='SNPchip'), 'version1.1' (='version111'), referring to the sequoia version in which it was used as default. Only used if `Err` is a single number.
`Return`	output, 'matrix' (default), 'vector', 'function' (matrix-generating), or 'v_function' (vector-generating)

Details

By default (`flavour = "version2.9"`), `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.

This function, and throughout the package, it is assumed that the two alleles A and a are equivalent. Thus, using notation P(observed genotype |actual genotype), that $P(AA|aa) = P(aa|AA)$, $P(aa|Aa) = P(AA|Aa)$, and $P(aA|aa) = P(aA|AA)$.

version	**hom|hom**	**het|hom**	**hom|het**
2.9	$(E/2)^2$	$E-(E/2)^2$	$E/2$
2.0	$(E/2)^2$	$E(1-E/2)$	$E/2$
1.3	$(E/2)^2$	E	$E/2$
1.1	$E/2$	$E/2$	$E/2$
0.9	0	E	$E/2$

or in matrix form, Pr(observed genotype (columns) | actual genotype (rows)):

version2.9:

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

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	$1-E$

When `Err` is a length 3 vector, or if `Return = 'vector'` these are the following probabilities:

- hom|hom: an actual homozygote is observed as the other homozygote (E_1)
- het|hom: an actual homozygote is observed as heterozygote (E_2)
- hom|het: an actual heterozygote is observed as homozygote (E_3)

and Pr(observed genotype (columns) | actual genotype (rows)) is then:

	0	**1**	**2**
0	$1-E_1-E_2$	E_2	E_1
1	E_3	$1-2E_3$	E_3
2	E_1	E_2	$1-E_1-E_2$

When the SNPs are scored via sequencing (e.g. RADseq or DArTseq), the 3rd error rate (hom|het) is typically considerably higher than the other two, while for SNP arrays it tends to be similar to P(het|hom).

Value

Depending on `Return`, either:

- `'matrix'`: a 3x3 matrix, with probabilities of observed genotypes (columns) conditional on actual (rows)
- `'function'`: a function taking a single value `Err` as input, and generating a 3x3 matrix
- `'vector'`: a length 3 vector, with the probabilities (observed given actual) hom|other hom, het|hom, and hom|het.

Examples

```
ErM <- ErrToM(Err = 0.05)
ErM
ErrToM(ErM, Return = 'vector')
```

```
# use error matrix from Whalen, Gorjanc & Hickey 2018
funE <- function(E) {
 matrix(c(1-E*3/4, E/2, E/4,
          E/4, 1-2*E/4, E/4,
          E/4, E/2, 1-E*3/4),
          3,3, byrow=TRUE)  }
ErrToM(Err = 0.05, flavour = funE)
# equivalent to:
ErrToM(Err = c(0.05/4, 0.05/2, 0.05/4))
```

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), birth year, and optionally BY.min - BY.max - YearLast.
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 rounds 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). With large datasets, the amount of computer memory may be the limiting factor for the number of cores you can use.
quiet	suppress messages. TRUE runs SimGeno and sequoia quietly, 'very' also suppresses other messages and the simulation counter when nCores=1 (there is no simulation 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 separately 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 simulation the inferred pedigree based on the simulated data
SimSNPd	a list with for each simulation the IDs of the individuals simulated to have been genotyped
PedComp.fwd	array with Counts from the 'forward' PedCompare, from which PedErrors is calculated
RunParams	a list with the call to EstConf as a semi-nested list (args.sim, args.seq, nSim, nCores), 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` simulations

Array `PedErrors` has three dimensions:

`class`
: - `FalseNeg`(atives): could have been assigned but was not (individual + parent both genotyped or dummifiable; 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 round, a new random set of parents (proportion set by `ParMis`) is mimicked to be non-genotyped. In addition, SNPs are assumed to segregate independently.

An experimental version offering more fine-grained control is available at https://github.com/JiscaH/sequoiaExtra .

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.

Errors

If you have a large pedigree and try to run this function on multiple cores, you may run into "Cannot allocate vector of size ..." errors or even unexpected crashes: there is not enough computer memory for each separate run. Try reducing ‘nCores‘.

See Also

`SimGeno`, `sequoia`, `PedCompare`.

Examples

```
# estimate proportion of parents that are genotyped (= 1 - ParMis)
prop_parents_genotyped <- c(
  dam = mean(unique(SeqOUT_griffin$Pedigree$dam) %in% rownames(Geno_griffin)),
sire = mean(unique(SeqOUT_griffin$Pedigree$sire) %in% rownames(Geno_griffin))
)

# Example for parentage assignment only
conf_grif <- EstConf(Pedigree = SeqOUT_griffin$Pedigree,
               LifeHistData = SeqOUT_griffin$LifeHist,
               args.sim = list(nSnp = 150,   # no. in actual data, or what-if
                               SnpError = 5e-3,  # best estimate, or what-if
                               CallRate=0.9,     # from SnpStats()
                               ParMis=c(0.28, 0.22)),  # calc'd above
               args.seq = list(Err=5e-3, Module="par"),  # as in real run
               nSim = 1,   # try-out, proper run >=20 (10 if huge pedigree)
               nCores=1)

# parent-pair confidence, per category (Genotyped/Dummy/None)
conf_grif$ConfProb

# Proportion of true parents that was correctly assigned
1 - apply(conf_grif$PedErrors, MARGIN=c('cat','parent'), FUN=sum, na.rm=TRUE)

# add columns with confidence probabilities to pedigree
# first add columns with category (G/D/X)
Ped.withConf <- getAssignCat(Pedigree = SeqOUT_griffin$Pedigree,
                             SNPd = SeqOUT_griffin$PedigreePar$id)
Ped.withConf <- merge(Ped.withConf, conf_grif$ConfProb, all.x=TRUE,
                      sort=FALSE)  # (note: merge() messes up column order)
head(Ped.withConf[Ped.withConf$dam.cat=="G", ])

# save output summary
## Not run:
conf_griff[['Note']] <- 'You could add a note'
saveRDS(conf_grif[c('ConfProb','PedComp.fwd','RunParams','RunTime','Note')],
   file = 'conf_200SNPs_Err005_Callrate80.RDS')

## End(Not run)

## overall assignment rate (AR), error rate (ER) & runtime
AR_max <- sum(!is.na(Ped_griffin$dam)) + sum(!is.na(Ped_griffin$sire))
ER_max <- 2*nrow(Ped_griffin)
PCT <- conf_grif$PedComp.fwd[,'TT',,]   # Total-Total counts
list(AR = mean(apply(PCT[,'Match',],1,sum)/AR_max),  # sum over dam+sire
     ER = mean(apply(PCT[,c('Mismatch','P2only'),],1,sum)/ER_max),
     Time = mean(conf_grif$RunTime)/60)   # runtime in seconds --> minutes


## P(actual FS | inferred as FS) etc.
## Not run:
PairL <- list()
```

```
for (i in 1:length(conf_grif$Pedigree.inferred)) {  # nSim
  cat(i, "\t")
  PairL[[i]] <- ComparePairs(conf_grif$Pedigree.reference,
                             conf_grif$Pedigree.inferred[[i]],
                             GenBack=1, patmat=TRUE, ExcludeDummies = TRUE,
                             Return="Counts")
}
# P(actual relationship (Ped1) | inferred relationship (Ped2))
PairRel.prop.A <- plyr::laply(PairL, function(M)
                     sweep(M, MARGIN='Ped2', STATS=colSums(M), FUN="/"))
PairRel.prop <- apply(PairRel.prop.A, 2:3, mean, na.rm=TRUE) #avg across sims
round(PairRel.prop, 3)
# or: P(inferred relationship | actual relationship)
PairRel.prop2 <- plyr::laply(PairL, function(M)
   sweep(M, MARGIN='Ped1', STATS=rowSums(M), FUN="/"))

## End(Not run)

## Not run:
# confidence probability vs. sibship size
source('https://raw.githubusercontent.com/JiscaH/sequoiaExtra/main/conf_vs_sibsize.R')
conf_grif_nOff <- Conf_by_nOff(conf_grif)
conf_grif_nOff['conf',,'GD',]
conf_grif_nOff['N',,'GD',]

## End(Not run)
```

