Package 'polysat'

August 23, 2022

Version 1.7-7

Date 2022-08-20

Title Tools for Polyploid Microsatellite Analysis

Imports methods, stats, utils, grDevices, graphics, Rcpp

Suggests ade4, adegenet, ape

LinkingTo Rcpp

Description A collection of tools to handle microsatellite data of any ploidy (and samples of mixed ploidy) where allele copy number is not known in partially heterozygous genotypes. It can import and export data in ABI 'GeneMapper', 'Structure', 'ATetra', 'Tetrasat'/'Tetra', 'GenoDive', 'SPAGeDi', 'POPDIST', 'STRand', and binary presence/absence formats. It can calculate pairwise distances between individuals using a stepwise mutation model or infinite alleles model, with or without taking ploidies and allele frequencies into account. These distances can be used for the calculation of clonal diversity statistics or used for further analysis in R. Allelic diversity statistics and Polymorphic Information Content are also available. polysat can assist the user in estimating the ploidy of samples, and it can estimate allele frequencies in populations, calculate pairwise or global differentiation statistics based on those frequencies, and export allele frequencies to 'SPAGeDi' and 'adegenet'. Functions are also included for assigning alleles to isoloci in cases where one pair of microsatellite primers amplifies alleles from two or more independently segregating isoloci. polysat is described by Clark and Jasieniuk (2011) <doi:10.1111/j.1755-0998.2011.02985.x> and Clark and Schreier (2017) <doi:10.1111/1755-0998.12639>.

License GPL-2

URL https://github.com/lvclark/polysat/wiki

NeedsCompilation yes

Author Lindsay V. Clark [aut, cre] (<https://orcid.org/0000-0002-3881-9252>), Alistair J. Hall [ctb] (<https://orcid.org/0000-0001-9293-8909>), Handunnethi Nihal de Silva [ctb], Tyler William Smith [ctb] (<https://orcid.org/0000-0001-7683-2653>)

Maintainer Lindsay V. Clark <Lindsay.Clark@seattlechildrens.org>

Repository CRAN

Date/Publication 2022-08-23 14:10:02 UTC

R topics documented:

`Accessors` *Accessor and Replacement Functions for "gendata" Objects*

Description

The accessor functions return information that is contained, either directly or indirectly, in the slots of a `gendata` object. The replacement functions alter information in one or more slots as appropriate.

Usage

```
Samples(object, populations, ploidies)
Samples(object) <- value
Loci(object, usatnts, ploidies)
Loci(object) <- value
PopInfo(object)
PopInfo(object) <- value
PopNames(object)
PopNames(object) <- value
PopNum(object, popname)
PopNum(object, popname) <- value
Ploidies(object, samples, loci)
Ploidies(object) <- value
Usatnts(object)
Usatnts(object) <- value
Description(object)
Description(object) <- value
Missing(object)
Missing(object) <- value
```

```
Present(object)
Present(object) <- value
Absent(object)
Absent(object) <- value
Genotype(object, sample, locus)
Genotype(object, sample, locus) <- value
Genotypes(object, samples = Samples(object), loci = Loci(object))
Genotypes(object, samples = Samples(object), loci = Loci(object)) <- value
```

Arguments

object	An object of the class gendata or one of its subclasses.
populations	A character or numeric vector indicating from which populations to return samples. (optional)
ploidies	A numeric vector indicating ploidies, if only samples or loci with a certain ploidy should be returned. (optional)
sample	A character string or number indicating the name or number of the sample whose genotype should be returned.
locus	A character string or number indicating the name or number of the locus whose genotype should be returned.
samples	A character or numeric vector indicating samples for which to return genotypes or ploidies. (optional)
loci	A character or numeric vector indicating loci for which to return genotypes or ploidies. (optional)
usatnts	A numeric vector indicating microsatellite repeat lengths, where only loci of those repeat lengths should be returned. (optional)
popname	Chacter string or vector. The name(s) of the population(s) for which to retrieve or replace the corresponding PopInfo number(s). The replacement function should only be used for one population at a time.
value	• For Samples: a character vector of sample names. • For Loci: a character vector of locus names. • For PopInfo: A numeric vector (integer or can be coerced to integer) indicating the population identities of samples. • For PopNames: A character vector indicating the names of populations. • For PopNum: A number (integer or can be coerced to integer) that should be the new population number associated with popname. • For Ploidies: A numeric vector or matrix (integer or can be coerced to integer) indicating the ploidy of each sample, locus, and/or the dataset. See reformatPloidies and "ploidysuper". • For Usatnts: A numeric vector (integer or can be coerced to integer) indicating the repeat type of each microsatellite locus. Dinucleotide repeats should be represented with 2, trinucleotide repeat with 3, and so on. If the alleles for a given locus are already stored in terms of repeat number rather than fragment length in nucleotides, the Usatnts value for that locus should be 1.

- For Description: A character string or character vector describing the dataset.
- For Missing: A symbol (usually an integer) to be used to indicate missing data.
- For Present: A symbol (usually an integer) to be used to indicate the presence of an allele.
- For Absent: A symbol (usually an integer) to be used to indicate the absence of an allele.
- For Genotype: a vector of alleles, if the object is of class genambig.
- For Genotypes: A list of vectors (genotypes), of the same dimensionality as c(length(samples), length(loci)), if the object is of class genambig. If the object is of class genbinary, value should be a matrix, with column names of the form "locus.allele". See Genotypes<-,genbinary-method for more information.

Details

Samples<- and Loci<- can only be used to change sample and locus names, not to add or remove samples and loci from the dataset.

For slots that require integer values, numerical values used in replacement functions will be coerced to integers. The replacement functions also ensure that all slots remain properly indexed.

The Missing<- function finds any genotypes with the old missing data symbol and changes them to the new missing data symbol, then assigns the new symbol to the slot that indicates what the missing data symbol is. Present<- and Absent<- work similarly for the genbinary class.

The Genotype access and replacement functions deal with individual genotypes, which are vectors in the genambig class. The Genotypes access and replacement functions deal with lists of genotypes.

The PopInfo<- replacement function also adds elements to PopNames(object) if necessary in order to have names for all of the populations. These will be of the form "Pop" followed by the population number, and can be later edited using PopNames<-.

The PopNum<- replacement function first finds all samples in the population popname, and replaces the number in PopInfo(object) for those samples with value. It then inserts NA into the original PopNames slot that contained popname, and inserts popname into PopNames(object)[value]. If this results in two populations being merged into one, a message is printed to the console.

Value

PopInfo, PopNames, Missing, Description, Usatnts, Ploidies and Genotypes simply return the contents of the slots of the same names (or in the case of Ploidies, object@Ploidies@pld is returned). Samples and Loci return character vectors taken from the names of other slots (PopInfo and Usatnts, respectively; the initialization and replacement methods ensure that these slots are always named according to samples and loci). PopNum returns an integer vector indicating the population number(s) of the population(s) named in popname. Genotype returns a single genotype for a given sample and locus, which is a vector whose exact form will depend on the class of object.

Author(s)

Lindsay V. Clark

See Also

deleteSamples, deleteLoci, viewGenotypes, editGenotypes, isMissing, estimatePloidy, merge,gendata,gendata-method, gendata

Examples

```
# create a new genambig (subclass of gendata) object to manipulate
mygen <- new("genambig", samples=c("a", "b", "c"), loci=c("locG",
"locH"))

# retrieve the sample and locus names
Samples(mygen)
Loci(mygen)

# change some of the sample and locus names
Loci(mygen) <- c("lG", "lH")
Samples(mygen)[2] <- "b1"

# describe the dataset
Description(mygen) <- "Example dataset for documentation."

# name some populations and assign samples to them
PopNames(mygen) <- c("PopL", "PopK")
PopInfo(mygen) <- c(1,1,2)
# now we can retrieve samples by population
Samples(mygen, populations="PopL")
# we can also adjust the numbers if we want to make them
# match another dataset
PopNum(mygen, "PopK") <- 3
PopNames(mygen)
PopInfo(mygen)
# change the population identity of just one sample
PopInfo(mygen)["b1"] <- 3

# indicate that both loci are dinucleotide repeats
Usatnts(mygen) <- c(2,2)

# indicate that all samples are tetraploid
Ploidies(mygen) <- 4
# or
Ploidies(mygen) <- rep(4, times = length(Samples(mygen)) * length(Loci(mygen)))
# actually, one sample is triploid
Ploidies(mygen)["c",] <- 3
# view ploidies
Ploidies(mygen)

# view the genotype array as it currently is: filled with missing
# values
Genotypes(mygen)
# fill in the genotypes
Genotypes(mygen, loci="lG") <- list(c(120, 124, 130, 136), c(122, 120),
                                    c(128, 130, 134))
```

```
Genotypes(mygen, loci="1H") <- list(c(200, 202, 210), c(206, 208, 210,
                                                        214),
                                    c(208))
# genotypes can also be edited or retrieved by sample
Genotypes(mygen, samples="a")
# fix a single genotype
Genotype(mygen, "a", "1H") <- c(200, 204, 210)
# retrieve a single genotype
Genotype(mygen, "c", "1G")

# change a genotype to being missing
Genotype(mygen, "c", "1H") <- Missing(mygen)
# show the current missing data symbol
Missing(mygen)
# an example of genotypes where one contains the missing data symbol
Genotypes(mygen, samples="c")
# change the missing data symbol
Missing(mygen) <- as.integer(-1)
# now look at the genotypes
Genotypes(mygen, samples="c")
```

`alleleCorrelations` *Assign Alleles to Isoloci Based on Distribution of Genotypes*

Description

Where a single locus represents two or more independent isoloci (as in an allopolyploid, or a diploidized autopolyploid), these two functions can be used in sequence to assign alleles to isoloci. `alleleCorrelations` uses K-means and UPGMA clustering of pairwise p-values from Fisher's exact test to make initial groupings of alleles into putative isoloci. `testAlGroups` is then used to check those groupings against individual genotypes, and adjust the assignments if necessary.

Usage

```
alleleCorrelations(object, samples = Samples(object), locus = 1,
                   alpha = 0.05, n.subgen = 2, n.start = 50)

testAlGroups(object, fisherResults, SGploidy=2, samples=Samples(object),
             null.weight=0.5, tolerance=0.05, swap = TRUE,
             R = 100, rho = 0.95, T0 = 1, maxreps = 100)
```

Arguments

`object`	A "genambig" or "genbinary" object containing the data to analyze.
`samples`	An optional character or numeric vector indicating which samples to analyze.
`locus`	A single character string or integer indicating which locus to analyze.

alpha	The significance threshold, before multiple correction, for determining whether two alleles are significantly correlated.
n.subgen	The number of subgenomes (number of isoloci) for this locus. This would be 2 for an allotetraploid or 3] for an allohexaploid. For an allo-octoploid, the value would be 2 if there were two tetraploid subgenomes, or 4 if there were four diploid subgenomes.
n.start	Integer, passed directly to the nstart argument of the R base function kmeans. Lowering this number will speed up computation time, whereas increasing it will improve the probability of finding the correct allele assignments. The default value of 50 should work well in most cases.
fisherResults	A list output from alleleCorrelations.
SGploidy	The ploidy of each subgenome (each isolocus). This is 2 for an allotetraploid, an allohexaploid, or an allo-octoploid with four tetraploid subgenomes, or 4 for an allo-octoploid with two tetraploid genomes.
null.weight	Numeric, indicating how genotypes with potential null alleles should be counted when looking for signs of homoplasy. null.weight should be 0 if null alleles are expected to be common, and 1 if there are no null alleles in the dataset. The default of 0.5 was chosen to reflect the fact that the presence of null alleles is generally unknown.
tolerance	The proportion of genotypes that are allowed to be in disagreement with the allele assignments. This is the proportion of genotypes that are expected to have meiotic error or scoring error.
swap	Boolean indicating whether or not to use the allele swapping algorithm before checking for homoplasy. TRUE will yield more accurate results in most cases, but FALSE may be preferable for loci with null or homoplasious alleles at high frequency.
R	Simulated annealing parameter for the allele swapping algorithm. Indicates how many swaps to attempt in each rep (*i.e.* how many swaps to attempt before changing the temperature).
rho	Simulated annealing parameter for the allele swapping algorithm. Factor by which to reduce the temperature at the end of each rep.
T0	Simulated annealing parameter for the allele swapping algorithm. Starting temperature.
maxreps	Simulated annealing parameter for the allele swapping algorithm. Maximum number of reps if convergence is not achieved.

Details

These functions implement a novel methodology, introduced in **polysat** version 1.4 and updated in version 1.6, for cases where one pair of microsatellite primers amplifies alleles at two or more independently-segregating loci (referred to here as isoloci). This is not typically the case with new autopolyploids, in which all copies of a locus have equal chances of pairing with each other at meiosis. It is, however, frequently the case with allopolyploids, in which there are two homeologous subgenomes that do not pair (or infrequently pair) at meiosis, or ancient autopolyploids, in which duplicated chromosomes have diverged to the point of no longer pairing at meiosis.