`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.

Examples

```
## Not run:
PC_griffin <- PedCompare(Ped1 = cbind(FieldMums_griffin, sire=NA),
                         Ped2 = SeqOUT_griffin$Pedigree)

## End(Not run)
```

FindFamilies *Assign Family IDs*

Description

Find clusters of connected individuals in a pedigree, and assign each cluster a unique family ID (FID).

Usage

```
FindFamilies(Pedigree = NULL, SeqList = NULL, MaybeRel = NULL)
```

Arguments

Pedigree	dataframe with columns id - parent1 - parent2; only the first 3 columns will be used.
SeqList	list as returned by sequoia. If Pedigree is not provided, the element Pedigree from this list will be used if present, and element Pedigreepar otherwise.
MaybeRel	Output from GetMaybeRel, a dataframe with probable but non-assigned relatives.

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 numeric vector with length equal to the number of unique individuals in the pedigree (i.e. number of rows in pedigree after running PedPolish on Pedigree).

See Also

GetAncestors, GetDescendants, getGenerations

Examples

```
PedG <- SeqOUT_griffin$PedigreePar[,1:3]
FID_G <- FindFamilies(PedG)
PedG[FID_G==4,]
```

Geno_griffin *Example genotype file: Griffins*

Description

Simulated genotype data from Pedigree Ped_griffin

Usage

```
data(Geno_griffin)
```

Format

A genotype matrix with 142 rows (individuals) 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

SimGeno

Geno_HSg5 *Example genotype file: 'HSg5'*

Description

Simulated genotype data for all* individuals in Pedigree Ped_HSg5 (*: with 40

Usage

```
data(Geno_HSg5)
```

Format

A genotype matrix with 920 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

LH_HSg5, SimGeno, SeqOUT_HSg5

Examples

```
## Not run:
# this output was created as follows:
Geno_HSg5 <- SimGeno(Ped = Ped_HSg5, nSnp = 200, ParMis=0.4,
                     CallRate = 0.9, SnpError = 0.005)

## End(Not run)
```

GenoConvert *Convert Genotype Data*

Description

Convert genotype data in various formats to sequoia's 1-column-per-marker format, PLINK's ped 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", "-1", c("0")[InFormat %in% c("col",
    "ped")]),
  sep = c(" ", "\t", ",", ";"),
  header = NA,
  IDcol = NA,
  FIDcol = NA,
  FIDsep = "__",
  dropcol = NA,
  quiet = FALSE
)
```

Arguments

InData	dataframe, matrix or genlight object with genotypes to be converted.
InFile	character string with name of genotype file to be converted.
InFormat	One of 'seq' (sequoia), 'ped' (PLINK .ped file), 'col' (COLONY), 'raw' (PLINK –recodeA), 'vcf' (requires library {vcfR}), 'single' (1 column per SNP), or 'double' (2 columns per SNP); 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', 'col', and 'ped' are implemented. For 'ped' also a sham .map file is created, so that the file can be read by PLINK. Only for 'ped' are extensions .ped & .map added to the specified OutFile filename.
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. Ignored if package **data.table** is present or if InFormat='vcf' or 'vcf.gz'. 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.

If package **data.table** is detected, fread is used to read in the data from file. Otherwise, a combination of readLines and strsplit is used.

Value

A genotype matrix in the specified output format; the default sequoia format ('seq') has 1 column per SNP coded in 0/1/2 format (major homozygote /heterozygote /minor homozygote) with -9 for missing values, sample IDs in row names and SNP names in column names. If 'OutFile' is specified, the matrix is written to this file and nothing is returned inside R.

Input formats

The following formats can be specified by `InFormat`:

seq (sequoia) genotypes are coded as 0, 1, 2, missing as -9 (in input any negative number or NA are OK), in 1 column per marker. Column 1 contains IDs, there is no header row.

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). If an associated .map file exists, SNP names will be read from there.

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

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

vcf (VCF) genotypes are coded as '0/0','0/1','1/1', variable number of header rows followed by 1 row per SNP, with various columns of metadata followed by 1 column per individual. Requires package **vcfR**.

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.

Any warnings generated by `CheckGeno` regarding SNPs scored for few individuals and/or individuals scored for few SNPs etc. are only for your information; none are excluded from `GenoConvert`'s output, but these SNPs and/or individuals will be excluded during pre-processing of the data in any of the other functions in this package.

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", InFormat='raw')
# which is the same as
GenoM <- GenoConvert(PlinkOUT.raw, InFormat='single',
                     IDcol=2, dropcol=c(1,3:6), header=TRUE)
# (but it will complain that InFormat='single' is not consistent with .raw
# file extension)

# 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)

# can also write to file, e.g. simulated genotypes:
GenoConvert(Geno_A, InFormat='seq', OutFormat='ped', OutFile = sim_genotypes)

## End(Not run)
```

GetAncestors *Get ancestors*

Description

get all ancestors of an individual

Usage

```
GetAncestors(id, Pedigree)
```

Arguments

id	id of the individual
Pedigree	dataframe with columns id - parent1 - parent2; only the first 3 columns will be used.

Value

a list with as first element id, second parents, third grandparents, etc.. Each element is a vector with ids, the first three elements are named, the rest numbered. Ancestors are unsorted within each list element.

Examples

```
Anc_i200  <- GetAncestors('i200_2010_F', Ped_griffin)
```

---

getAssignCat *Assignability of Reference Pedigree*

Description

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

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 is 1 genotyped offspring, and • '1sib1GP': at least 1 grandparent (G or D) or 1 more offspring (G or D); these are potentially assignable by sequoia • '2sib': at least 1 more offspring (i.e. 2 siblings). Old default for PedCompare. .

Details

Non-genotyped individuals can potentially be substituted by a dummy during pedigree reconstruction by sequoia when they have at least one genotyped offspring, and either one additional offspring (genotyped or dummy) or an genotyped/dummy parent (i.e. a grandparent to the genotyped offspring).

Note that this is the bare minimum requirement; e.g. grandparents are often indistinguishable from full avuncular (see sequoia and vignette for details). G-G parent-offspring pairs are only assignable if there is age information, or information from the surrounding pedigree, to tell which of the two is the parent.

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.

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

Examples

```
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)
```

GetDescendants *Get descendants*

Description

get all descendants of an individual

Usage

```
GetDescendants(id, Pedigree)
```

Arguments

id	id of the individual
Pedigree	dataframe with columns id - parent1 - parent2; only the first 3 columns will be used.

Value

a list with as first element id, second offspring, third grand-offspring, etc.. Each element is a vector with ids, the first three elements are named, the rest numbered.

---

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. Offspring of G0 X G0 are G1, offspring of G0 X G1 or G1 x G1 are G2, etc. NA indicates a pedigree loop where an individual is its own ancestor (or that the pedigree has >1000 generations).

If no output name is specified, no results are returned, only an error message when the pedigree contains a loop.

To get more details about a pedigree loop, you can use https://github.com/JiscaH/sequoiaExtra/blob/main/find_pedigree_loop

See Also

GetAncestors, GetDescendants to get all ancestors resp. descendants of a specific individual (with a warning if it is its own ancestor); FindFamilies to find connected sub-pedigrees.

Examples

```
# returns nothing if OK, else error:
getGenerations(SeqOUT_griffin$Pedigree)

# returns vector with generation numbers:
G <- getGenerations(SeqOUT_griffin$Pedigree, StopIfInvalid=FALSE)
table(G, useNA='ifany')
Ped_plus_G <- cbind(SeqOUT_griffin$Pedigree, G)
```

`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).

Details

This is a simple helper function to extract values from `AgePriorExtra`, e.g. to use together with `CalcPairLL`.

Value

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

Examples

```
# For a pair with unknown age difference, explore the difference age-based
# LLRs for all relationships, for a range of plausible age differences.
PairsG <- data.frame(ID1 = 'A', ID2 = 'B', AgeDif = rep(c(-2,2,3),2),
                     PatMat = rep(1:2, each=3))
cbind(PairsG,
      GetLLRAge(SeqOUT_griffin$AgePriorExtra,
                agedif = PairsG$AgeDif, patmat = PairsG$PatMat))
```

`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,
  Module = "par",
  Complex = "full",
  Herm = "no",
  Err = 1e-04,
  ErrFlavour = "version2.9",
  Tassign = 0.5,
  Tfilter = -2,
  MaxPairs = 7 * nrow(GenoM),
  quiet = FALSE
)
```

Arguments

`GenoM`	numeric matrix with genotype data: One row per individual, one column per SNP, coded as 0, 1, 2, missing values as a negative number or NA. Row names must be individual IDs, column names are ignored. You can reformat data with `GenoConvert`, or use other packages to get it into a genlight object and then use `as.matrix`.
`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`	data.frame with up to 6 columns: **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 number.

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

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

Year.last Last year in which individual could have had offspring. Can e.g. in mammals be the year before death for females, and year after death for males.

"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, and only integers are used. 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 $P(A|R)/P(A) > 0$).

`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` assumed per-locus genotyping error rate, as a single number, or a length 3 vector with P(hom|hom), P(het|hom), P(hom|het), or a 3x3 matrix. See details below. The error rate is presumed constant across SNPs, and missingness is presumed random with respect to actual genotype. Using `Err` >5% is not recommended, and `Err` >10% strongly discouraged. See `Err_RADseq` to convert per-allele rates at homozygous and heterozygous sites to the required length-3 vector, and `ErrToM` for further genotyping error details.

`ErrFlavour` function that takes `Err` (single number) as input, and returns a length 3 vector or 3x3 matrix, or choose from inbuilt options 'version2.9', 'version2.0', 'version1.3', or 'version1.1', referring to the sequoia version in which they were the default. Ignored if `Err` is a vector or matrix.

`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 — A dataframe with non-assigned likely parent-offspring pairs, with columns:

- ID1
- ID2
- TopRel: the most likely relationship, using abbreviations listed below
- LLR: Log10-Likelihood Ratio between most likely and next most likely relationship
- OH: Number of loci at which the two individuals are opposite homozygotes
- BirthYear1: Birth year of ID1 (copied from LifeHistData)
- BirthYear2
- AgeDif: Age difference; BirthYear1 - BirthYear2
- Sex1: Sex of ID1 (copied from LifeHistData)
- Sex2
- SnpdBoth: Number of loci at which the two individuals are both successfully genotyped

MaybeRel — A dataframe with non-assigned likely pairs of relatives, with columns identical to MaybePar

MaybeTrio — A dataframe with non-assigned parent-parent-offspring trios, with columns:

- ID
- parent1
- parent2
- TopRel: the most likely relationship, using abbreviations listed below
- LLRparent1: Log10-Likelihood Ratio between parent1 being a parent of ID vs the next most likely relationship between the pair, ignoring parent2
- LLRparent2: as LLRparent1
- LLRpair: LLR for the parental pair, versus the next most likely configuration between the three individuals (with one or neither parent assigned)
- OHparent1: Number of loci at which ID and parent1 are opposite homozygotes
- OHparent2: as OHparent1

- 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.parent1: Number of loci at which ID and parent1 are both successfully genotyped
- SNPd.id.parent2: as SNPd.id.parent1

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

```
## Not run:
# without conditioning on pedigree
MaybeRel_griffin <- GetMaybeRel(GenoM=Geno_griffin, Err=0.001, Module='par')

## End(Not run)
names(MaybeRel_griffin)

# conditioning on pedigree
MaybePO <- GetMaybeRel(GenoM = Geno_griffin, SeqList = SeqOUT_griffin,
                       Module = 'par')
head(MaybePO$MaybePar)

# instead of providing the entire SeqList, one may specify the relevant
# elements separately
Maybe <- GetMaybeRel(GenoM = Geno_griffin,
                     Pedigree = SeqOUT_griffin$PedigreePar,
                     LifeHistData = LH_griffin,
                     Err=0.0001, Complex = "full",
                     Module = "ped")
head(Maybe$MaybeRel)

# visualise results, turn dataframe into matrix first:
```

```
MaybeM <- GetRelM(Pairs = Maybe$MaybeRel)
PlotRelPairs(MaybeM)
# or combine with pedigree (note suffix '?')
RelM <- GetRelM(Pedigree =SeqOUT_griffin$PedigreePar, Pairs = Maybe$MaybeRel)
PlotRelPairs(RelM)
```

`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,
  directed = TRUE,
  Return = "Matrix",
  Pairs_suffix = "?"
)
```

Arguments

`Pedigree`	dataframe with columns id - dam - sire.
`Pairs`	dataframe with columns ID1 - ID2 - Rel, e.g. as returned by `GetMaybeRel`. Combining `Pedigree` and `Pairs` works best if the relationships are coded as listed below.
`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.
`directed`	logical, distinguish between e.g. ID1=offspring, ID2=mother ('M') and ID1=mother, ID2=offspring ('O')? Defaults to TRUE; if FALSE both are are scored as 'PO', as are father-offspring pairs, and all grandparent– grand-offspring pairs are scored as 'GPO', and avuncular pairs as 'FNA' and 'HNA'. Not (currently) compatible with `patmat`. When `Return`='List', each pair is included twice (as ID1-ID2 & ID2-ID1)
`Return`	'Matrix', 'Array', or 'List'. 'Matrix' returns an N x N matrix with the closest relationship between each pair. 'Array' returns 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. 'List' returns a list with for each of the R considered relationships a 2-column matrix with the IDs of the pairs having such a relationship. The size of the list (in Mb) is much smaller than for the matrix or array, and this is therefore the only format suitable for pedigrees with many thousands of individuals. If `Pairs` is specified, the only possible return type is 'Matrix'.