Within the two functions there are four major steps:

1. `alleleCorrelations` checks to see if there are any alleles that are present in every genotype in the dataset. Such invariable alleles are assumed to be fixed at one isolocus (which is not necessarily true, but may be corrected by `testAlGroups` in steps 4 and 5). If present, each invariable allele is assigned to its own isolocus. If there are more invariable alleles than isoloci, the function throws an error. If only one isolocus remains, all remaining (variable) alleles are assigned to that isolocus. If there are as many invariable alleles as isoloci, all remaining (variable) alleles are assigned to all isoloci (*i.e.* they are considered homoplasious because they cannot be assigned).
2. If, after step 1, two or more isoloci remain without alleles assigned to them, correlations between alleles are tested by `alleleCorrelations`. The dataset is converted to `"genbinary"` if not already in that format, and a Fisher's exact test, with negative association (odds ratio being less than one) as the alternative hypothesis, is performed between each pair of columns (alleles) in the genotype matrix. The p-value of this test between each pair of alleles is stored in a square matrix, and zeros are inserted into the diagonal of the matrix. K-means clustering and UPGMA are then performed on the square matrix of p-values, and the clusters that are produced represent initial assignments of alleles to isoloci.
3. The output of `alleleCorrelations` is then passed to `testAlGroups`. If the results of K-means clustering and UPGMA were not identical, `testAlGroups` checks both sets of assignments against all genotypes in the dataset. For a genotype to be consistent with a set of assignments, it should have at least one allele and no more than `SGploidy` alleles belonging to each isolocus. The set of assignments that is consistent with the greatest number of genotypes is chosen, or in the case of a tie, the set of assignments produced by K-means clustering.
4. If `swap = TRUE` and the assignments chosen in the previous step are inconsistent with some genotypes, `testAlGroups` attempts to swap the isoloci of single alleles, using a simulated annealing (Bertsimas and Tsitsiklis 1993) algorithm to search for a new set of assignments that is consistent with as many genotypes as possible. At each step, an allele is chosen at random to be moved to a different isolocus (which is also chosen at random if there are more than two isoloci). If the new set of allele assignments is consistent with an equal or greater number of genotypes than the previous set of assignments, the new set is retained. If the new set is consistent with fewer genotypes than the old set, there is a small probability of retaining the new set, dependent on how much worse the new set of assignments is and what the current "temperature" of the algorithm is. After `R` allele swapping attempts, the temperature is lowered, reducing the probability of retaining a set of allele assignments that is worse than the previous set. A new rep of `R` swapping attempts then begins. If a set of allele assignments is found that is consistent with all genotypes, the algorithm stops immediately. Otherwise it stops if no changes are made during an entire rep of `R` swap attempts, or if `maxreps` reps are performed.
5. `testAlGroups` then checks through all genotypes to look for signs of homoplasy, meaning single alleles that should be assigned to more than one isolocus. For each genotype, there should be no more than `SGploidy` alleles assigned to each isolocus. Additionally, if there are no null alleles, each genotype should have at least one allele belonging to each isolocus. Each time a genotype is encountered that does not meet these criteria, the a score is increased for all alleles that might be homoplasious. (The second criterion is not checked if `null.weight = 0`.) This score starts at zero and is increased by 1 if there are too many alleles per isolocus or by `null.weight` if an isolocus has no alleles. Once all genotypes have been checked, the allele with the highest score is considered to be homoplasious and is added to the other isolocus. (In a hexaploid or higher, which isolocus the allele is added to depends on the genotypes that were found to be inconsistent with the allele assignments, and which isolocus or isoloci the

allele could have belonged to in order to fix the assignment.) Allele scores are reset to zero and all alleles are then checked again with the new set of allele assignments. The process is repeated until the proportion of genotypes that are inconsistent with the allele assignments is at or below `tolerance`.

Value

Both functions return lists. For `alleleCorrelations`:

`locus` The name of the locus that was analyzed.

`clustering.method`
: The method that was ultimately used to produce `value$Kmeans.groups` and `value$UPGMA.groups`. Either `"K-means and UPGMA"` or `"fixed alleles"`.

`significant.neg`
: Square matrix of logical values indicating whether there was significant negative correlation between each pair of alleles, after multiple testing correction by Holm-Bonferroni.

`significant.pos`
: Square matrix of logical values indicating whether there was significant positive correlation between each pair of alleles, after multiple testing correction by Holm-Bonferroni.

`p.values.neg` Square matrix of p-values from Fisher's exact test for negative correlation between each pair of alleles.

`p.values.pos` Square matrix of p-values from Fisher's exact test for positive correlation between each pair of alleles.

`odds.ratio` Square matrix of the odds ratio estimate from Fisher's exact test for each pair of alleles.

`Kmeans.groups` Matrix with `n.subgen` rows, and as many columns as there are alleles in the dataset. `1` indicates that a given allele belongs to a given isolocus, and `0` indicates that it does not. These are the groupings determined by K-means clustering.

`UPGMA.groups` Matrix in the same format as `value$Kmeans.groups`, showing groupings determined by UPGMA.

`heatmap.dist` Square matrix like `value$p.values.neg` but with zeros inserted on the diagonal. This is the matrix that was used for K-means clustering and UPGMA. This matrix can be passed to the `heatmap` function in R to visualize the clusters.

`totss` Total sums of squares output from K-means clustering.

`betweenss` Sums of squares between clusters output from K-means clustering. `value$betweenss/value$totss` can be used as an indication of clustering quality.

`gentable` The table indicating presence/absence of each allele in each genotype.

For `testAlGroups`:

`locus` Name of the locus that was tested.

`SGploidy` The ploidy of each subgenome, taken from the `SGploidy` argument that was passed to `testAlGroups`.

`assignments` Matrix with as many rows as there are isoloci, and as many columns as there are alleles in the dataset. `1` indicates that a given allele belongs to a given isolocus, and `0` indicates that it does not.

`proportion.inconsistent.genotypes`
A number ranging from zero to one indicating the proportion of genotypes from the dataset that are inconsistent with `assignments`.

Note

`alleleCorrelations` will print a warning to the console or to the standard output stream if a significant positive correlation is found between any pair of alleles. (This is not a "warning" in the technical sense usually used in R, because it can occur by random chance and I did not want it to cause **polysat** to fail package checks.) You can see which allele pair(s) caused this warning by looking at `value$significant.pos`. If you receive this warning for many loci, consider that there may be population structure in your dataset, and that you might split the dataset into multiple populations to test seperately. If it happens at just a few loci, check to make sure there are not scoring problems such as stutter peaks being miscalled as alleles. If it only happens at one locus and you can't find any evidence of scoring problems, two alleles may have been positively correlated simply from random chance, and the warning can be ignored.

`alleleCorrelations` can also produce an actual warning stating "Quick-TRANSfer stage steps exceeded maximum". This warning is produced internally by `kmeans` and may occur if many genotypes are similar, as in mapping populations. It can be safely ignored.

Author(s)

Lindsay V. Clark

References

Clark, L. V. and Drauch Schreier, A. (2017) Resolving microsatellite genotype ambiguity in populations of allopolyploid and diploidized autopolyploid organisms using negative correlations between allelic variables. *Molecular Ecology Resources*, **17**, 1090–1103. DOI: 10.1111/1755-0998.12639.

Bertsimas, D. and Tsitsiklis, J.(1993) Simulated annealing. *Statistical Science* **8**, 10–15.

See Also

`recodeAllopoly`, `mergeAlleleAssignments`, `catalanAlleles`, `processDatasetAllo`

Examples

```
# randomly generate example data for an allotetraploid
mydata <- simAllopoly(n.alleles=c(5,5), n.homoplasy=1)
viewGenotypes(mydata)

# test allele correlations
# n.start is lowered in this example to speed up computation time
myCorr <- alleleCorrelations(mydata, n.subgen=2, n.start=10)
myCorr$Kmeans.groups
myCorr$clustering.method
if(!is.null(myCorr$heatmap.dist)) heatmap(myCorr$heatmap.dist)
```

```
# check individual genotypes
# (low maxreps used in order to speed processing time for this example)
myRes <- testAlGroups(mydata, myCorr, SGploidy=2, maxreps = 5)
myRes$assignments
myRes2 <- testAlGroups(mydata, myCorr, SGploidy=2, swap = FALSE)
myRes2$assignments
```

`alleleDiversity` *Retrieve and Count Unique Alleles*

Description

`alleleDiversity` returns the number of unique alleles and/or a list of vectors of all unique alleles, indexed by locus and population.

Usage

```
alleleDiversity(genobject, samples = Samples(genobject),
                loci = Loci(genobject), alleles = TRUE, counts = TRUE)
```

Arguments

`genobject`	An object of the class `"genambig"`.
`samples`	Optional. A character or numeric vector indicating samples to include in the analysis.
`loci`	Optional. A character or numeric vector indicating loci to include in the analysis.
`alleles`	Boolean, indicating whether or not to return the alleles themselves.
`counts`	Boolean, indicating whether or not to return the number of unique alleles.

Value

Under default settings, a list is returned:

`alleles`	A two dimensional list. The first dimension is indexed by population, with the additional element "overall" representing the entire dataset. The second dimension is indexed by locus. Each element of the list is a vector, containing all unique alleles found for that population and locus. `Missing(genobject)` is not counted as an allele.
`counts`	A matrix, indexed in the same way as `alleles`. Each element of the matrix is an integer indicating how many alleles were found at that population and locus.

If the argument `alleles` or `counts` is set to `FALSE`, then only one of the above list elements is returned.

Author(s)

Lindsay V. Clark

See Also

genotypeDiversity

Examples

```
# generate a dataset for this example
mygen <- new("genambig", samples=c("a","b","c","d"), loci=c("E","F"))
PopInfo(mygen) <- c(1,1,2,2)
Genotypes(mygen, loci="E") <- list(c(122,132),c(122,124,140),
                                   c(124,130,132),c(132,136))
Genotypes(mygen, loci="F") <- list(c(97,99,111),c(113,115),
                                   c(99,113),c(111,115))

# look at unique alleles
myal <- alleleDiversity(mygen)
myal$counts
myal$alleles
myal$alleles[["Pop1","E"]]
myal$alleles[["overall","F"]]
```

AllopolyTutorialData *Simulated Allotetraploid Data*

Description

This is a simulated microsatellite dataset for seven loci and 303 individuals. It is intended to be used with the tutorial "Assigning alleles to isoloci in **polysat**".

Usage

```
data("AllopolyTutorialData")
```

Format

A "genambig" object.

Examples

```
data(AllopolyTutorialData)
summary(AllopolyTutorialData)
viewGenotypes(AllopolyTutorialData, samples=1:10, loci=1)
```

`assignClones` *Group Individuals Based on a Distance Threshold*

Description

`assignClones` uses a distance matrix such as that produced by `meandistance.matrix` or `meandistance.matrix2` to place individuals into groups representing asexually-related ramets, or any other grouping based on a distance threshold.

Usage

```
assignClones(d, samples = dimnames(d)[[1]], threshold = 0)
```

Arguments

`d`	A symmetrical, square matrix containing genetic distances between individuals. Both dimensions should be named according to the names of the individuals (samples). A matrix produced by `meandistance.matrix` or `meandistance.matrix2` when `all.distances = FALSE`, or the matrix that is the second item in the list produced if `all.distances = TRUE`, will be in the right format. `meandist.from.array` will also produce a matrix in the correct format.
`samples`	A character vector containing the names of samples to analyze. This should be all or a subset of the names of `d`.
`threshold`	A number indicating the maximum distance between two individuals that will be placed into the same group.

Details

This function groups individuals very similarly to the software GenoType (Meirmans and van Tienderen, 2004). If a distance matrix from **polysat** is exported to GenoType, the results will be the same as those from `assignClones` assuming the same threshold is used. Note that GenoType requires that distances be integers rather than decimals, so you will have to multiply the distances produced by **polysat** by a large number and round them to the nearest integer if you wish to export them to GenoType. When comparing the results of `assignClones` and GenoType using my own data, the only differences I have seen have been the result of rounding; a decimal that was slightly above the threshold in when analyzed in R was rounded down to the threshold when analyzed in GenoType.

Note that when using a distance threshold of zero (the default), it is advisable to exclude all samples with missing data, in order to prevent the merging of non-identical clones. At higher thresholds, some missing data are allowable, but samples that have missing data at many loci should be excluded.

The `write.table` function can be used for exporting the results to GenoDive. See the R documentation for information on how to make a tab-delimited file with no header.

Value

A numeric vector, named by `samples`. Each clone or group is given a number, and the number for each sample indicates the clone or group to which it belongs.

Author(s)

Lindsay V. Clark

References

Meirmans, P. G. and Van Tienderen, P. H. (2004) GENOTYPE and GENODIVE: two programs for the analysis of genetic diversity of asexual organisms. *Molecular Ecology Notes* **4**, 792–794.

See Also

genotypeDiversity

Examples

```
# set up a simple matrix with three samples
test <- matrix(c(0,0,.5,0,0,.5,.5,.5,0), ncol=3, nrow=3)
abc <- c("a", "b", "c")
dimnames(test) <- list(abc,abc)

# assign clones with a threshold of zero or 0.5
assignClones(test)
assignClones(test, threshold=0.5)
```

Bruvo.distance *Genetic Distance Metric of Bruvo et al.*

Description

This function calculates the distance between two individuals at one microsatellite locus using a method based on that of Bruvo *et al.* (2004).

Usage

```
Bruvo.distance(genotype1, genotype2, maxl=10, usatnt=2, missing=-9)
```

Arguments

genotype1	A vector of alleles for one individual at one locus. Allele length is in nucleotides or repeat count. Each unique allele corresponds to one element in the vector, and the vector is no longer than it needs to be to contain all unique alleles for this individual at this locus.
genotype2	A vector of alleles for another individual at the same locus.
maxl	If both individuals have more than this number of alleles at this locus, NA is returned instead of a numerical distance.
usatnt	Length of the repeat at this locus. For example usatnt=2 for dinucleotide repeats, and usatnt=3 for trinucleotide repeats. If the alleles in genotype1 and genotype2 are expressed in repeat count instead of nucleotides, set usatnt=1.
missing	A numerical value that, when in the first allele position, indicates missing data. NA is returned if this value is found in either genotype.

Details

Since allele copy number is frequently unknown in polyploid microsatellite data, Bruvo *et al.* developed a measure of genetic distance similar to band-sharing indices used with dominant data, but taking into account mutational distances between alleles. A matrix is created containing all differences in repeat count between the alleles of two individuals at one locus. These differences are then geometrically transformed to reflect the probabilities of mutation from one allele to another. The matrix is then searched to find the minimum sum if each allele from one individual is paired to one allele from the other individual. This sum is divided by the number of alleles per individual.

If one genotype has more alleles than the other, 'virtual alleles' must be created so that both genotypes are the same length. There are three options for the value of these virtual alleles, but `Bruvo.distance` only implements the simplest one, assuming that it is not known whether differences in ploidy arose from genome addition or genome loss. Virtual alleles are set to infinity, such that the geometric distance between any allele and a virtual allele is 1.

In the original publication by Bruvo *et al.* (2004), ambiguous genotypes were dealt with by calculating the distance for all possible unambiguous genotype combinations and averaging across all of them equally. When `Bruvo.distance` is called from `meandistance.matrix`, ploidy is unknown and all genotypes are simply treated as if they had one copy of each allele. When `Bruvo.distance` is called from `meandistance.matrix2`, the analysis is truer to the original, in that ploidy is known and all possible unambiguous genotype combinations are considered. However, instead of all possible unambiguous genotypes being weighted equally, in `meandistance.matrix2` they are weighted based on allele frequencies and selfing rate, since some unambiguous genotypes are more likely than others.

Value

A number ranging from 0 to 1, with 0 indicating identical genotypes, and 1 being a theoretical maximum distance if all alleles from `genotype1` differed by an infinite number of repeats from all alleles in `genotype2`. `NA` is returned if both genotypes have more than `maxl` alleles or if either genotype has the symbol for missing data as its first allele.

Note

The processing time is a function of the factorial of the number of alleles, since each possible combination of allele pairs must be evaluated. For genotypes with a sufficiently large number of alleles, it may be more efficient to estimate distances manually by creating the matrix in Excel and visually picking out the shortest distances between alleles. This is the purpose of the `maxl` argument. On my personal computer, if both genotypes had more than nine alleles, the calculation could take an hour or more, and so this is the default limit. In this case, `Bruvo.distance` returns `NA`.

Author(s)

Lindsay V. Clark

References

Bruvo, R., Michiels, N. K., D'Sousa, T. G., and Schulenberg, H. (2004) A simple method for calculation of microsatellite genotypes irrespective of ploidy level. *Molecular Ecology* **13**, 2101-

2106.

See Also

meandistance.matrix, Lynch.distance, Bruvo2.distance

Examples

```
Bruvo.distance(c(202,206,210,220),c(204,206,216,222))
Bruvo.distance(c(202,206,210,220),c(204,206,216,222),usatnt=4)
Bruvo.distance(c(202,206,210,220),c(204,206,222))
Bruvo.distance(c(202,206,210,220),c(204,206,216,222),maxl=3)
Bruvo.distance(c(202,206,210,220),c(-9))
```

Bruvo2.distance *Distance Measure of Bruvo et al. under Genome Loss and Addition*

Description

This is an inter-individual distance measure similar to Bruvo.distance, except that where genotypes have different numbers of alleles, virtual alleles are equal to those from the longer and/or shorter genotype, rather than being equal to infinity.

Usage

```
Bruvo2.distance(genotype1, genotype2, maxl = 7, usatnt = 2,
                missing = -9, add = TRUE, loss = TRUE)
```

Arguments

genotype1	A numeric vector representing the genotype of one individual at one locus. This type of vector is produced by the Genotype method for "genambig" objects.
genotype2	The second genotype for the distance calculation, in the same format as genotype1.
maxl	The maximum number of alleles that either genotype can have. If it is exceeded, NA is returned. This argument exists to prevent computations that would take in excess of an hour; see Bruvo.distance.
usatnt	The microsatellite repeat type for the locus. 2 for dinucleotide repeats, 3 for dinucleotide repeats, 1 if the alleles are already coded as repeat numbers, etc. See Usatnts.
missing	The symbol that indicates missing data for a given sample and locus. See Missing.
add	TRUE if the model of genome addition is being used, and FALSE if it is not. If this model is used, the shorter genotype will have virtual alleles added from the same genotype.
loss	TRUE if the model of genome loss is being used, and FALSE if it is not. If this model is used, the shorter genotype will have virtual alleles added from the longer genotype.

Details

Bruvo *et al.* (2004) describe multiple methods for calculating genetic distances between individuals of different ploidy. (See "Special cases" starting on page 2102 of the paper.) The original `Bruvo.distance` function in **polysat** uses the method described for systems with complex changes in ploidy, adding virtual alleles equal to infinity to the shorter genotype to make it the same length as the longer genotype. This method, however, can exaggerate distances between individuals of different ploidy, particularly when used with `meandistance.matrix2` as opposed to `meandistance.matrix`.

`Bruvo2.distance` calculates distances between individuals under the models of genome addition and genome loss. If `add = TRUE` and `loss = TRUE`, the distance produced is equal to that of equation 6 in the paper.

If `add = TRUE` and `loss = FALSE`, the distance calculated is that under genome addition only. Likewise if `add = FALSE` and `loss = TRUE` the distance is calculated under genome loss only. The latter distance should be greater than the former. If both were averaged together, they would give the identical result to that produced when `add = TRUE` and `loss = TRUE`. All three distances will be less than that produced by `Bruvo.distance`.

If both genotypes have the same number of alleles, they are passed to `Bruvo.distance` for the calculation. This also happens if `add = FALSE` and `loss = FALSE`. Otherwise, if the genotypes have different numbers of alleles, all possible genotypes with virtual alleles are enumerated and passed to `Bruvo.distance` one by one, and the results averaged.

The number of different genotypes simulated under genome loss or genome addition is l^d, where l is the length of the genotype from which virtual alleles are being taken, and d is the difference in length between the longer and shorter genotype. For example, under genome addition for a diploid individual with alleles 1 and 2 being compared to a tetraploid individual, the genotypes 1211, 1212, 1221, and 1222 will each be used once to represent the diploid individual.

Value

A decimal between 0 and 1, with 0 indicating complete identity of two genotypes, and 1 indicating maximum dissimilarity. `NA` is returned if one or both genotypes are missing or if `maxl` is exceeded.

Note

Figure 1B and 1C of Bruvo *et al.* (2004) illustrate an example of this distance measure. To perform the identical calculation to that listed directly under equation 6, you would type:

```
Bruvo2.distance(c(20,23,24), c(20,24,26,43), usatnt=1)
```

However, you will notice that the result, 0.401, is slightly different from that given in the paper. This is due to an error in the paper. For the distance under genome loss when the virtual allele is 26, the result should be 1 instead of 1.75.

Author(s)

Lindsay V. Clark

References

Bruvo, R., Michiels, N. K., D'Sousa, T. G., and Schulenberg, H. (2004) A simple method for calculation of microsatellite genotypes irrespective of ploidy level. *Molecular Ecology* **13**, 2101–2106.

See Also

`Lynch.distance`, `Bruvo.distance`, `meandistance.matrix2`

Examples

```
Bruvo2.distance(c(102,104), c(104,104,106,110))
Bruvo2.distance(c(102,104), c(104,104,106,110), add = FALSE)
Bruvo2.distance(c(102,104), c(104,104,106,110), loss = FALSE)
```

---

calcPopDiff *Estimate Population Differentiation Statistics*

Description

Given a data frame of allele frequencies and population sizes, `calcPopDiff` calculates a matrix of pairwise F_{ST}, G_{ST}, Jost's D, or R_{ST} values, or a single global value for any of these four statistics. `calcFst` is a wrapper for `calcPopDiff` to allow backwards compatibility with previous versions of **polysat**.

Usage

```
calcPopDiff(freqs, metric, pops = row.names(freqs),
            loci = unique(gsub("\\..*$", "", names(freqs))), global = FALSE,
            bootstrap = FALSE, n.bootstraps = 1000, object = NULL)

calcFst(freqs, pops = row.names(freqs),
        loci = unique(gsub("\\..*$", "", names(freqs))), ...)
```

Arguments

`freqs`	A data frame of allele frequencies and population sizes such as that produced by `simpleFreq` or `deSilvaFreq`. Each population is in one row, and a column called `Genomes` (or multiple columns containing the locus names and "Genomes" seperated by a period) contains the relative size of each population. All other columns contain allele frequencies. The names of these columns are the locus name and allele name, separated by a period.
`metric`	The population differentiation statistic to estimate. Can be "Fst", "Gst", "Jost's D", or "Rst".
`pops`	A character vector. Populations to analyze, which should be a subset of `row.names(freqs)`.
`loci`	A character vector indicating which loci to analyze. These should be a subset of the locus names as used in the column names of `freqs`.

global	Boolean. If TRUE, a single global statistic will be estimated across all populations. If FALSE, pairwise statistics will be estimated between populations.
bootstrap	Boolean. If TRUE, a set of replicates bootstrapped across loci will be returned. If FALSE, a single value will be returned for each pair of populations (if global = FALSE) or for the whole set (if global = TRUE).
n.bootstraps	Integer. The number of bootstrap replicates to perform. Ignored if bootstrap = FALSE.
object	A "genambig" or "genbinary" object with the Usatnts slot filled in. Required for metric = "Rst" only.
...	Additional arguments to be passed to calcPopDiff (*i.e.* global, bootstrap, and/or n.bootstraps).

Details

For metric = "Fst" or calcFst:

H_S and H_T are estimate directly from allele frequencies for each locus for each pair of populations, then averaged across loci. Wright's F_{ST} is then calculated for each pair of populations as $\frac{H_T - H_S}{H_T}$.

H values (expected heterozygosities for populations and combined populations) are calculated as one minus the sum of all squared allele frequencies at a locus. To calculte H_T, allele frequencies between two populations are averaged before the calculation. To calculate H_S, H values are averaged after the calculation. In both cases, the averages are weighted by the relative sizes of the two populations (as indicated by freqs$Genomes).

For metric = "Gst":

This metric is similar to F_{ST}, but mean allele frequencies and mean expected heterozygosities are not weighted by population size. Additionally, unbiased estimators of H_S and H_T are used according to Nei and Chesser (1983; equations 15 and 16, reproduced also in Jost (2008)). Instead of using twice the harmonic mean of the number of individuals in the two subpopulations as described by Nei and Chesser (1983), the harmonic mean of the number of allele copies in the two subpopulations (taken from freq$Genomes) is used, in order to allow for polyploidy. G_{ST} is estimated for each locus and then averaged across loci.

For metric = "Jost's D":

The unbiased estimators of H_S and H_T and calculated as with G_{ST}, without weighting by population size. They are then used to estimate D at each locus according to Jost (2008; equation 12):

$$2 * \frac{H_T - H_S}{1 - H_S}$$

Values of D are then averaged across loci.

For metric = "Rst":

R_{ST} is estimated on a per-locus basis according to Slatkin (1995), but with populations weighted equally regardless of size. Values are then averaged across loci.

For each locus:

$$S_w = \frac{1}{d} * \sum_{j}^{d} \frac{\sum_i \sum_{i'<i} p_{ij} * p_{i'j} * n_j^2 * (a_i - a_{i'})^2}{n_j * (n_j - 1)}$$

$$\bar{S} = \frac{\sum_i \sum_{i'<i} \bar{p}_i * \bar{p}_{i'} * n^2 * (a_i - a_{i'})^2}{n * (n - 1)}$$

$$R_{ST} = \frac{\bar{S} - S_w}{\bar{S}}$$

where d is the number of populations, j is an individual population, i and i' are individual alleles, p_{ij} is the frequency of an allele in a population, n_j is the number of allele copies in a population, a_i is the size of an allele expressed in number of repeat units, $\bar{p}_i$ is an allele frequency averaged across populations (with populations weighted equally), and n is the total number of allele copies across all populations.

Value

If `global` = `FALSE` and `bootstrap` = `FALSE`, a square matrix containing F_{ST}, G_{ST}, D, or R_{ST} values. The rows and columns of the matrix are both named by population.

If `global` = `TRUE` and `bootstrap` = `FALSE`, a single value indicating the F_{ST}, G_{ST}, D, or R_{ST} value.

If `global` = `TRUE` and `bootstrap` = `TRUE`, a vector of bootstrapped replicates of F_{ST}, G_{ST}, D, or R_{ST}.

If `global` = `FALSE` and `bootstrap` = `TRUE`, a square two-dimensional list, with rows and columns named by population. Each item is a vector of bootstrapped values for F_{ST}, G_{ST}, D, or R_{ST} for that pair of populations.

Author(s)

Lindsay V. Clark

References

Nei, M. (1973) Analysis of gene diversity in subdivided populations. *Proceedings of the National Academy of Sciences of the United States of America* **70**, 3321–3323.

Nei, M. and Chesser, R. (1983) Estimation of fixation indices and gene diversities. *Annals of Human Genetics* **47**, 253–259.

Jost, L. (2008) G_{ST} and its relatives to not measure differentiation. *Molecular Ecology* **17**, 4015–4026.

Slatkin, M. (1995) A measure of population subdivision based on microsatellite allele frequencies. *Genetics* **139**, 457–462.

See Also

simpleFreq, deSilvaFreq

Examples

```
# create a data set (typically done by reading files)
mygenotypes <- new("genambig", samples = paste("ind", 1:6, sep=""),
                   loci = c("loc1", "loc2"))
Genotypes(mygenotypes, loci = "loc1") <- list(c(206), c(208,210),
                                              c(204,206,210),
    c(196,198,202,208), c(196,200), c(198,200,202,204))
Genotypes(mygenotypes, loci = "loc2") <- list(c(130,134), c(138,140),
                                              c(130,136,140),
    c(138), c(136,140), c(130,132,136))
PopInfo(mygenotypes) <- c(1,1,1,2,2,2)
mygenotypes <- reformatPloidies(mygenotypes, output="sample")
Ploidies(mygenotypes) <- c(2,2,4,4,2,4)
Usatnts(mygenotypes) <- c(2,2)

# calculate allele frequencies
myfreq <- simpleFreq(mygenotypes)

# calculate pairwise differentiation statistics
myfst <- calcPopDiff(myfreq, metric = "Fst")
mygst <- calcPopDiff(myfreq, metric = "Gst")
myD <- calcPopDiff(myfreq, metric = "Jost's D")
myrst <- calcPopDiff(myfreq, metric = "Rst", object = mygenotypes)

# examine the results
myfst
mygst
myD
myrst

# get global statistics
calcPopDiff(myfreq, metric = "Fst", global = TRUE)
calcPopDiff(myfreq, metric = "Gst", global = TRUE)
calcPopDiff(myfreq, metric = "Jost's D", global = TRUE)
calcPopDiff(myfreq, metric = "Rst", global = TRUE, object = mygenotypes)
```

---

catalanAlleles *Sort Alleles into Isoloci*

Description

catalanAlleles uses genotypes present in a "genambig" object to sort alleles from one locus into two or more isoloci in an allopolyploid or diploidized autopolyploid species. Alleles are determined to belong to different isoloci if they are both present in a fully homozygous genotype. If necessary, heterozygous genotypes are also examined to resolve remaining alleles.

Usage

```
catalanAlleles(object, samples = Samples(object), locus = 1,
               n.subgen = 2, SGploidy = 2, verbose = FALSE)
```

Arguments

`object`	A `"genambig"` object containing the dataset to analyze. All individuals should be the same ploidy, although the function does not access the `Ploidies` slot. Missing data are allowed. For the locus to be examined, no genotype should have fewer than `n.subgen` alleles or more than `n.subgen*SGploidy` alleles.
`samples`	Optional argument indicating samples to be analyzed. Can be integer or character, as with other **polysat** functions.
`locus`	An integer or character string indicating which locus to analyze. Cannot be a vector greater than length 1. (The function will only analyze one locus at a time.)
`n.subgen`	The number of isoloci (number of subgenomes). For example, `2` for an allotetraploid, and `3` for an allohexaploid (three diploid genomes).
`SGploidy`	The ploidy of each genome. Only one value is allowed; all genomes must be the same ploidy. `2` indicates that each subgenome is diploid (as in an allotetraploid, or an allohexaploid with three diploid genomes).
`verbose`	Boolean. Indicates whether results, and if applicable, problematic genotypes, should be printed to the console.

Details

`catalanAlleles` implements and extends an approach used by Catalan *et al.* (2006) that sorts alleles from a duplicated microsatellite locus into two or more isoloci (homeologous loci on different subgenomes). First, fully homozygous genotypes are identified and used in analysis. If a genotype has as many alleles as there are subgenomes (for example, a genotype with two alleles in an allotetraploid species), it is assumed to be fully homozygous and the alleles are assumed to belong to different subgenomes. If some alleles remain unassigned after examination of all fully homozygous genotypes, heterozygous genotypes are also examined to attempt to assign those remaining alleles.