`Pairs_suffix` symbol added to the relationship abbreviations derived from `Pairs`, when both `Pedigree` and `Pairs` are provided. Can be an empty string.

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'. Double half-avuncular relationships are indicated as both FA and 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` or of the `List`:

`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.
FN	Full nephew/niece
HA	Half avuncular
HN	Half nephew/niece
DFC1	Double full first cousin
FC1	Full first cousin
U	Unrelated (or otherwise related)
X	Unknown, e.g. when only Pairs is provided and does not include this pair

See Also

ComparePairs for comparing pairwise relationships between two pedigrees; PlotRelPairs.

Examples

```
Rel.griffin <- GetRelM(Ped_griffin, directed=FALSE)  # few categories
Rel.griffin <- GetRelM(Ped_griffin, patmat=TRUE, GenBack=2)  # many cat.
table(as.vector(Rel.griffin))
# turning matrix into vector first makes table() much faster
PlotRelPairs(Rel.griffin)
```

Inherit_patterns *Inheritance patterns*

Description

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

Usage

```
data(Inherit_patterns)
```

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

LH_griffin *Example life history data: griffins*

Description

Example life history data associated with the 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

Examples

```
## Not run:
BY <- rep(c(2001:2010), each=20)
Sex <- sample.int(n=2, size=200, replace=TRUE)
ID <- paste0("i", formatC(1:200, width=3, flag="0"), "_", BY, "_",
             ifelse(Sex==1, "F", "M"))
LH_griffin <- data.frame(ID, Sex, BirthYear = BY)

## End(Not run)
```

LH_HSg5 *Example life history file: 'HSg5'*

Description

This is the life history 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

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)
```

`LLtoProb` *transform log-likelihoods to probabilities*

Description

transform a vector with log10 likelihoods to a vector with probabilities summing to one.

Usage

```
LLtoProb(LLv)
```

Arguments

`LLv`	a vector with log10-likelihoods. All values >0 are set to NA.

Details

The returned probabilities are calculated from the likelihoods used throughout the rest of this package, by scaling them to sum to one across all possible relationships. For `Complex='simp'` these are PO=parent-offspring, FS=full siblings, HS=half siblings, GP=grand-parental, FA=full avuncular, HA=third degree relatives (incl half avuncular), and U=unrelated. For `Complex='full'` there are numerous double relationship considered (PO & HS, HS & HA, etc), making both numerator and denominator in the scaling step less unambiguous, and the returned probabilities an approximation.

Computational under/overflow issues are reduced by subtracted the maximum value before converting from log to regular scale. Probabilities that would still be smaller than the machine precision (`(LL - min(LL)/2) < log10(.Machine$double.xmin)`) are set to NA en then to 0, instead of `-Inf`, to avoid issues when scaling to sum to 1.

Value

a vector with probabilities, with the same length and names.

Examples

```
LL_pairs <- CalcPairLL(data.frame(ID1='i042_2003_F',
                          ID2=c('i015_2001_F', 'i022_2002_F', 'i035_2002_F')),
                  GenoM = Geno_griffin, Complex='simp', Err=1e-3, Plot=FALSE)
prob_pairs <- t(apply(LL_pairs[,10:16], MARGIN=1, LLtoProb))
# - or -
prob_pairs <- plyr::aaply(as.matrix(LL_pairs[,10:16]), .margin=1, LLtoProb)
round(prob_pairs, 3)

# i035_2002_F is MHS of i042_2003_F, but when not conditioning on any other
# relatives has a higher LL to be 3rd degree relative (HA)
# (possibly genotyping errors, or just randomness of Mendelian inheritance)
```

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,
  MinAgeParent = 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) - birthyear (optional columns BY.min - BY.max - YearLast not used), with unknown birth years coded as negative numbers or NA. "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.
MinAgeParent	minimum age of a parent, a single number (min across dams and sires) or a vector of length two (dams, sires). Defaults to 1. When there is a conflict with the minimum age in the pedigree, the pedigree takes precedent.
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 set to latest - earliest birth year in LifeHistData, or estimated from the pedigree if one is provided. 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 (from zero up to max(MaxAgeParent)+1) 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, flattened 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.

A few outlier birth years can heavily influence the output; these may be easiest to spot with Smooth=FALSE, Flatten=FALSE.

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. Not compatible with `Smooth`. If the largest age difference in the pedigree is larger than the specified `MaxAgeParent`, the pedigree takes precedent (i.e. the largest of the two is used).

@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(MaxAgeParent = c(2,3))
MakeAgePrior(Discrete=TRUE)

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

# overlapping generations:
# 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)

# Call from sequoia() via args.AP:
Seq_HSg5 <- sequoia(SimGeno_example, LH_HSg5, Module="par",
                args.AP=list(Discrete = TRUE),  # non-overlapping generations
                CalcLLR = FALSE,   # skip time-consuming calculation of LLR's
                Plot = FALSE)      # no summary plots when finished
```

MaybeRel_griffin *Example output from check for relatives: griffins*

Description

Example output of a check for parent-offspring pairs and parent-parent-offspring trios with GetMaybeRel, with Geno_griffin as input (simulated from Ped_griffin).

Usage

```
data(MaybeRel_griffin)
```

Format

a list with 2 dataframes, 'MaybePar' and 'MaybeTrio'. See GetMaybeRel for further details.

Author(s)

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

See Also

SeqOUT_griffin

Examples

```
## Not run:
MaybeRel_griffin <- GetMaybeRel(GenoM = Geno_griffin, Err=0.001,
                                Module = 'par')

## End(Not run)
```

MkGenoErrors *Simulate Genotyping Errors*

Description

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

Usage

```
MkGenoErrors(
  SGeno,
  CallRate = 0.99,
  SnpError = 5e-04,
  ErrorFV = function(E) c((E/2)^2, E - (E/2)^2, E/2),
  ErrorFM = NULL,
  Error.shape = 0.5,
  CallRate.shape = 1,
  WithLog = FALSE
)
```

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 as parent) not included in this vector are presumed non-genotyped.
SnpError	either a single value which will be combined with ErrorFV, or a length 3 vector with probabilities (observed given actual) hom|other hom, het|hom, and hom|het; OR a vector or 3XnSnp matrix with the genotyping error rate(s) for each SNP.
ErrorFV	function taking the error rate (scalar) as argument and returning a length 3 vector with hom->other hom, hom->het, het->hom. May be an 'ErrFlavour', e.g. 'version2.9'.
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). See below for details. To use, set ErrorFV = NULL
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.
WithLog	Include dataframe in output with which datapoints have been edited, with columns id - SNP - actual (original, input) - observed (edited, output).