For example, in an allotetraploid, if a genotype contains one unassigned allele, and all other alleles in the genotype are known to belong to one isolocus, the unassigned allele can be assigned to the other isolocus. Or, if two alleles in a genotype belong to one isolocus, one allele belongs to the other isolocus, and one allele is unassigned, the unassigned allele can be assigned to the latter isolocus. The function follows such logic (which can be extended to higher ploidies) until all alleles can be assigned, or returns a text string saying that the allele assignments were unresolvable.

It is important to note that this method assumes no null alleles and no homoplasy across isoloci. If the function encounters evidence of either it will not return allele assignments. Null alleles and homoplasy are real possibilities in any dataset, which means that this method simply will not work for some microsatellite loci.

(Null alleles are those that do not produce a PCR amplicon, usually because of a mutation in the primer binding site. Alleles that exhibit homoplasy are those that produce amplicons of the same size, despite not being identical by descent. Specifically, homoplasy between alleles from different isoloci will interfere with the Catalan method of allele assignment.)

Value

A list containing the following items:

`locus`	A character string giving the name of the locus.
`SGploidy`	A number giving the ploidy of each subgenome. Identical to the `SGploidy` argument.
`assignments`	If assignments cannot be made, a character string describing the problem. Otherwise, a matrix with `n.subgen` rows and a labeled column for each allele, with a `1` if the allele belongs to that subgenome and a `0` if it does not.

Note

Aside from homoplasy and null alleles, stochastic effects may prevent the minimum combination of genotypes needed to resolve all alleles from being present in the dataset. For a typical allotetraploid dataset, 50 to 100 samples will be needed, whereas an allohexaploid dataset may require over 100 samples. In simulations, allo-octoploid datasets with two tetraploid genomes were unresolvable even with 10,000 samples due to the low probability of finding full homozygotes. Additionally, loci are less likely to be resolvable if they have many alleles or if one isolocus is monomorphic.

Although determination of allele copy number by is not needed (or expected) for `catalanAlleles` as it was in the originally published Catalan method, it is still very important that the genotypes be high quality. Even a single scoring error can cause the method to fail, including allelic dropout, contamination between samples, stutter peaks miscalled as alleles, and PCR artifacts miscalled as alleles. Poor quality loci (those that require some "artistic" interpretation of gels or electropherograms) are unlikely to work with this method. Individual genotypes that are of questionable quality should be discarded before running the function.

Author(s)

Lindsay V. Clark

References

Catalan, P., Segarra-Moragues, J. G., Palop-Esteban, M., Moreno, C. and Gonzalez-Candelas, F. (2006) A Bayesian approach for discriminating among alternative inheritance hypotheses in plant polyploids: the allotetraploid origin of genus Borderea (Dioscoreaceae). *Genetics* **172**, 1939–1953.

See Also

`alleleCorrelations`, `mergeAlleleAssignments`, `recodeAllopoly`, `simAllopoly`

Examples

```
# make the default simulated allotetraploid dataset
mydata <- simAllopoly()

# resolve the alleles
myassign <- catalanAlleles(mydata)
```

`deleteSamples` *Remove Samples or Loci from an Object*

Description

These functions remove samples or loci from all relevant slots of an object.

Usage

```
deleteSamples(object, samples)
deleteLoci(object, loci)
```

Arguments

`object`	An object containing the dataset of interest. Generally an object of some subclass of `gendata`.
`samples`	A numerical or character vector of samples to be removed.
`loci`	A numerical or character vector of loci to be removed.

Details

These are generic functions with methods for `genambig`, `genbinary`, and `gendata` objects. The methods for the subclasses remove samples or loci from the `@Genotypes` slot, then pass the object to the method for `gendata`, which removes samples or loci from the `@PopInfo`, `@Ploidies`, and/or `@Usatnts` slots, as appropriate. The `@PopNames` slot is left untouched even if an entire population is deleted, in order to preserve the connection between the numbers in `@PopInfo` and the names in `@PopNames`.

If your intent is to experiment with excluding samples or loci, it may be a better idea to create character vectors of samples and loci that you want to use and then use these vectors as the `samples` and `loci` arguments for analysis or export functions.

Value

An object identical to `object`, but with the specified samples or loci removed.

Note

These functions are somewhat redundant with the subscripting function `"["`, which also works for all `gendata` objects. However, they may be more convenient depending on whether the user prefers to specify the samples and loci to use or to exclude.

Author(s)

Lindsay V. Clark

See Also

`Samples`, `Loci`, `merge,gendata,gendata-method`

Examples

```
# set up genambig object
mygen <- new("genambig", samples = c("ind1", "ind2", "ind3", "ind4"),
             loci = c("locA", "locB", "locC", "locD"))

# delete a sample
Samples(mygen)
mygen <- deleteSamples(mygen, "ind1")
Samples(mygen)

# delete some loci
Loci(mygen)
mygen <- deleteLoci(mygen, c("locB", "locC"))
Loci(mygen)
```

`deSilvaFreq` *Estimate Allele Frequencies with EM Algorithm*

Description

This function uses the method of De Silva *et al.* (2005) to estimate allele frequencies under polysomic inheritance with a known selfing rate.

Usage

```
deSilvaFreq(object, self, samples = Samples(object),
            loci = Loci(object), initNull = 0.15,
            initFreq = simpleFreq(object[samples, loci]),
            tol = 1e-08, maxiter = 1e4)
```

Arguments

`object`	A "genambig" or "genbinary" object containing the dataset of interest. All ploidies for `samples` and `loci` should be the same, and this should be an even number. `PopInfo` must also be filled in for `samples`.
`self`	A number between 1 and 0, indicating the rate of selfing.
`samples`	An optional character vector indicating a subset of samples to use in the calculation.
`loci`	An optional character vector indicating a subset of loci for which to calculate allele frequencies.
`initNull`	A single value or numeric vector indicating initial frequencies to use for the null allele at each locus.
`initFreq`	A data frame containing allele frequencies (for non-null loci) to use for initialization. This needs to be in the same format as the output of `simpleFreq` with a single "Genomes" column (similarly to the format of the output of `deSilvaFreq`). By default, the function will do a quick estimation of allele frequencies using `simpleFreq` and then initialize the EM algorithm at these frequencies.

`tol`	The tolerance level for determining when the results have converged. Where `p2` and `p1` are the current and previous vectors of allele frequencies, respectively, the EM algorithm stops if `sum(abs(p2-p1)/(p2+p1)) <= tol`.
`maxiter`	The maximum number of iterations that will be performed for each locus and population.

Details

Most of the SAS code from the supplementary material of De Silva *et al.* (2005) is translated directly into the R code for this function. The SIMSAMPLE (or CreateRandomSample in the SAS code) function is omitted so that the actual allelic phenotypes from the dataset can be used instead of simulated phenotypes. `deSilvaFreq` loops through each locus and population, and in each loop tallies the number of alleles and sets up matrices using GENLIST, PHENLIST, RANMUL, SELFMAT, and CONVMAT as described in the paper. Frequencies of each allelic phenotype are then tallied across all samples in that population with non-missing data at the locus. Initial allele frequencies for that population and locus are then extraced from `initFreq` and adjusted according to `initNull`. The EM iteration then begins for that population and locus, as described in the paper (EXPECTATION, GPROBS, and MAXIMISATION).

Each repetition of the EM algorithm includes an expectation and maximization step. The expectation step uses allele frequencies and the selfing rate to calculate expected genotype frequencies, then uses observed phenotype frequencies and expected genotype frequencies to estimate genotype frequencies for the population. The maximization step uses the estimated genotype frequencies to calculate a new set of allele frequencies. The process is repeated until allele frequencies converge.

In addition to returning a data frame of allele frequencies, `deSilvaFreq` also prints to the console the number of EM repetitions used for each population and locus. When each locus and each population is begun, a message is printed to the console so that the user can monitor the progress of the computation.

Value

A data frame containing the estimated allele frequencies. The row names are population names from `PopNames(object)`. The first column shows how many genomes each population has. All other columns represent alleles (including one null allele per locus). These column names are the locus name and allele name separated by a period.

Note

It is possible to exceed memory limits for R if a locus has too many alleles in a population (e.g. 15 alleles in a tetraploid if the memory limit is 1535 Mb, see `memory.limit`).

De Silva *et al.* mention that their estimation method could be extended to the case of disomic inheritence. A method for disomic inheritence is not implemented here, as it would require knowledge of which alleles belong to which isoloci.

De Silva *et al.* also suggest a means of estimating the selfing rate with a least-squares method. Using the notation in the source code, this would be:

```
lsq <- smatt %*% EP - rvec
self <- as.vector((t(EP - rvec) %*% lsq)/(t(lsq) %*% lsq))
```

However, in my experimentation with this calculation, it sometimes yields selfing rates greater than one. For this reason, it is not implemented here.

Author(s)

Lindsay V. Clark

References

De Silva, H. N., Hall, A. J., Rikkerink, E., and Fraser, L. G. (2005) Estimation of allele frequencies in polyploids under certain patterns of inheritance. *Heredity* **95**, 327–334

See Also

simpleFreq, write.freq.SPAGeDi, GENLIST

Examples

```
## Not run:
## An example with a long run time due to the number of alleles

# create a dataset for this example
mygen <- new("genambig", samples=c(paste("A", 1:100, sep=""),
                                   paste("B", 1:100, sep="")),
             loci=c("loc1", "loc2"))
PopNames(mygen) <- c("PopA", "PopB")
PopInfo(mygen) <- c(rep(1, 100), rep(2, 100))
mygen <- reformatPloidies(mygen, output="one")
Ploidies(mygen) <- 4
Usatnts(mygen) <- c(2, 2)
Description(mygen) <- "An example for allele frequency calculation."

# create some genotypes at random for this example
for(s in Samples(mygen)){
    Genotype(mygen, s, "loc1") <- sample(seq(120, 140, by=2),
                                         sample(1:4, 1))
}
for(s in Samples(mygen)){
    Genotype(mygen, s, "loc2") <- sample(seq(130, 156, by=2),
                                         sample(1:4, 1))
}
# make one genotype missing
Genotype(mygen, "B4", "loc2") <- Missing(mygen)

# view the dataset
summary(mygen)
viewGenotypes(mygen)

# calculate the allele frequencies if the rate of selfing is 0.2
myfrequencies <- deSilvaFreq(mygen, self=0.2)

# view the results
```

```
myfrequencies

## End(Not run)

## An example with a shorter run time, for checking that the funciton
## is working.  Genotype simulation is also a bit more realistic here.

# Create a dataset for the example.
mygen <- new("genambig", samples=paste("A", 1:100, sep=""), loci="loc1")
PopNames(mygen) <- "PopA"
PopInfo(mygen) <- rep(1, 100)
mygen <- reformatPloidies(mygen, output="one")
Ploidies(mygen) <- 4
Usatnts(mygen) <- 2
for(s in Samples(mygen)){
    alleles <- unique(sample(c(122,124,126,0), 4, replace=TRUE,
                             prob = c(0.3, 0.2, 0.4, 0.1)))
    Genotype(mygen, s, "loc1") <- alleles[alleles != 0]
    if(length(Genotype(mygen, s, "loc1"))==0)
        Genotype(mygen, s, "loc1") <- Missing(mygen)
}

# We have created a random mating populations with four alleles
# including one null.  The allele frequencies are given in the
# 'prob' argument.

# Estimate allele frequencies
myfreq <- deSilvaFreq(mygen, self=0.01)
myfreq
```

`editGenotypes` *Edit Genotypes Using the Data Editor*

Description

The genotypes from an object of one of the subclasses of `gendata` are converted to a data frame (if necessary), then displayed in the data editor. After the user makes the desired edits and closes the data editor window, the new genotypes are written to the `gendata` object and the object is returned.

Usage

```
editGenotypes(object, maxalleles = max(Ploidies(object)),
              samples = Samples(object), loci = Loci(object))
```

Arguments

`object`	An object of the class `genambig` or `genbinary`. Contains the genotypes to be edited.

maxalleles	Numeric. The maximum number of alleles found in any given genotype. The method for genambig requires this information in order to determine how many columns to put in the data frame.
samples	Character or numeric vector indicating which samples to edit.
loci	Character or numeric vector indicating which loci to edit.

Details

The method for genambig lists sample and locus names in each row in order to identify the genotypes. However, only the alleles themselves should be edited. NA values and duplicate alleles in the data editor will be omitted from the genotype vectors that are written back to the genambig object.

Value

An object identical to object but with edited genotypes.

Author(s)

Lindsay V. Clark

See Also

viewGenotypes, Genotype<-, Genotypes<-

Examples

```
if(interactive()){  #this line included for automated checking on CRAN

# set up "genambig" object to edit
mygen <- new("genambig", samples = c("a", "b", "c"),
             loci = c("loc1", "loc2"))
Genotypes(mygen, loci="loc1") <- list(c(133, 139, 142),
                                      c(130, 136, 139, 145),
                                      c(136, 142))
Genotypes(mygen, loci="loc2") <- list(c(202, 204), Missing(mygen),
                                      c(200, 206, 208))
mygen <- reformatPloidies(mygen, output="one")
Ploidies(mygen) <- 4

# open up the data editor
mygen <- editGenotypes(mygen)

# view the results of your edits
viewGenotypes(mygen)

}
```

`estimatePloidy` *Estimate Ploidies Based on Allele Counts*

Description

`estimatePloidy` calculates the maximum and mean number of unique alleles for each sample across a given set of loci. These values are presented in a data editor, along with other pertinent information, so that the user can then edit the ploidy values for the object.

Usage

```
estimatePloidy(object, extrainfo, samples = Samples(object),
               loci = Loci(object))
```

Arguments

`object`	The object containing genotype data, and to which ploidies will be written.
`extrainfo`	A named or unnamed vector or data frame containing extra information (such as morphological or flow cytometry data) to display in the data editor, to assist with making decisions about ploidy. If unnamed, the vector (or the rows of the data frame) is assumed to be in the same order as `samples`. An array can also be given as an argument here, and will be coerced to a data frame.
`samples`	A numeric or character vector indicating a subset of samples to evaluate.
`loci`	A numeric or character vector indicating a subset of loci to use in the calculation of mean and maximum allele number.

Details

`estimatePloidy` is a generic function with methods written for the `genambig` and `genbinary` classes.

If the `Ploidies` slot of `object` is not already a `"ploidysample"` object, the function will first convert the `Ploidies` slot to this format, deleting any data that is currently there. (Ploidies must be indexed by sample and not by locus.) If ploidies were already in the `"ploidysample"` format, any ploidy data already in the object is retained and put into the table (see below).

Population identities are displayed in the table only if more than one population identity is found in the dataset. Likewise, the current ploidies of the dataset are only displayed if there is more than one ploidy level already found in `Ploidies(object)`.