Value

The input genotype matrix, with some genotypes replaced, and some set to missing (-9). If WithLog=TRUE, a list with 3 elements: GenoM, Log, and Counts_actual (genotype counts in input, to allow double checking of simulated genotyping error rate).

Ped_griffin *Example pedigree: griffins*

Description

Example pedigree with overlapping generations and polygamy.

Usage

```
data(Ped_griffin)
```

Format

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

Code

The R code used to create this pedigree can be found in /data-raw.

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: 'HSg5'*

Description

A pedigree with five non-overlapping generations and considerable inbreeding. Each female mated with two random males and each male with three random females, producing four full-sib offspring per mating. This is **Pedigree II** in the paper.

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

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 = "1sib1GP",
  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`:

- '1sib' : sibship of size 1, with or without grandparents. The latter aren't really a sibship, but can be useful in some situations.
- '1sib1GP': sibship of size 1 with at least 1 grandparent (default)
- '2sib': at least 2 siblings, with or without grandparents (default prior to version 2.4)

`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, ComparePairs 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; getAssignCat for all parents in the reference pedigree that could have been assigned; CalcOHLLR to check how well an 'old' pedigree fits with the SNP data.

Examples

```
compare <- PedCompare(Ped_griffin, SeqOUT_griffin$Pedigree)
compare$Counts["TT",,]  # totals only; 45 dams & 47 sires non-assigned
compare$Counts[,,"dam"]  # dams only

# inspect non-assigned in Ped2, id genotyped, dam might-be-dummy
PedM <- compare$MergedPed  # for brevity
PedM[PedM$id.dam.cat=='GD' & PedM$dam.class=='P1only',]
# zoom in on specific dam
PedM[which(PedM$dam.1=="i011_2001_F"), ]
# no sire for 'i034_2002_F' -> impossible to tell if half-sibs or avuncular

# 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",]

# default before version 3.5: minSibSize = '2sib'
compare_2s <- PedCompare(Ped_griffin, SeqOUT_griffin$Pedigree,
                         minSibSize = '2sib')
compare_2s$Counts[,,"dam"]  # note decrease in Total 'dummies
with(compare_2s$MergedPed, table(id.dam.cat, dam.class))
# some with id.cat = 'X' or dam.cat='X' are nonetheless dam.class='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.. Does not sort parents before offspring.

Usage

```
PedPolish(
  Pedigree,
  gID = NULL,
  ZeroToNA = TRUE,
  NAToZero = FALSE,
  DropNonSNPd = TRUE,
  addParentRows = TRUE,
  FillParents = FALSE,
  KeepAllColumns = TRUE,
  KeepAllRows = 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.
addParentRows	add rows for any dams, sires, or individuals in gID not yet occurring in the id column.
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.
KeepAllColumns	Keep all columns in Pedigree (TRUE, default), or only id - dam - sire (FALSE).
KeepAllRows	Keep all rows in Pedigree (TRUE), or drop rows where id = NA (FALSE, default). Duplicated rows are always removed.
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 an exact or partial match with (case is ignored):

id "id", "iid", "off"

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 by this function if necessary.

Examples

```
PedZ <- Ped_HSg5[41:50, ]
PedPolish(PedZ)
PedPolish(PedZ, gID = rownames(SimGeno_example)[30:40], DropNonSNPd=TRUE)

## 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

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

```
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

```
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

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

`PlotPropAssigned` *Plot proportion of individuals that has a parent assigned*

Description

For any pedigree, plot the proportion of individuals that has a genotyped, dummy, observed, or no dam/sire assigned.

Usage

```
PlotPropAssigned(Pedigree = NULL, DumPrefix = c("F0", "M0"), SNPd = NULL, ...)
```

Arguments

Pedigree	dataframe where the first 3 columns are id, dam, sire.
DumPrefix	character vector with prefixes for dummy dams (mothers) and sires (fathers), used to distinguish between dummies and non-dummies.
SNPd	character vector with ids of genotyped individuals (e.g. rownames of genotype matrix).
...	further arguments passed to barplot

Details

This function offers a more flexible interface to some of the plots included in SummarySeq

Value

a 2x4 matrix with counts, returned invisibly.

Examples

```
PlotPropAssigned(SeqOUT_griffin$Pedigree, SNPd = rownames(Geno_griffin))
```

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

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

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

## End(Not run)

# 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])
```

PlotSeqSum *Plot Summary Overview of sequoia Output*

Description

visualise the numbers of assigned parents, sibship sizes, and parental LLRs

Usage

```
PlotSeqSum(SeqSum, Pedigree = NULL, Panels = "all", ask = TRUE)
```

Arguments

SeqSum	list output from SummarySeq.
Pedigree	dataframe with at least id, dam and sire in columns 1-3, respectively. If columns with parental LLRs and/or Mendelian errors are present, these will be plotted as well.
Panels	character vector with panel(s) to plot. Choose from 'all', 'G.parents' (parents of genotyped individuals), 'D.parents' (parents of dummies), 'O.parents' (parents of non-genotyped non-dummies), sibships', 'LLR', 'OH'.
ask	ask for user key stroke before proceeding to next plot.

Examples

```
sumry <- SummarySeq(SeqOUT_griffin, Plot=FALSE)
PlotSeqSum(sumry, SeqOUT_griffin$Pedigree, Panels='all', ask=FALSE)
```

SeqOUT_griffin *Example output from pedigree inference: griffins*

Description

Example output of a sequoia run including sibship clustering, with Geno_griffin as input (simulated from Ped_griffin).

Usage

```
data(SeqOUT_griffin)
```

Format

a list, see sequoia

Author(s)

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

See Also

sequoia

Examples

```
## Not run:
SeqOUT_griffin <- sequoia(GenoM = Geno_griffin,
                          LifeHistData = LH_griffin,
                          Module = 'ped')

## End(Not run)
```

SeqOUT_HSg5 *Example output from pedigree inference: 'HSg5'*

Description

Example output of a sequoia run including sibship clustering, based on Pedigree Geno_HSg5.