Missing genotypes are ignored; maximum and mean allele counts are only calculted across genotypes that are not missing. If all genotypes for a given sample are missing, `NA` is displayed in the corresponding cells in the data editor.

The default values for `new.ploidy` are the maximum number of alleles per locus for each sample.

Value

`object` is returned, with `Ploidies(object)` now equal to the values set in the `new.ploidy` column of the data editor.

Author(s)

Lindsay V. Clark

See Also

genambig, genbinary, Ploidies

Examples

```
if(interactive()){ #this line included for automated checking on CRAN

# create a dataset for this example
mygen <- new("genambig", samples=c("a", "b", "c"),
             loci=c("loc1", "loc2"))
Genotypes(mygen, loci="loc1") <- list(c(122, 126, 128), c(124, 130),
                                      c(120, 122, 124))
Genotypes(mygen, loci="loc2") <- list(c(140, 148), c(144, 150), Missing(mygen))

# estimate the ploidies
mygen <- estimatePloidy(mygen)

# view the ploidies
Ploidies(mygen)

}
```

FCRinfo *Additional Data on Rubus Samples*

Description

For 20 *Rubus* samples, contains colors and symbols to use for plotting data.

Usage

```
data(FCRinfo)
```

Format

Data frame. `FCRinfo$Plot.color` contains character strings of the colors to be used to represent species groups. `FCR.info$Plot.symbol` contains integers to be passed to `pch` to designate the symbol used to represent each individual. These reflect chloroplast haplotypes.

Source

Clark, L. V. and Jasieniuk, M. (2012) Spontaneous hybrids between native and exotic *Rubus* in the Western United States produce offspring both by apomixis and by sexual recombination. *Heredity* **109**, 320–328. Data available at: doi:10.5061/dryad.m466f

See Also

testgenotypes

`find.missing.gen` *Find Missing Genotypes*

Description

This function returns a data frame listing the locus and sample names of all genotypes with missing data.

Usage

```
find.missing.gen(object, samples = Samples(object),
                 loci = Loci(object))
```

Arguments

`object`	A `genambig` or `genbinary` object containing the genotypes of interest.
`samples`	A character vector of all samples to be searched. Must be a subset of `Samples(object)`.
`loci`	A character vector of all loci to be searched. Must be a subset of `Loci(object)`.

Value

A data frame with no row names. The first column is named "Locus" and the second column is named "Sample". Each row represents one missing genotype, and gives the locus and sample of that genotype.

Author(s)

Lindsay V. Clark

See Also

isMissing

Examples

```
# set up the genotype data
samples <- paste("ind", 1:4, sep="")
samples
loci <- paste("loc", 1:3, sep="")
loci
testgen <- new("genambig", samples = samples, loci = loci)
Genotypes(testgen, loci="loc1") <- list(c(-9), c(102,104),
                                        c(100,106,108,110,114),
                                        c(102,104,106,110,112))
Genotypes(testgen, loci="loc2") <- list(c(77,79,83), c(79,85), c(-9),
```

```
                                        c(83,85,87,91))
Genotypes(testgen, loci="loc3") <- list(c(122,128), c(124,126,128,132),
                                        c(120,126), c(124,128,130))

# look up which samples*loci have missing genotypes
find.missing.gen(testgen)
```

`freq.to.genpop` *Convert Allele Frequencies for Adegenet*

Description

Given a data frame of allele frequencies such as that produced by `simpleFreq` or `deSilvaFreq`, `freq.to.genpop` creates a data frame of allele counts that can be read by the `as.genpop` function in the package **adegenet**.

Usage

```
freq.to.genpop(freqs, pops = row.names(freqs),
               loci =
                unique(as.matrix(as.data.frame(strsplit(names(freqs),
                split = ".", fixed = TRUE), stringsAsFactors = FALSE))[1, ]))
```

Arguments

`freqs`	A data frame of allele frequencies. Row names are population names. The first column is called `"Genomes"` and indicates the size of each population in terms of number of haploid genomes. All other column names are the locus and allele separated by a period. These columns contain the frequencies of each allele in each population. For each locus and population, all frequencies should total to 1.
`pops`	An optional character vector indicating the names of populations to use.
`loci`	An optional character vector indicating the names of loci to use.

Details

adegenet expects one ploidy for the entire dataset. Therefore, data frames of allele frequencies with multiple "Genomes" columns, such as those produced when ploidy varies by locus, are not allowed as the `freqs` argument.

Value

A data frame with row and column names identical to those in `freqs`, minus the `"Genomes"` column and any columns for loci not included in `loci`. Allele frequencies are converted to counts by multiplying by the values in the `"Genomes"` column and rounding to the nearest integer.

Author(s)

Lindsay V. Clark

References

Jombart, T. (2008) adegenet: a R package for the multivariate analysis of genetic markers. *Bioinformatics* **24**, 1403-1405.

See Also

simpleFreq, deSilvaFreq, write.freq.SPAGeDi, gendata.to.genind

Examples

```
# create a simple allele frequency table
# (usually done with simpleFreq or deSilvaFreq)
myfreq <- data.frame(row.names=c("popA","popB"), Genomes=c(120,100),
                     locG.152=c(0.1,0.4), locG.156=c(0.5, 0.3),
                     locG.160=c(0.4, 0.3), locK.179=c(0.15, 0.25),
                     locK.181=c(0.35, 0.6), locK.183=c(0.5, 0.15))
myfreq

# convert to adegenet format
gpfreq <- freq.to.genpop(myfreq)
gpfreq

## Not run:
# If you have adegenet installed, you can now make this into a
# genpop object.
require(adegenet)
mygenpop <- genpop(gpfreq, ploidy=as.integer(4), type="codom")

# examine the object that has been created
mygenpop
popNames(mygenpop)
mygenpop@tab
mygenpop@all.names

# Perform a distance calculation with the object
dist.genpop(mygenpop)

## End(Not run)
```

---

genambig-class *Class "genambig"*

Description

Objects of this class store microsatellite datasets in which allele copy number is ambiguous. Genotypes are stored as a two-dimensional list of vectors, each vector containing all unique alleles for a given sample at a given locus. genambig is a subclass of gendata.

Objects from the Class

Objects can be created by calls of the form `new("genambig", samples, loci, ...)`. This automatically sets up a two-dimensional list in the `Genotypes` slot, with `dimnames=list(samples, loci)`. This array-list is initially populated with the missing data symbol. All other slots are given initial values according to the `initialize` method for `gendata`. Data can then be inserted into the slots using the replacement functions (see `Accessors`).

Slots

`Genotypes`**:** Object of class `"array"`. The first dimension of the array represents and is named by samples, while the second dimension represents and is named by loci. Each element of the array can contain a vector. Each vector should contain each unique allele for the genotype once. If an array element contains a vector of length 1 containing only the symbol that is in the `Missing` slot, this indicates missing data for that sample and locus.

`Description`**:** Object of class `"character"`. This stores a description of the dataset for the user's convenience.

`Missing`**:** Object of class `"ANY"`. A symbol to be used to indicate missing data in the `Genotypes` slot. This is the integer `-9` by default.

`Usatnts`**:** Object of class `"integer"`. A vector, named by loci. Each element indicates the repeat type of the locus. `2` indicates dinucleotide repeats, `3` indicates trinucleotide repeats, and so on. If the alleles stored in the `Genotypes` slot for a given locus are already written in terms of repeat number, the `Usatnts` value for that locus should be `1`. In other words, all alleles for a locus can be divided by the number in `Usatnts` to give alleles expressed in terms of relative repeat number.

`Ploidies`**:** Object of class `"integer"`. A vector, named by samples. This stores the ploidy of each sample. `NA` indicates unknown ploidy. See `Ploidies<-` and `estimatePloidy` for ways to fill this slot.

`PopInfo`**:** Object of class `"integer"`. A vector, named by samples, containing the population identity of each sample.

`PopNames`**:** Object of class `"character"`. A vector containing names for all populations. The position of a population name in the vector indicates the integer used to represent that population in `PopInfo`.

Extends

Class `"gendata"`, directly.

Methods

For more information on any of these methods, see the help files of their respective generic functions.

deleteLoci `signature(object = "genambig")`: Removes columns in the array in the `Genotypes` slot corresponding to the locus names supplied, then passes the arguments to the method for `gendata`.

deleteSamples `signature(object = "genambig")`: Removes rows in the array in the `Genotypes` slot corresponding to the sample names supplied, then passes the arguments to the method for `gendata`.

editGenotypes `signature(object = "genambig")`: Each vector in the `Genotypes` slot is placed into the row of a data frame, along with the sample and locus name for this vector. The data frame is then opened in the Data Editor so that the user can make changes. When the Data Editor window is closed, vectors are extracted back out of the data frame and written to the `Genotypes` slot.

estimatePloidy `signature(object = "genambig")`: Calculates the length of each genotype vector (excluding those with the missing data symbol), and creates a data frame showing the maximum and mean number of alleles per locus for each sample. This data frame is then opened in the Data Editor, where the user may edit ploidy levels. Once the Data Editor is closed, the `genambig` object is returned with the new values written to the `Ploidies` slot.

Genotype `signature(object = "genambig")`: Retrieves a single genotype vector, as specified by `sample` and `locus` arguments.

Genotype<- `signature(object = "genambig")`: Replaces a single genotype vector.

Genotypes `signature(object = "genambig")`: Retrieves a two-dimensional list of genotype vectors.

Genotypes<- `signature(object = "genambig")`: Replaces a one- or two-dimensional list of genotype vectors.

initialize `signature(.Object = "genambig")`: When `new` is called to create a new `genambig` object, the `initialize` method sets up a two dimensional list in the `Genotypes` slot indexed by sample and locus, and fills this list with the missing data symbol. The `initialize` method for `gendata` is then called.

isMissing `signature(object = "genambig")`: Given a set of samples and loci, each position in the array in the `Genotypes` slot is checked to see if it matches the missing data value. A single Boolean value or an array of Boolean values is returned.

Loci<- `signature(object = "genambig")`: For changing the names of loci. The names are changed in the second dimension of the array in the `Genotypes` slot, and then the `Loci<-` method for `gendata` is called.

Missing<- `signature(object = "genambig")`: For changing the missing data symbol. All elements of the `Genotypes` array that match the current missing data symbol are changed to the new missing data symbol. The `Missing<-` method for `gendata` is then called.

Samples<- `signature(object = "genambig")`: For changing the names of samples. The names are changed in the first dimension of the array in the `Genotypes` slot, and then the `Samples<-` method for `gendata` is called.

summary `signature(object = "genambig")`: Prints the dataset description (`Description` slot) to the console as well as the number of missing genotypes, then calls the `summary` method for `gendata`.

show `signature(object = "genambig")`: Prints the data to the console, formatted to make it more legible. The genotype for each locus is shown as the size of each allele, separated by a '/'. The SSR motif length ('Usatnts'), ploidies, population names, and population membership ('PopInfo') are displayed if they exist.

viewGenotypes `signature(object = "genambig")`: Prints a tab-delimited table of samples, loci, and genotype vectors to the console.

"[" `signature(x = "genambig", i = "ANY", j = "ANY")`: For subscipting `genambig` objects. Should be of the form `mygenambig[mysamples, myloci]`. Returns a `genambig` object. The `Genotypes`

slot is replaced by one containing only samples `i` and loci `j`. Likewise, the `PopInfo` and `Ploidies` slots are truncated to contain only samples `i`, and the `Usatnts` slot is truncated to contain only loci `j`. Other slots are left unaltered.

merge `signature(x = "genambig", y = "genambig")`: Merges two genotypes objects together. See `merge,genambig,genambig-method`.

Author(s)

Lindsay V. Clark, Tyler W. Smith

See Also

`gendata`, `Accessors`, `merge,genambig,genambig-method`

Examples

```
# display class definition
showClass("genambig")

# create a genambig object
mygen <- new("genambig", samples=c("a", "b", "c", "d"),
             loci=c("L1", "L2", "L3"))
# add some genotypes
Genotypes(mygen)[,"L1"] <- list(c(133, 139, 145), c(142, 154),
                                c(130, 142, 148), Missing(mygen))
Genotypes(mygen, loci="L2") <- list(c(105, 109, 113), c(111, 117),
                                    c(103, 115), c(105, 109, 113))
Genotypes(mygen, loci="L3") <- list(c(254, 258), Missing(mygen),
                                    c(246, 250, 262), c(250, 258))

# see a summary of the object
summary(mygen)
# display some of the genotypes
viewGenotypes(mygen[c("a", "b", "c"),])
```

---

genambig.to.genbinary *Convert Between Genotype Object Classes*

Description

These functions convert back and forth between the `genambig` and `genbinary` classes.

Usage

```
genambig.to.genbinary(object, samples = Samples(object),
                      loci = Loci(object))

genbinary.to.genambig(object, samples = Samples(object),
                      loci = Loci(object))
```

Arguments

object	The object containing the genetic dataset. A genambig object for genambig.to.genbinary, or a genbinary object for genbinary.to.genambig.
samples	An optional character vector indicating samples to include in the new object.
loci	An optional character vector indicating loci to include in the new object.

Details

The slots Description, Ploidies, Usatnts, PopNames, and PopInfo are transferred as-is from the old object to the new. The value in the Genotypes slot is converted from one format to the other, with preservation of allele names.

Value

For genambig.to.genbinary: a genbinary object containing all of the data from object. Missing, Present, and Absent are set at their default values.

For genbinary.to.genambig: a genambig object containing all of the data from object. Missing is at the default value.

Author(s)

Lindsay V. Clark

See Also

genambig, genbinary

Examples

```
# set up a genambig object for this example
mygen <- new("genambig", samples = c("A", "B", "C", "D"),
             loci = c("locJ", "locK"))
PopNames(mygen) <- c("PopQ", "PopR")
PopInfo(mygen) <- c(1,1,2,2)
Usatnts(mygen) <- c(2,2)
Genotypes(mygen, loci="locJ") <- list(c(178, 184, 186), c(174,186),
                                      c(182, 188, 190),
                                      c(182, 184, 188))
Genotypes(mygen, loci="locK") <- list(c(133, 135, 141),
                                      c(131, 135, 137, 143),
                                      Missing(mygen), c(133, 137))

# convert it to a genbinary object
mygenB <- genambig.to.genbinary(mygen)

# check the results
viewGenotypes(mygenB)
viewGenotypes(mygen)
PopInfo(mygenB)
```

```
# convert back to a genambig object
mygenA <- genbinary.to.genambig(mygenB)
viewGenotypes(mygenA)

# note: identical(mygen, mygenA) returns FALSE, because the alleles
# origninally input are not stored as integers, while the alleles
# produced by genbinary.to.genambig are integers.
```

---

genbinary-class *Class "genbinary"*

Description

This is a subclass of gendata that allows genotypes to be stored as a matrix indicating the presence and absence of alleles.

Objects from the Class

Objects can be created by calls of the form `new("genbinary", samples, loci, ...)`. After objects are initialized with sample and locus names, data can be added to slots using the replacement functions.

Slots

`Genotypes`**:** Object of class `"matrix"`. Row names of the matrix are sample names. Each column name is a locus name and an allele separated by a period (*e.g.* `"loc1.124"`); each column represents on allele. The number of alleles per locus is not limited and can be expanded even after entering initial data. Each element of the matrix must be equal to either `Present(object)`, `Absent(object)`, or `Missing(object)`. These symbols indicate, respectively, that a sample has an allele, that a sample does not have an allele, or that data for the sample at that locus are missing.

`Present`**:** Object of class `"ANY"`. The integer `1` by default. This symbol is used in the `Genotypes` slot to indicate the presence of an allele in a sample.

`Absent`**:** Object of class `"ANY"`. The integer `0` by default. This symbol is used in the `Genotypes` slot to indicate the absence of an allele in a sample.

`Description`**:** Object of class `"character"`. A character string or vector describing the dataset, for the convenience of the user.

`Missing`**:** Object of class `"ANY"`. The integer `-9` by default. This symbol is used in the `Genotypes` slot to indicate that data are missing for a given sample and locus.

`Usatnts`**:** Object of class `"integer"`. A vector, named by loci. This indicates the repeat length of each locus. `2` indicates dinucleotide repeats, `3` indicates trinucleotide repeats, and so on. If the alleles stored in the column names of the `Genotypes` slot for a given locus are already written in terms of repeat number, the `Usatnts` value for that locus should be `1`. In other words, all alleles for a locus can be divided by the number in `Usatnts` to give alleles expressed in terms of relative repeat number.

`Ploidies`: Object of class `"integer"`. A vector, named by samples. This indicates the ploidy of each sample.

`PopInfo`: Object of class `"integer"`. A vector, named by samples. This indicates the population identity of each sample.

`PopNames`: Object of class `"character"`. Names of each population. The position of the population name in the vector corresponds to the number used to represent that population in the `PopInfo` slot.

Extends

Class `"gendata"`, directly.

Methods

Absent `signature(object = "genbinary")`: Returns the symbol used to indicate that a given allele is absent in a given sample.

Absent<- `signature(object = "genbinary")`: Changes the symbol used to indicate that a given allele is absent in a given sample. The matrix in the `Genotypes` slot is searched for the old symbol, which is replaced by the new. The new symbol is then written to the `Absent` slot.

Genotype `signature(object = "genbinary")`: Returns a matrix containing the genotype for a given sample and locus (by a call to `Genotypes`).

Genotypes `signature(object = "genbinary")`: Returns the matrix stored in the `Genotypes` slot, or a subset as specified by the `samples` and `loci` arguments.

Genotypes<- `signature(object = "genbinary")`: A method for adding or replacing genotype data in the object. Note that allele columns cannot be removed from the matrix in the `Genotypes` slot using this method, although an entire column could be filled with zeros in order to effectively remove an allele from the dataset. If the order of rows in `value` (the matrix containing values to be assigned to the `Genotypes` slot) is not identical to `Samples(object)`, the `samples` argument should be used to indicate row order. Row names in `value` are ignored. The `loci` argument can be left at the default, even if only a subset of loci are being assigned. Column names of `value` are important, and should be the locus name and allele name separated by a period, as they are in the `Genotypes` slot. After checking that the column name is valid, the method checks for whether the column name already exists or not in the `Genotypes` slot. If it does exist, data from that column are replaced with data from `value`. If not, a column is added to the matrix in the `Genotypes` slot for the new allele. If the column is new and data are not being written for samples, the method automatically fills in `Missing` or `Absent` symbols for additional samples, depending on whether or not data for the locus appear to be missing for the sample or not.

initialize `signature(.Object = "genbinary")`: Sets up a `genbinary` object when `new("genbinary")` is called. If `samples` or `loci` arguments are missing, these are filled in with dummy values (`"ind1"`, `"ind2"`, `"loc1"`, `"loc2"`). The matrix is then set up in the `Genotypes` slot. Sample names are used for row names, and there are zero columns. The `initialize` method for `gendata` is then called.

Missing<- `signature(object = "genbinary")`: Replaces all elements in matrix in the `Genotypes` slot containing the old `Missing` symbol with the new `Missing` symbol. The method for `gendata` is then called to replace the value in the `Missing` slot.

Present `signature(object = "genbinary")`: Returns the symbol used to indicate that a given allele is present in a given sample.

Present<- `signature(object = "genbinary")`: Changes the symbol used for indicating that a given allele is present in a given sample. The symbol is first replaced in the `Genotypes` slot, and then in the `Present` slot.

Samples<- `signature(object = "genbinary")`: Changes sample names in the dataset. Changes the row names in the `Genotypes` slot, then calls the method for `gendata` to change the names in the `PopInfo` and `Ploidies` slots.

Loci<- `signature(object = "genbinary")`: Changes locus names in the dataset. Replaces the locus portion of the column names in the `Genotypes` slot, then calls the method for `gendata` to change the names in the `Usatnts` slot.

isMissing `signature(object = "genbinary")`: Returns Boolean values, by sample and locus, indicating whether genotypes are missing. If there are any missing data symbols within the genotype, it is considered missing.

summary `signature(object = "genbinary")`: Prints description of dataset and number of missing genotypes, then calls the method for `gendata` to print additional information.

editGenotypes `signature(object = "genbinary")`: Opens the genotype matrix in the Data Editor for editing. Useful for making minor changes, although allele columns cannot be added using this method.

viewGenotypes `signature(object = "genbinary")`: Prints the genotype matrix to the console, one locus at a time.

deleteSamples `signature(object = "genbinary")`: Removes the specified samples from the genotypes matrix, then calls the method for `gendata`.

deleteLoci `signature(object = "genbinary")`: Removes the specified loci from the genotypes matrix, then calls the method for `gendata`.

"[" `signature(x = "genbinary", i = "ANY", j = "ANY")`: Subscripting method. Returns a `genbinary` object with a subset of the samples and/or loci from `x`. Usage: `genobject[samples,loci]`.

estimatePloidy `signature(object = "genbinary")`: Creates a data frame of mean and maximum number of alleles per sample, which is then opened in the Data Editor so that the user can manually specify the ploidy of each sample. Ploidies are then written to the `Ploidies` slot of the object.

merge `signature(x = "genbinary", y = "genbinary")`: Merges two genotype objects together. See `merge,genbinary,genbinary-method`.

Author(s)

Lindsay V. Clark

See Also

`gendata`, `Accessors`, `genambig`

Examples

```
# show the class definition
showClass("genbinary")

# create a genbinary object
mygen <- new("genbinary", samples = c("indA", "indB", "indC", "indD"),
             loci = c("loc1", "loc2"))
Description(mygen) <- "Example genbinary object for the documentation."
Usatnts(mygen) <- c(2,3)
PopNames(mygen) <- c("Maine", "Indiana")
PopInfo(mygen) <- c(1,1,2,2)
Genotypes(mygen) <- matrix(c(1,1,0,0, 1,0,0,1, 0,0,1,1,
                             1,-9,1,0, 0,-9,0,1, 1,-9,0,1, 0,-9,1,1),
   nrow=4, ncol=7, dimnames = list(NULL,
   c("loc1.140", "loc1.144", "loc1.150",
     "loc2.97", "loc2.100", "loc2.106", "loc2.109")))

# view all of the data in the object
mygen
```

---

gendata-class *Class "gendata"*

Description

This is a superclass for other classes that contain population genetic datasets. It has slots for population information, ploidy, microsatellite repeat lengths, and a missing data symbol, but does not have a slot to store genotypes. Sample and locus names are stored as the `names` of vectors in the slots.

Objects from the Class

Objects can be created by calls of the form `new("gendata", samples, loci, ...)`. The missing data symbol will be set to `-9` by default. The default initial value for `PopNames` is a character vector of length 0, and for `Description` is the string `"Insert dataset description here"`. The default initial value for the `Ploidies` slot is a `"ploidymatrix"` object, containing a matrix filled with `NA` and named by samples in the first dimension and loci in the second dimension. For other slots, vectors filled with `NA` will be generated and will be named by samples (for `PopInfo`) or loci (for `Usatnts`). The slots can then be edited using the methods described below.

Note that in most cases you will want to instead create an object from one of `gendata`'s subclasses, such as `genambig`.

Slots

`Description`**:** Object of class `"character"`. One or more character strings to name or describe the dataset.

`Missing`**:** Object of class `"ANY"`. A value to indicate missing data in the genotypes of the dataset. `-9` by default.

`Usatnts`: Object of class `"integer"`. This vector must be named by locus names. Each element should be the length of the microsatellite repeat for that locus, given in nucleotides. For example, `2` would indicate a locus with dinucleotide repeats, and `3` would indicate a locus with trinucleotide repeats. `1` should be used for mononucleotide repeats OR if alleles for that locus are already expressed in terms of repeat number rather than nucleotides. To put it another way, if you divided the number used to represent an allele by the corresponding number in `Usatnts` (and rounded if necessary), the result would be the number of repeats (plus some additional length for flanking regions).

`Ploidies`: Object of class `"ploidysuper"`. This object will contain a `pld` slot that is a matrix named by samples and loci, a vector named by samples or loci, or a single value, depending on the subclass. Each element is an integer that represents ploidy. `NA` indicates unknown ploidy.

`PopInfo`: Object of class `"integer"`. This vector also must be named by sample names. Each element represents the number of the population to which each sample belongs.

`PopNames`: Object of class `"character"`. An unnamed vector containing the name of each population. If a number from `PopInfo` is used to index `PopNames`, it should find the correct population name. For example, if the first element of `PopNames` is `"ABC"`, then any samples with `1` as their `PopInfo` value belong to population `"ABC"`.

Methods

deleteLoci `signature(object = "gendata")`: Permanently remove loci from the dataset. This removes elements from `Usatnts`.

deleteSamples `signature(object = "gendata")`: Permanently remove samples from the dataset. This removes elements from `PopInfo` and `Ploidies`.

Description `signature(object = "gendata")`: Returns the character vector in the `Description` slot.

Description<- `signature(object = "gendata")`: Assigns a new value to the character vector in the `Description` slot.

initialize `signature(.Object = "gendata")`: This is called when the `new("gendata")` function is used. A new gendata object is created with sample and locus names used to index the appropriate slots.

Loci `signature(object = "gendata", usatnts = "missing", ploidies="missing")`: Returns a character vector containing all locus names for the object. The method accomplishes this by returning `names(object@Usatnts)`.

Loci `signature(object = "gendata", usatnts = "numeric", ploidies = "missing")`: Returns a character vector of all loci for a given set of repeat lengths. For example, if `usatnts = 2` all loci with dinucleotide repeats will be returned.

Loci `signature(object= "gendata", usatnts = "missing", ploidies = "numeric")`: Returns a character vector of all loci for a given set of ploidies. Only works if `object@Ploidies` is a `"ploidylocus"` object.

Loci `signature(object = "gendata", usatnts = "numeric",ploidies = "numeric")`: Returns a character vector of all loci that have one of the indicated repeat types and one of the indicated ploidies. Only works if `object@Ploidies` is a `"ploidylocus"` object.

Loci<- `signature(object = "gendata")`: Assigns new names to loci in the dataset (changes `names(object@Usants)`. Should not be used for adding or removing loci.

Missing `signature(object = "gendata")`: Returns the missing data symbol from `object@Missing`.

Missing<- `signature(object = "gendata")`: Assigns a new value to `object@Missing` (changes the missing data symbol).

Ploidies `signature(object = "gendata", samples = "ANY", loci = "ANY")`: Returns the ploidies in the dataset (`object@Ploidies`), indexed by sample and locus if applicable..

Ploidies<- `signature(object = "gendata")`: Assigns new values to ploidies of samples in the dataset. The assigned values are coerced to integers by the method. Names in the assigned vector or matrix are ignored; sample and/or locus names already present in the `gendata` object are used instead.

PopInfo `signature(object = "gendata")`: Returns the population numbers of samples in the dataset (`object@PopInfo`).

PopInfo<- `signature(object = "gendata")`: Assigns new population numbers to samples in the dataset. The assigned values are coerced to integers by the method. Names in the assigned vector are ignored; sample names already present in the `gendata` object are used instead.

PopNames `signature(object = "gendata")`: Returns a character vector of population names (`object@PopNames`).

PopNames<- `signature(object = "gendata")`: Assigns new names to populations.

PopNum `signature(object = "gendata",popname="character")`: Returns the number corresponding to a population name.

PopNum<- `signature(object = "gendata", popname = "character")`: Changes the population number for a given population name, merging it with an existing population of that number if applicable.

Samples `signature(object = "gendata", populations = "character", ploidies = "missing")`: Returns all sample names for a given set of population names.

Samples `signature(object = "gendata", populations = "character", ploidies = "numeric")`: Returns all sample names for a given set of population names and ploidies. Only samples that fit both criteria will be returned.

Samples `signature(object = "gendata", populations = "missing", ploidies = "missing")`: Returns all sample names.

Samples `signature(object = "gendata", populations = "missing", ploidies = "numeric")`: Returns all sample names for a given set of ploidies. Only works if `object@Ploidies` is a `"ploidysample"` object.

Samples `signature(object = "gendata", populations = "numeric", ploidies = "missing")`: Returns all sample names for a given set of population numbers.

Samples `signature(object = "gendata", populations = "numeric", ploidies = "numeric")`: Returns all sample names for a given set of population numbers and ploidies. Only samples that fit both criteria will be returned. Only works if `object@Ploidies` is a `"ploidysample"` object.

Samples<- `signature(object = "gendata")`: Assigns new names to samples. This edits both `names(object@PopInfo)` and `names(object@Ploidies)`. It should not be used for adding or removing samples from the dataset.

summary `signature(object = "gendata")`: Prints some informaton to the console, including the numbers of samples, loci, and populations, the ploidies present, and the types of microsatellite repeats present.

Usatnts `signature(object = "gendata")`: Returns microsatellite repeat lengths for loci in the dataset (`object@Usatnts`).

Usatnts<- `signature(object = "gendata")`: Assigns new values to microsatellite repeat lengths of loci (`object@Usatnts`). The assigned values are coerced to integers by the method. Names in the assigned vector are ignored; locus names already present in the `gendata` object are used instead.