Usage

```
data(SeqOUT_HSg5)
```

Format

a list, see sequoia

Author(s)

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

See Also

Ped_HSg5, LH_HSg5

Examples

```
## Not run:
# this output was created as follows:
Geno <- SimGeno(Ped = Ped_HSg5, nSnp = 200)
SeqOUT_HSg5 <- sequoia(GenoM = Geno, LifeHistData = LH_HSg5, Module = "ped",
                       Err = 0.005)

## End(Not run)
# some ways to inspect the output; see vignette for more info:
names(SeqOUT_HSg5)
SeqOUT_HSg5$Specs
SummarySeq(SeqOUT_HSg5)
```

---

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",
  Err = 1e-04,
  Tfilter = -2,
  Tassign = 0.5,
  MaxSibshipSize = 100,
  DummyPrefix = c("F", "M"),
  Complex = "full",
  Herm = "no",
  UseAge = "yes",
  args.AP = list(Flatten = NULL, Smooth = TRUE),
  mtSame = NULL,
  CalcLLR = FALSE,
  quiet = FALSE,
```

```
  Plot = NULL,
  StrictGenoCheck = TRUE,
  ErrFlavour = "version2.9"
)
```

Arguments

GenoM	numeric matrix with genotype data: One row per individual, one column per SNP, coded as 0, 1, 2, missing values as a negative number or NA. Row names must be individual IDs, column names are ignored. You can reformat data with GenoConvert, or use other packages to get it into a genlight object and then use as.matrix.
LifeHistData	data.frame with up to 6 columns: **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 number. **BY.min** minimum birth year, only used if BirthYear is missing **BY.max** maximum birth year, only used if BirthYear is missing **Year.last** Last year in which individual could have had offspring. Can e.g. in mammals be the year before death for females, and year after death for males. "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, and only integers are used. 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$Specs is provided, all input parameters with the same name as its items are ignored, except Module.
Module	one of **pre** Only input check, return SeqList$Specs **dup** Also check for duplicate genotypes **par** Also perform parentage assignment (genotyped parents to genotyped offspring) **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.
Err	assumed per-locus genotyping error rate, as a single number, or a length 3 vector with P(hom|hom), P(het|hom), P(hom|het), or a 3x3 matrix. See details below. The error rate is presumed constant across SNPs, and missingness is presumed random with respect to actual genotype. Using Err >5% is not recommended, and Err >10% strongly discouraged. See Err_RADseq to convert

per-allele rates at homozygous and heterozygous sites to the required length-3 vector, and ErrToM for further genotyping error details.

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" (only use age differences for filtering), 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, e.g. 'Discrete' (non-overlapping generations), 'MinAgeParent', 'MaxAgeParent'.

mtSame — matrix indicating whether individuals (might) have the same mitochondrial haplotype (1), and may thus be matrilineal relatives, or not (0). This potentially be useful to distinguish between maternal and paternal half siblings when few parents are genotyped, or between maternal and paternal parents when their sex(role) is unknown. Row names and column names should match IDs in 'GenoM'. Not all individuals need to be included and order is not important. For details see the mtDNA vignette.

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().

StrictGenoCheck — Automatically exclude any individuals genotyped for <5 the unavoidable default up to version 2.4.1. Otherwise only excluded are (very nearly) monomorphic SNPs, SNPs scored for fewer than 2 individuals, and individuals scored for fewer than 2 SNPs.

`ErrFlavour` function that takes `Err` (single number) as input, and returns a length 3 vector or 3x3 matrix, or choose from inbuilt options 'version2.9', 'version2.0', 'version1.3', or 'version1.1', referring to the sequoia version in which they were the default. Ignored if `Err` is a vector or matrix.

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.

Full explanation of the various options and interpretation of the output is provided in the vignettes and on the package website, https://jiscah.github.io/index.html .

Value

A list with some or all of the following components, depending on `Module`. All input except `GenoM` is included in the output.

`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.

`args.AP` (input) arguments used to specify age prior matrix. If a custom ageprior was provided via `SeqList$AgePrior`, this matrix is returned instead

`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` (input) 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	(input) Dataframe with sex and birth year data. All missing birth years are coded as '-999', all missing sex as '3'.
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
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. This includes the maximum OH for potential (parent-)parent-offspring pairs (trios), which is calculated by CalcMaxMismatch
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.

Genotyping error rate

The genotyping error rate `Err` can be specified three different ways:

- A single number, which is combined with `ErrFlavour` by `ErrToM` to create a length 3 vector (next item). By default (`ErrFlavour` = 'version2.9'), P(hom|hom)=$(E/2)^2$, P(het|hom)=$E-(E/2)^2$, P(hom|het)=$E/2$.
- a length 3 vector (NEW from version 2.6), with the probabilities to observe a actual homozygote as the other homozygote (hom|hom), to observe a homozygote as heterozygote (het|hom), and to observe an actual heterozygote as homozygote (hom|het). This assumes that the two alleles are equivalent with respect to genotyping errors, i.e. $P(AA|aa) = P(aa|AA)$, $P(aa|Aa)=P(AA|Aa)$, and $P(aA|aa)=P(aA|AA)$.
- a 3x3 matrix, with the probabilities of observed genotype (columns) conditional on actual genotype (rows). Only needed when the assumption in the previous item does not hold. See `ErrToM` for details.

(Too) Few Assignments?

Possibly `Err` is much lower than the actual genotyping error rate.

Alternatively, a true parent will not be assigned when it is:

- unclear who is the parent and who the offspring, due to unknown birth year for one or both individuals
- unclear whether the parent is the father or mother
- unclear if it is a parent or e.g. full sibling or grandparent, due to insufficient genetic data

And true half-siblings will not be clustered when it is:

- unclear if they are maternal or paternal half-siblings
- unclear if they are half-siblings, full avuncular, or grand-parental
- unclear what type of relatives they are due to insufficient genetic data

All pairs of non-assigned but likely/definitely relatives can be found with `GetMaybeRel` (run without conditioning on a pedigree). For a method to do pairwise 'assignments', see https://jiscah.github.io/articles/pairLL_classificat ; for further information, see the vignette.

If you already had a partial pedigree, running `CalcOHLLR` or `CalcParentProbs` on it with the new SNP data and various parameter value combinations may be informative.

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.

Website

https://jiscah.github.io/

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 and PlotAgePrior to visualise those,
- SummarySeq and PlotPropAssigned to visualise results,
- GetMaybeRel to find pairs of potential relatives,
- GetRelM to turn a pedigree into pairwise relationships,
- CountOH, CalcPairLL and LLtoProb for specified pairs of individuals respectively count Opposing Homozygous SNPs, calculate likelihoods of various relationships, and transform those likelihoods to probabilities,
- CalcOHLLR to count Opposing Homozygous SNPs and calculate LLR for all parent-offspring pairs in any pedigree,
- CalcParentProbs to calculate assignment probabilities (instead of LLRs) in any pedigree,
- CalcBYprobs to estimate birth years,
- PedCompare and ComparePairs to compare two pedigrees,
- EstConf to estimate assignment errors,
- writeSeq to save sequoia output as text or excel files,
- vignette("sequoia") for detailed manual & FAQ.

Examples

```
# ===  EXAMPLE 1: simulated data  ===
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:
# (note: this can be rather time consuming)
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=FALSE))
# or for very large datasets:
GenoM <- data.table::fread('MyGenoData.txt') %>% as.matrix(rownames=1)

LHdata <- read.table("LifeHistoryData.txt", header=T) # ID-Sex-birthyear
SeqOUT <- sequoia(GenoM, LHdata, Err=0.005)
SummarySeq(SeqOUT)

SeqOUT$notes <- "Trial run on cleaned data"  # add notes for future reference
saveRDS(SeqOUT, file="sequoia_output_42.RDS")  # save to R-specific file
writeSeq(SeqOUT, folder="sequoia_output")  # save to several plain 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 missingness, genotyping errors, and non-genotyped parents.

Usage

```
SimGeno(
  Pedigree,
  nSnp = 400,
  ParMis = c(0, 0),
  MAF = 0.3,
  CallRate = 0.99,
  SnpError = 5e-04,
  ErrorFV = function(E) c((E/2)^2, E - (E/2)^2, E/2),
  ErrorFM = NULL,
  ReturnStats = FALSE,
  quiet = FALSE
)
```

Arguments

Pedigree	dataframe, pedigree with the first three columns being id - dam - sire, additional columns are ignored.
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. NOTE: default changed from 0.4 (up to version 2.8.5) to 0 (from version 2.9).
MAF	either a single number with minimum minor allele frequency, and allele frequencies will be sampled uniformly between this minimum ('min_maf') and 1-min_maf (NOTE: between min_maf and 0.5 up to version 2.11) 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. Occasionally, alleles may get fixed, and all founders carrying the allele simulated as non-genotyped, resulting in a sample MAF of 0.
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 as parent) not included in this vector are presumed non-genotyped.
SnpError	either a single value which will be combined with ErrorFV, or a length 3 vector with probabilities (observed given actual) hom|other hom, het|hom, and hom|het; OR a vector or 3XnSnp matrix with the genotyping error rate(s) for each SNP.

ErrorFV	function taking the error rate (scalar) as argument and returning a length 3 vector with hom->other hom, hom->het, het->hom. May be an 'ErrFlavour', e.g. 'version2.9'.
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). See below for details. To use, set ErrorFV = NULL
ReturnStats	in addition to the genotype matrix, return the input parameters and mean & quantiles of MAF, error rate and call rates.
quiet	suppress messages.

Details

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.

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

Genotyping errors

If SnpError is a length 3 vector, genotyping errors are generated following a length 3 vector with probabilities that 1) an actual homozygote is observed as the other homozygote, 2) an actual homozygote is observed as a heterozygote, and 3) an heterozygote is observed as an homozygote. The only assumption made is that the two alleles can be treated equally, i.e. observing actual allele A as a is as likely as observing actual a as A.

If SnpError is a single value, by default this 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 $(\texttt{SnpError}/2)^2$. How this single value is rendered into a 3x3 error matrix is fully flexible and specified via ErrorFM; see link{ErrToM} for details.

The default values of SnpError=5e-4 and ErrorFM='version2.9' correspond to the length 3 vector c((5e-4/2)^2, 5e-4*(1-5e-4/2), 5e-4/2).

A beta-distribution is used to simulate variation in the error rate between SNPs, the shape parameter of this distribution can be specified via MkGenoErrors. It is also possible to specify the error rate per SNP.

Call Rate

Variation in call rates across SNPs is assumed to follow a highly skewed (beta) distribution, with many SNPs 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. 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.

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; ErrToM for more information about genotyping error patterns.

Examples

```
Geno_A <- SimGeno(Pedigree = Ped_griffin, nSnp=200, ParMis=c(0.1, 0.6),
                  MAF = 0.25, SnpError = 0.001)

Geno_B <- SimGeno(Pedigree = Ped_HSg5, nSnp = 100, ParMis = 0.2,
                 SnpError = c(0.01, 0.04, 0.1))

Geno_C <- SimGeno(Pedigree = Ped_griffin, nSnp=200, ParMis=0, CallRate=0.6,
                  SnpError = 0.05, ErrorFV=function(E) c(E/10, E/10, E))

# genotype matrix with duplicated samples:
```

```
Dups_grif <- data.frame(ID1 = c('i006_2001_M', 'i021_2002_M', 'i064_2004_F'))
Dups_grif$ID2 <- paste0(Dups_grif$ID1, '_2')
Err <- c(0.01, 0.04, 0.1)
Geno_act <- SimGeno(Ped_griffin, nSnp=500, ParMis=0, CallRate=1, SnpError=0)
Geno_sim <- MkGenoErrors(Geno_act, SnpError=Err, CallRate=0.99)
Geno_dups <- MkGenoErrors(Geno_act[Dups_grif$ID1, ], SnpError=Err,
                          CallRate=0.99)
rownames(Geno_dups) <- Dups_grif$ID2
Geno_sim <- rbind(Geno_sim, Geno_dups)

## Not run:
# write simulated genotypes to a file, e.g. for use by PLINK:
GenoConvert(Geno_A, InFormat='seq', OutFormat='ped', OutFile = sim_genotypes)

## End(Not run)
```

SimGeno_example *Example genotype file: 'HSg5'*

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

LH_HSg5, SimGeno

SnpStats *SNP Summary Statistics*

Description

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

Usage

```
SnpStats(
  GenoM,
  Pedigree = NULL,
  Duplicates = NULL,
  Plot = TRUE,
  quiet = TRUE,
  calc_HWE = TRUE,
  ErrFlavour
)
```

Arguments

GenoM	genotype matrix, in sequoia's format: 1 column per SNP, 1 row per individual, genotypes coded as 0/1/2/-9, and row names giving individual IDs.
Pedigree	dataframe with 3 columns: ID - parent1 - parent2. Additional columns and non-genotyped individuals are ignored. Used to count Mendelian errors per SNP and (poorly) estimate the error rate.
Duplicates	dataframe with pairs of duplicated samples
Plot	logical, show histograms of the results?
quiet	logical, suppress messages?
calc_HWE	logical, calculate chi-square test for Hardy-Weinberg equilibrium? Can be relatively time consuming for large datasets.
ErrFlavour	DEPRECATED AND IGNORED. Was used to estimate Err.hat

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
`HWE.p`	p-value from chi-square test for Hardy-Weinberg equilibrium

When a Pedigree is provided, there are 8 additional columns:

`n.dam`, `n.sire`, `n.pair`	Number of dams, sires, parent-pairs successfully genotyped for the SNP
`OHdam`, `OHsire`	Count of number of opposing homozygous cases
`MEpair`	Count of Mendelian errors, includes opposing homozygous cases when only one parent is genotyped
`n.dups`, `n.diff`	Number of duplicate pairs successfully genotyped for the SNP; number of differences. The latter does not count cases where one duplicate is not successfully genotyped at the SNP