"[" `signature(x = "gendata", i = "ANY", j = "ANY")`: Subscripts the data by a subset of samples and/or loci. Should be used in the format `mygendata[mysamples, myloci]`. Returns a `gendata` object with `PopInfo`, `Ploidies`, and `Usatnts` truncated to only contain the samples and loci listed in `i` and `j`, respectively. `Description`, `Missing`, and `PopNames` are left unaltered.

merge `signature(x = "gendata", y = "gendata")`: Merges two genotype objects. See merge,gendata,gendata-method

Author(s)

Lindsay V. Clark

See Also

genambig, genbinary, Accessors

Examples

```
# show class definition
showClass("gendata")

# create an object of the class gendata
# (in reality you would want to create an object belonging to one of the
# subclasses, but the procedure is the same)
mygen <- new("gendata", samples = c("a", "b", "c"),
             loci = c("loc1", "loc2"))
Description(mygen) <- "An example for the documentation"
Usatnts(mygen) <- c(2,3)
PopNames(mygen) <- c("PopV", "PopX")
PopInfo(mygen) <- c(2,1,2)
Ploidies(mygen) <- c(2,2,4,2,2,2)

# view a summary of the object
summary(mygen)
```

---

gendata.to.genind *Convert Data to genind Format*

Description

This is a function for exporting data to the package **adegenet**.

Usage

```
gendata.to.genind(object, samples = Samples(object), loci = Loci(object))
```

Arguments

object	A "genambig" or (preferably) a "genbinary" object.
samples	A character vector indicating the samples to include in the output.
loci	A character vector indicating the loci to include in the output.

Details

gendata.to.genind converts a "genambig" or "genbinary" object to a "genind" object using the package **adegenet**. Each individual must have a single ploidy. Ploidy and population information are carried over to the new object. Data will be coded as presence/absence in the new object. The locus names in the new object are locus and allele names seperated by a hyphen.

adegenet must be installed in order to use this function.

Value

A genetic dataset in the "genind" class, ready for use in **adegenet**.

Author(s)

Lindsay V. Clark

References

http://adegenet.r-forge.r-project.org/

Jombart, T. (2008) adegenet: a R package for the multivariate analysis of genetic markers. *Bioinformatics* **24**, 1403–1405.

See Also

freq.to.genpop

Examples

```
# create a "genambig" object
mydata <- new("genambig", samples=c("a","b","c","d"), loci=c("e","f"))
PopNames(mydata) <- c("G","H")
PopInfo(mydata) <- c(1,1,2,2)
mydata <- reformatPloidies(mydata, output="one")
Ploidies(mydata) <- 3
Genotypes(mydata, loci="e") <- list(c(100),c(100,102),
                                    c(98,102,104),c(102,106))
Genotypes(mydata, loci="f") <- list(c(200,202,204),Missing(mydata),
                                    c(210,212),c(204,210,214))

# convert to "genind"; not tested as it takes several seconds to load adegenet
```

```
if(require("adegenet")){
  mydata2 <- gendata.to.genind(mydata)
  mydata2@tab
  locNames(mydata2)
  indNames(mydata2)
  popNames(mydata2)
  pop(mydata2)
}
```

---

genIndex *Find All Unique Genotypes for a Locus*

Description

This function will return all unique genotypes for a given locus (ignoring allele order, but taking copy number into account) and return those genotypes as well as an index indicating which genotype(s) each individual has. This is a generic function with methods for "genambig" objects and for arrays. The array method is primarily intended for internal use with meandistance.matrix2, processing the output of genotypeProbs.

Usage

```
genIndex(object, locus)
```

Arguments

object	Typically, a "genambig" object. A two-dimentional list (array) can also be used here, where samples are in the first dimension and loci in the second dimension and each element of the list is output from genotypeProbs.
locus	A character string or integer indicating which locus to process.

Value

A list with two elements:

uniquegen	A list, where each element in the list is a vector indicating a unique genotype that was found.
genindex	For "genambig" objects, an integer vector, with one value per sample. This is the index of that sample's genotype in uniquegen. For arrays, a list with one element per sample. Each element is a vector of indices of that sample's possible genotypes in uniquegen, in the same order as in the genotypeProbs output.

Author(s)

Lindsay V. Clark

See Also

meandistance.matrix uses the "genambig" method internally.

.unal1loc, assignClones

Examples

```
data(simgen)
genIndex(simgen, 1)
```

---

genotypeDiversity *Genotype Diversity Statistics*

Description

genotypeDiversity calculates diversity statistics based on genotype frequencies, using a distance matrix to assign individuals to genotypes. The Shannon and Simpson functions are also available to calculate these statistics directly from a vector of frequencies.

Usage

```
genotypeDiversity(genobject, samples = Samples(genobject),
                  loci = Loci(genobject),
                  d = meandistance.matrix(genobject, samples, loci,
                                          all.distances = TRUE,
                                          distmetric = Lynch.distance),
                  threshold = 0, index = Shannon, ...)

Shannon(p, base = exp(1))

Simpson(p)

Simpson.var(p)
```

Arguments

genobject	An object of the class "genambig" (or more generally, "gendata" if a value is supplied to d). If there is more than one population, the PopInfo slot should be filled in. genobject is the dataset to be analyzed, although the genotypes themselves will not be used if d has already been calculated. Missing genotypes, however, will indicate individuals that should be skipped in the analysis.
samples	An optional character vector indicating a subset of samples to analyze.
loci	An optional character vector indicating a subset of loci to analyze.
d	A list such as that produced by meandistance.matrix or meandistance.matrix2 when all.distances = TRUE. The first item in the list is a three dimensional array, with the first dimension indexed by locus and the second and third dimensions indexed by sample. These are genetic distances between samples, by

	locus. The second item in the list is the distance matrix averaged across loci. This mean matrix will be used only if all loci are being analyzed. If `loci` is a subset of the loci found in `d`, the mean matrix will be recalculated.
`threshold`	The maximum genetic distance between two samples that can be considered to be the same genotype.
`index`	The diversity index to calculate. This should be `Shannon`, `Simpson`, or a user-defined function that takes as its first argument a vector of frequencies that sum to one.
`...`	Additional arguments to pass to `index`, for example the `base` argument for `Shannon`.
`p`	A vector of counts.
`base`	The base of the logarithm for calculating the Shannon index. This is `exp(1)` for the natural log, or `2` for log base 2.

Details

`genotypeDiversity` runs `assignClones` on distance matrices for individual loci and then for all loci, for each seperate population. The results of `assignClones` are used to calculate a vector of genotype frequencies, which is passed to `index`.

`Shannon` calculates the Shannon index, which is:

$$-\sum \frac{p_i}{N} \ln(\frac{p_i}{N})$$

(or log base 2 or any other base, using the `base` argument) given a vector p of genotype counts, where N is the sum of those counts.

`Simpson` calculates the Simpson index, which is:

$$\sum \frac{p_i(p_i-1)}{N(N-1)}$$

`Simpson.var` calculates the variance of the Simpson index:

$$\frac{4N(N-1)(N-2)\sum p_i^3 + 2N(N-1)\sum p_i^2 - 2N(N-1)(2N-3)(\sum p_i^2)^2}{[N(N-1)]^2}$$

The variance of the Simpson index can be used to calculate a confidence interval, for example the results of `Simpson` plus or minus twice the square root of the results of `Simpson.var` would be the 95% confidence interval.

Value

A matrix of diversity index results, with populations in rows and loci in columns. The final column is called `"overall"` and gives the results when all loci are analyzed together.

Author(s)

Lindsay V. Clark

References

Shannon, C. E. (1948) A mathematical theory of communication. *Bell System Technical Journal* **27**:379–423 and 623–656.

Simpson, E. H. (1949) Measurement of diversity. *Nature* **163**:688.

Lowe, A., Harris, S. and Ashton, P. (2004) *Ecological Genetics: Design, Analysis, and Application*. Wiley-Blackwell.

Arnaud-Haond, S., Duarte, M., Alberto, F. and Serrao, E. A. (2007) Standardizing methods to address clonality in population studies. *Molecular Ecology* **16**:5115–5139.

http://www.comparingpartitions.info/index.php?link=Tut4

See Also

assignClones, alleleDiversity

Examples

```
# set up dataset
mydata <- new("genambig", samples=c("a","b","c"), loci=c("F","G"))
Genotypes(mydata, loci="F") <- list(c(115,118,124),c(115,118,124),
                                    c(121,124))
Genotypes(mydata, loci="G") <- list(c(162,170,174),c(170,172),
                                     c(166,180,182))
Usatnts(mydata) <- c(3,2)

# get genetic distances
mydist <- meandistance.matrix(mydata, all.distances=TRUE)

# calculate diversity under various conditions
genotypeDiversity(mydata, d=mydist)
genotypeDiversity(mydata, d=mydist, base=2)
genotypeDiversity(mydata, d=mydist, threshold=0.3)
genotypeDiversity(mydata, d=mydist, index=Simpson)
genotypeDiversity(mydata, d=mydist, index=Simpson.var)
```

---

genotypeProbs *Calculate Probabilities of Unambiguous Genotypes*

Description

Given an ambiguous genotype and either a data frame of allele frequencies or a vector of genotype probabilities, genotypeProbs calculates all possible unambiguous genotypes and their probabilities of being the true genotype.

Usage

```
genotypeProbs(object, sample, locus, freq = NULL, gprob = NULL,
              alleles = NULL)
```

Arguments

object	An object of class "genambig". The Ploidies and PopInfo slots must be filled in for the sample and locus of interest.
sample	Number or character string indicating the sample to evaluate.
locus	Character string indicating the locus to evaluate.
freq	A data frame of allele frequencies, such as that produced by simpleFreq or deSilvaFreq. This argument should only be provided if the selfing rate is zero.
gprob	A vector of genotype probabilities based on allele frequencies and selfing rate. This is generated by meandistance.matrix2 and passed to genotypeProbs only if the selfing rate is greater than zero.
alleles	An integer vector of all alleles. This argument should only be used if gprob is also being used.

Details

This function is primarily designed to be called by meandistance.matrix2, in order to calculate distances between all possible unambiguous genotypes. Ordinary users won't use genotypeProbs unless they are designing a new analysis.

The genotype analyzed is Genotype(object, sample, locus). If the genotype is unambiguous (fully heterozygous or homozygous), a single unambiguous genotype is returned with a probability of one.

If the genotype is ambiguous (partially heterozygous), a recursive algorithm is used to generate all possible unambiguous genotypes (all possible duplications of alleles in the genotype, up to the ploidy of the individual.)

If the freq **argument is supplied:**

The probability of each unambiguous genotype is then calculated from the allele frequencies of the individual's population, under the assumption of random mating. Allele frequencies are normalized so that the frequencies of the alleles in the ambiguous genotype sum to one; this converts each frequency to the probability of the allele being present in more than one copy. The product of these probabilities is multiplied by the appropriate polynomial coefficient to calculate the probability of the unambiguous genotype.

$$p = \prod_{i=1}^{n} f_i^{c_i} * \frac{(k-n)!}{\prod_{i=1}^{n} c_i!}$$

where p is the probability of the unambiguous genotype, n is the number of alleles in the ambiguous genotype, f is the normalized frequency of each allele, c is the number of duplicated copies (total number of copies minus one) of the allele in the unambiguous genotype, and k is the ploidy of the individual.

If the gprob **and** alleles **arguments are supplied:**

The probabilities of all possible genotypes in the population have already been calculated, based on allele frequencies and selfing rate. This is done in meandistance.matrix2 using code from De Silva *et al.* (2005). Probabilities for the genotypes of interest (those that the ambiguous genotype could represent) are normalized to sum to 1, in order to give the conditional probabilities of the possible genotypes.

Value

`probs`	A vector containing the probabilities of each unambiguous genotype.
`genotypes`	A matrix. Each row represents one genotype, and the number of columns is equal to the ploidy of the individual. Each element of the matrix is an allele.

Author(s)

Lindsay V. Clark

References

De Silva, H. N., Hall, A. J., Rikkerink, E., and Fraser, L. G. (2005) Estimation of allele frequencies in polyploids under certain patterns of inheritance. *Heredity* **95**, 327–334

See Also

`meandistance.matrix2`, `GENLIST`

Examples

```
# get a data set and define ploidies
data(testgenotypes)
Ploidies(testgenotypes) <- c(8,8,8,4,8,8,rep(4,14))
# get allele frequencies
tfreq <- simpleFreq(testgenotypes)

# see results of genotypeProbs under different circumstances
Genotype(testgenotypes, "FCR7", "RhCBA15")
genotypeProbs(testgenotypes, "FCR7", "RhCBA15", tfreq)
Genotype(testgenotypes, "FCR10", "RhCBA15")
genotypeProbs(testgenotypes, "FCR10", "RhCBA15", tfreq)
Genotype(testgenotypes, "FCR1", "RhCBA15")
genotypeProbs(testgenotypes, "FCR1", "RhCBA15", tfreq)
Genotype(testgenotypes, "FCR2", "RhCBA23")
genotypeProbs(testgenotypes, "FCR2", "RhCBA23", tfreq)
Genotype(testgenotypes, "FCR3", "RhCBA23")
genotypeProbs(testgenotypes, "FCR3", "RhCBA23", tfreq)
```

`Internal Functions` *Internal Functions in polysat*

Description

The internal functions `G`, `INDEXG`, `GENLIST`, `RANMUL`, and `SELFMAT` are used for calculating genotype probabilities under partial selfing. The internal function `.unal1loc` finds all unique alleles at a single locus. The internal function `fixloci` converts locus names to a format that will be compatible as column headers for allele frequency tables.

Usage

```
G(q, n)
INDEXG(ag1, na1, m2)
GENLIST(ng, na1, m2)
RANMUL(ng, na1, ag, m2)
SELFMAT(ng, na1, ag, m2)
.unal1loc(object, samples, locus)
fixloci(loci, warn = TRUE)
```

Arguments

q	Integer.
n	Integer.
ag1	A vector representing an unambiguous genotype.
na1	Integer. The number of alleles, including a null.
m2	Integer. The ploidy.
ng	Integer. The number of genotypes.
ag	An array of genotypes such as that produced by .genlist.
object	A "genambig" object.
samples	Optional, a numeric or character vector indicating which samples to use.
locus	A character string or number indicating which locus to use.
loci	A character vector of locus names.
warn	Boolean indicating whether a warning should be issued if locus names are changed.

Value

G returns

$$\frac{(n+q)!}{(q+1)! * (n-1)!}$$

INDEXG returns an integer indicating the row containing a particular genotype in the matrix produced by GENLIST.

GENLIST returns an array with dimensions ng, m2, containing all possible unambiguous genotypes, one in each row. The null allele is the highest-numbered allele.

RANMUL returns a list. The first item is a vector of polynomial coefficients for calculating genotype frequencies under random mating. The second is an array showing how many copies of each allele each genotype has.

SELFMAT returns the selfing matrix. Parental genotypes are represented in rows, and offspring genotypes in columns. The numbers indicate relative amounts of offspring genotypes produced when the parental genotypes are self-fertilized.