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

```
Genotypes <- SimGeno(Ped_HSg5, nSnp=100, CallRate = runif(100, 0.5, 0.8),
                     SnpError = 0.05)
SnpStats(Genotypes)   # only plots; data is returned invisibly
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. All ids in 'PedigreePar', and only those, are presumed genotyped.
Pedigree	dataframe, pedigree with the first three columns being id - dam - sire. Column names are ignored, as are additional columns, except for columns OHdam, OHsire, MEpair, LLRdam, LLRsire, LLRpair (plotting only).
DumPrefix	character vector with prefixes for dummy dams (mothers) and sires (fathers), used to distinguish between dummies and non-dummies. Will be read from SeqList's 'Specs' if provided.
SNPd	character vector with ids of SNP genotyped individuals. Only used when Pedigree is provided instead of SeqList, to distinguish between genetically assigned parents and 'observed' parents (e.g. observed in the field, or assigned previously using microsatellites). If NULL (the default), all parents are presumed observed.
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).

Details

The list with results is returned invisible

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	an array with the number of assigned parents, split by:

- offspringCat: Genotyped, Dummy, or Observed* (*: only when Pedigree is provided rather than SeqList, for ids which are not listed in SNPd and do not conform to DumPrefix + number (i.e. (almost) all individuals when SNPd = NULL, the default).
- offspringSex: Female, Male, Unknown, or Herm* (*: hermaphrodite, only if any individuals occur as both dam and sire). Based only on whether an individual occurs as Dam or Sire.
- parentSex: Dam or Sire
- parentCat: Genotyped, Dummy, Observed*, or None (*: as for offspringCat)

GPCount — an array with the number of assigned grandparents, split by:

- offspringCat: Genotyped, Dummy, Observed*, or All
- grandparent kind: maternal grandmothers (MGM), maternal grandfathers (MGF), paternal grandmothers (PGM), paternal grandfathers (PGF)
- grandparentCat: Genotyped, Dummy, Observed*, or None

SibSize — a list with elements 'mat' (maternal half + full siblings), 'pat' (paternal half + full siblings), and 'full' (full siblings). Each 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

PlotSeqSum to plot the output of this function; PlotPropAssigned for just a barplot of the proportion of individuals with genotyped/dummy/observed/no assigned dam/sire.

Examples

```
SummarySeq(Ped_griffin)
sumry_grif <- SummarySeq(SeqOUT_griffin, Panels=c("G.parents", "OH"))
sumry_grif$PedSummary

# invisible results (nothing printed) when no output object is specified:
SummarySeq(SeqOUT_griffin, Panels=c("LLR"))
```

---

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 **openxlsx**).

Usage

```
writeSeq(
  SeqList,
  GenoM = NULL,
  MaybeRel = NULL,
  PedComp = NULL,
  OutFormat = "txt",
  folder = "Sequoia-OUT",
  file = "Sequoia-OUT.xlsx",
  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'`.
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 sheet to the excel file:
library(openxlsx)
wb <- loadWorkbook("MyFile.xlsx")
addWorksheet(wb, sheetName = "ExtraData")
writeData(wb, sheet = "ExtraData", MyData, rowNames=FALSE)
saveWorkbook(wb, "MyFile.xlsx", overwrite=TRUE, returnValue=TRUE)

# or: (package requires java & is trickier to install)
xlsx::write.xlsx(MyData, file = "MyFile.xlsx", sheetName="ExtraData",
      col.names=TRUE, row.names=FALSE, append=TRUE, showNA=FALSE)

## End(Not run)
```

Index

∗ **datasets**
  Conf_griffin, 24
  FieldMums_griffin, 35
  Geno_griffin, 37
  Geno_HSg5, 37
  Inherit_patterns, 52
  LH_griffin, 53
  LH_HSg5, 54
  MaybeRel_griffin, 61
  Ped_griffin, 63
  Ped_HSg5, 63
  SeqOUT_griffin, 77
  SeqOUT_HSg5, 77
  SimGeno_example, 89
∗ **inherit**
  Inherit_patterns, 52
∗ **sequoia**
  Conf_griffin, 24
  FieldMums_griffin, 35
  Geno_griffin, 37
  Geno_HSg5, 37
  Inherit_patterns, 52
  LH_griffin, 53
  LH_HSg5, 54
  MaybeRel_griffin, 61
  Ped_griffin, 63
  Ped_HSg5, 63
  SeqOUT_griffin, 77
  SeqOUT_HSg5, 77
  SimGeno_example, 89

barplot, *74*

CalcBYprobs, 3, *81*, *84*
CalcMaxMismatch, 4, *8*, *26*, *82*
CalcOHLLR, 6, *13*, *20*, *26*, *67*, *80*, *83, 84*, *91*
CalcPairLL, *9*, 10, *16, 17*, *26*, *45*, *49*, *72*, *82*, *84*
CalcParentProbs, *15*, 16, *83, 84*
CalcRped, 18
CheckGeno, *9*, 19, *40, 41*, *43*, *84*, *91*
ComparePairs, 20, *26, 27*, *52*, *67*, *84*
Conf_griffin, 24
CountOH, 25, *84*

DyadCompare, *22*, 26

Err_RADseq, *7*, *12*, 27, *47*, *79*
ErrToM, *5*, *7*, *12*, *27*, 28, *47*, *80*, *83*, *88*
EstConf, *23, 24*, 31, *67*, *84*, *88*

FieldMums_griffin, 35
FindFamilies, 36, *44*, *70*
fread, *39*

genlight, *39*
Geno_griffin, *24*, 37, *61*, *77*
Geno_HSg5, 37, *77*
GenoConvert, *7*, *9*, *11*, *16*, *25*, 38, *46*, *55*, *70*, *79*, *84*, *91*
GetAncestors, *37*, 41, *44*
getAssignCat, *8, 9*, *13*, 42, *65–67*
GetDescendants, *37*, 43, *44*
getGenerations, *37*, 44, *69*, *75*
GetLLRAge, 45
GetMaybeRel, *15*, *21*, *36*, 46, *50*, *61*, *83, 84*, *94*
GetRelM, *15*, *23*, *49*, 50, *75*, *84*

Inherit_patterns, 52
invisible, *20*, *75*, *92*

kinship, *18*, *69*

layout, *71*
LH_griffin, 53, *63*
LH_HSg5, *38*, 54, *64*, *78*, *89*
LHConvert, *41*, 54, *70*
LLtoProb, *15*, *17*, 56, *84*

MakeAgePrior, *3, 4*, *7*, *12*, *14*, *47*, 57, *71*, *80, 81*, *84*

MaybeRel_griffin, 61
MkGenoErrors, 61, *88*

paste, *39*
Ped_griffin, *24*, *36, 37*, *53*, *61*, 63, *77*
Ped_HSg5, *37*, *54*, 63, *63*, *78*, *89*
PedCompare, *21*, *23*, *27*, *31*, *33*, *35*, *42, 43*, 64, *73*, *84*, *94*
PedPolish, *9*, *36*, *51*, 68
PedStripFID, *39*, *55*, 70
PlotAgePrior, *60*, 71, *84*
PlotPairLL, *13*, *15*, 71
PlotPedComp, 73
PlotPropAssigned, 73, *84*, *93*
PlotRelPairs, *23*, *52*, 74
PlotSeqSum, 76, *93*

read.table, *40*
readLines, *39, 40*

SeqOUT_griffin, *24*, *36*, *53*, *61*, *63*, 77
SeqOUT_HSg5, *38*, 77
sequoia, *5*, *7–9*, *12*, *19*, *24*, *26*, *31*, *33*, *36*, *42, 43*, *45–47*, *49*, *54, 55*, *60*, *63, 64*, *77*, 78, *92*, *94*
SimGeno, *31*, *33*, *37, 38*, *53*, *84*, 86, *89*
SimGeno_example, *64*, 89
SnpStats, *6*, *19, 20*, *41*, *80*, *84*, 90
strsplit, *39, 40*
SummarySeq, *9*, *71*, *74–76*, *80*, *84*, 91
system.time, *32*

write.table, *39*
writeColumns, 93, *95*
writeSeq, *84*, 94