.unal1loc returns a vector containing all unique alleles, not including Missing(object).

fixloci returns a character vector of corrected locus names.

Author(s)

Lindsay V. Clark

References

De Silva, H. N., Hall, A. J., Rikkerink, E., and Fraser, L. G. (2005) Estimation of allele frequencies in polyploids under certain patterns of inheritance. *Heredity* **95**, 327–334

See Also

deSilvaFreq, meandistance.matrix2, genotypeProbs, genambig.to.genbinary, alleleDiversity

Examples

```
# Calculation of genotype probabilities in a tetraploid with four
# alleles plus a null, and a selfing rate of 0.5.  This is a translation
# of code in the supplementary material of De Silva et al. (2005).
m2 <- 4
m <- m2/2
na1 <- 5
self <- 0.5
ng <- na1
for(j in 2:m2){
    ng <- ng*(na1+j-1)/j
}
ag <- polysat:::GENLIST(ng, na1, m2)
temp <- polysat:::RANMUL(ng, na1, ag, m2)
rmul <- temp[[1]]
arep <- temp[[2]]
rm(temp)
smat <- polysat:::SELFMAT(ng, na1, ag, m2)
smatdiv <- (polysat:::G(m-1,m+1))^2
p1 <- c(0.1, 0.4, 0.2, 0.2, 0.1) # allele frequencies

# GPROBS subroutine
rvec <- rep(0,ng)
for(g in 1:ng){
    rvec[g] <- rmul[g]
    for(j in 1:m2){
        rvec[g] <- rvec[g]*p1[ag[g,j]]
    }
}
id <- diag(nrow=ng)
smatt <- smat/smatdiv
s3 <- id - self * smatt
s3inv <- solve(s3)
gprob <- (1-self) * s3inv %*% rvec
# gprob is a vector of probabilities of the seventy genotypes.
```

isMissing *Determine Whether Genotypes Are Missing*

Description

isMissing returns Boolean values indicating whether the genotypes for a given set of samples and loci are missing from the dataset.

Usage

```
isMissing(object, samples = Samples(object), loci = Loci(object))
```

Arguments

object	An object of one of the subclasses of gendata, containing the genotypes to be tested.
samples	A character or numeric vector indicating samples to be tested.
loci	A character or numeric vector indicating loci to be tested.

Details

isMissing is a generic function with methods for genambig and genbinary objects.

For each genotype in a genambig object, the function evaluates and returns Genotype(object, sample, locus)[1] == Missing(object). For a genbinary object, TRUE %in% (Genotype(object, sample, locus) == Missing(object)) is returned for the genotype. If only one sample and locus are being evaluated, this is the Boolean value that is returned. If multiple samples and/or loci are being evaluated, the function creates an array of Boolean values and recursively calls itself to fill in the result for each element of the array.

Value

If both samples and loci are of length 1, a single Boolean value is returned, TRUE if the genotype is missing, and FALSE if it isn't. Otherwise, the function returns a named array with samples in the first dimension and loci in the second dimension, filled with Boolean values indicating whether the genotype for each sample*locus combination is missing.

Author(s)

Lindsay V. Clark

See Also

Missing, Missing<-, Genotype, find.missing.gen

Examples

```
# set up a genambig object for this example
mygen <- new("genambig", samples=c("a", "b"), loci=c("locD", "locE"))
Genotypes(mygen) <- array(list(c(122, 126), c(124, 128, 134),
                               Missing(mygen), c(156, 159)),
                          dim=c(2,2))
viewGenotypes(mygen)

# test if some individual genotypes are missing
isMissing(mygen, "a", "locD")
isMissing(mygen, "a", "locE")

# test an array of genotypes
isMissing(mygen, Samples(mygen), Loci(mygen))
```

Lynch.distance *Calculate Band-Sharing Dissimilarity Between Genotypes*

Description

Given two genotypes in the form of vectors of unique alleles, a dissimilarity is calculated as: 1 - (number of alleles in common)/(average number of alleles per genotype).

Usage

```
Lynch.distance(genotype1, genotype2, usatnt = NA, missing = -9)
```

Arguments

genotype1	A vector containing all alleles for a particular sample and locus. Each allele is only present once in the vector.
genotype2	A vector of the same form as genotype1, for another sample at the same locus.
usatnt	The microsatellite repeat length for this locus (ignored by the function).
missing	The symbol used to indicate missing data in either genotype vector.

Details

Lynch (1990) defines a simple measure of similarity between DNA fingerprints. This is 2 times the number of bands that two fingerprints have in common, divided by the total number of bands that the two genotypes have. Lynch.distance returns a dissimilarity, which is 1 minus the similarity.

Value

If the first element of either or both input genotypes is equal to missing, NA is returned.

Otherwise, a numerical value is returned. This is one minus the similarity. The similarity is calculated as the number of alleles that the two genotypes have in common divided by the mean length of the two genotypes.

Author(s)

Lindsay V. Clark

References

Lynch, M. (1990) The similarity index and DNA fingerprinting. *Molecular Biology and Evolution* **7**, 478-484.

See Also

Bruvo.distance, meandistance.matrix

Examples

```
Lynch.distance(c(100,102,104), c(100,104,108))
Lynch.distance(-9, c(102,104,110))
Lynch.distance(c(100), c(100,104,106))
```

---

meandist.from.array *Tools for Working With Pairwise Distance Arrays*

Description

meandist.from.array produces a mean distance matrix from an array of pairwise distances by locus, such as that produced by meandistance.matrix when all.distances=TRUE. find.na.dist finds missing distances in such an array, and find.na.dist.not.missing finds missing distances that aren't the result of missing genotypes.

Usage

```
meandist.from.array(distarray, samples = dimnames(distarray)[[2]],
loci = dimnames(distarray)[[1]])

find.na.dist(distarray, samples = dimnames(distarray)[[2]],
loci = dimnames(distarray)[[1]])

find.na.dist.not.missing(object, distarray,
samples = dimnames(distarray)[[2]], loci = dimnames(distarray)[[1]])
```

Arguments

distarray	A three-dimensional array of pairwise distances between samples, by locus. Loci are represented in the first dimension, and samples are represented in the second and third dimensions. Dimensions are named accordingly. Such an array is the first element of the list produced by meandistance.matrix if all.distances=TRUE.
samples	Character vector. Samples to analyze.
loci	Character vector. Loci to analyze.
object	A genambig object. Typically the genotype object that was used to produce distarray.

Details

`find.na.dist.not.missing` is primarily intended to locate distances that were not calculated by `Bruvo.distance` because both genotypes had too many alleles (more than `maxl`). The user may wish to estimate these distances manually and fill them into the array, then recalculate the mean matrix using `meandist.from.array`.

Value

`meandist.from.array` returns a matrix, with both rows and columns named by samples, of distances averaged across loci.

`find.na.dist` and `find.na.dist.not.missing` both return data frames with three columns: Locus, Sample1, and Sample2. Each row represents the index in the array of an element containing NA.

Author(s)

Lindsay V. Clark

See Also

`meandistance.matrix`, `Bruvo.distance`, `find.missing.gen`

Examples

```
# set up the genotype data
samples <- paste("ind", 1:4, sep="")
samples
loci <- paste("loc", 1:3, sep="")
loci
testgen <- new("genambig", samples=samples, loci=loci)
Genotypes(testgen, loci="loc1") <- list(c(-9), c(102,104),
                                        c(100,106,108,110,114),
                                        c(102,104,106,110,112))
Genotypes(testgen, loci="loc2") <- list(c(77,79,83), c(79,85), c(-9),
                                        c(83,85,87,91))
Genotypes(testgen, loci="loc3") <- list(c(122,128), c(124,126,128,132),
                                        c(120,126), c(124,128,130))
Usatnts(testgen) <- c(2,2,2)

# look up which samples*loci have missing genotypes
find.missing.gen(testgen)

# get the three-dimensional distance array and the mean of the array
gendist <- meandistance.matrix(testgen, distmetric=Bruvo.distance,
                               maxl=4, all.distances=TRUE)
# look at the distances for loc1, where there is missing data and long genotypes
gendist[[1]]["loc1",,]

# look up all missing distances in the array
find.na.dist(gendist[[1]])
```

```
# look up just the missing distances that don't result from missing genotypes
find.na.dist.not.missing(testgen, gendist[[1]])

# Copy the array to edit the new copy
newDistArray <- gendist[[1]]
# calculate the distances that were NA from genotype lengths exceeding maxl
# (in reality, if this were too computationally intensive you might estimate
# it manually instead)
subDist <- Bruvo.distance(c(100,106,108,110,114), c(102,104,106,110,112))
subDist
# insert this distance into the correct positions
newDistArray["loc1","ind3","ind4"] <- subDist
newDistArray["loc1","ind4","ind3"] <- subDist
# calculate the new mean distance matrix
newMeanMatrix <- meandist.from.array(newDistArray)
# look at the difference between this matrix and the original.
newMeanMatrix
gendist[[2]]
```

`meandistance.matrix` *Mean Pairwise Distance Matrix*

Description

Given a genambig object, `meandistance.matrix` produces a symmetrical matrix of pairwise distances between samples, averaged across all loci. An array of all distances prior to averaging may also be produced.

Usage

```
meandistance.matrix(object, samples = Samples(object),
                    loci = Loci(object), all.distances=FALSE,
                    distmetric = Bruvo.distance, progress = TRUE,
                    ...)
meandistance.matrix2(object, samples = Samples(object),
                     loci = Loci(object),
                     freq = simpleFreq(object, samples, loci), self = 0,
                     all.distances = FALSE, distmetric = Bruvo.distance,
                     progress = TRUE, ...)
```

Arguments

`object`	A genambig object containing the genotypes to be analyzed. If `distmetric = Bruvo.distance`, the `Usatnts` slot should be filled in. For `meandistance.matrix2`, `Ploidies` and `PopInfo` are also required.
`samples`	A character vector of samples to be analyzed. These should be all or a subset of the sample names used in `object`.
`loci`	A character vector of loci to be analyzed. These should be all or a subset of the loci names used in `object`.

freq	A data frame of allele frequencies such as that produced by simpleFreq or deSilvaFreq.
self	A number ranging from 0 to 1, indicating the rate of selfing.
all.distances	If FALSE, only the mean distance matrix will be returned. If TRUE, a list will be returned containing an array of all distances by locus and sample as well as the mean distance matrix.
distmetric	The function to be used to calculate distances between genotypes. Bruvo.distance, Lynch.distance, or a distance function written by the user.
progress	If TRUE, loci and samples will be printed to the console as distances are calculated, so that the user can monitor the progress of the computation.
...	Additional arguments (such as maxl, add, and loss) to pass to distmetric.

Details

Each distance for the three-dimensional array is calculated only once, to save computation time. Since the array (and resulting mean matrix) is symmetrical, the distance is written to two positions in the array at once.

meandistance.matrix uses ambiguous genotypes exactly as they are, whereas meandistance.matrix2 uses genotypeProbs to calculate all possible unambiguous genotypes and their probabilities under random mating or partial selfing. The distance between each possible pair of unambiguous genotypes for the two samples is calculated with distmetric and weighted by the product of the probabilities of the two gentoypes. As you might expect, meandistance.matrix2 takes longer to process a given "genambig" object than meandistance.matrix does. Additionally, the distance between two identical ambiguous genotypes will be zero when calculated with meandistance.matrix, and greater than zero when calculated with meandistance.matrix2, due to potential differences in copy number of the alleles.

When Bruvo.distance is used, meandistance.matrix2 exaggerates distances between individuals of different ploidy as compared to meandistance.matrix. The use of Bruvo2.distance with meandistance.matrix2 allows individuals with different ploidies to have similar inter-individual distances to those between individuals of the same ploidy. In general, it will be desirable to use Bruvo.distance with meandistance.matrix for complex datasets with high ploidy levels, or Bruvo.distance2 with meandistance.matrix2 for hexaploid or lower datasets (based on how long it takes my personal computer to perform these calculations) where changes in ploidy are due to genome doubling or genome loss. If all individuals have the same ploidy, Bruvo.distance and Bruvo2.distance will give identical results regardless of whether meandistance.matrix or meandistance.matrix2 is used.

meandistance.matrix2 does not allow a genotype to have more alleles than the ploidy of the individual (as listed in the Ploidies slot). Additionally, if self is greater than zero, each population may only have one ploidy at each locus.

Value

A symmetrical matrix containing pairwise distances between all samples, averaged across all loci. Row and column names of the matrix will be the sample names provided in the samples argument. If all.distances=TRUE, a list will be produced containing the above matrix as well as a three-dimensional array containing all distances by locus and sample. The array is the first item in the list, and the mean matrix is the second.

Author(s)

Lindsay V. Clark

See Also

Bruvo.distance, Bruvo2.distance, Lynch.distance, meandist.from.array, GENLIST

Examples

```
# create a list of genotype data
mygendata <- new("genambig", samples = c("ind1","ind2","ind3","ind4"),
                 loci = c("locus1","locus2","locus3","locus4"))
Genotypes(mygendata) <-
  array(list(c(124,128,138),c(122,130,140,142),c(122,132,136),c(122,134,140),
             c(203,212,218),c(197,206,221),c(215),c(200,218),
             c(140,144,148,150),c(-9),c(146,150),c(152,154,158),
             c(233,236,280),c(-9),c(-9),c(-9)))
Usatnts(mygendata) <- c(2,3,2,1)

# make index vectors of data to use
myloci <- c("locus1","locus2","locus3")
mysamples <- c("ind1","ind2","ind4")

# calculate array and matrix
mymat <- meandistance.matrix(mygendata, mysamples, myloci,
                             all.distances=TRUE)
# view the results
mymat[[1]]["locus1",,]
mymat[[1]]["locus2",,]
mymat[[1]]["locus3",,]
mymat[[2]]

# add addtional info needed for meandistance.matrix2
mygendata <- reformatPloidies(mygendata, output="one")
Ploidies(mygendata) <- 4
PopInfo(mygendata) <- c(1,1,1,1)

# calculate distances taking allele freqs into account
mymat2 <- meandistance.matrix2(mygendata, mysamples, myloci)
mymat2
# now do the same under selfing
mymat3 <- meandistance.matrix2(mygendata, mysamples, myloci, self=0.3)
mymat3
```

---

merge-methods *Merge Two Genotype Objects into One*

Description

The generic function merge has methods defined in polysat to merge two genotype objects of the same class. Each method has optional samples and loci arguments for specifying subsets of samples and loci to be included in the merged object. Each method also has an optional overwrite argument to specify which of the two objects should not be used in the case of conflicting data.

Usage

```
merge(x, y, ...)
```

Arguments

x	One of the objects to be merged. For the methods defined for **polysat** this should be of class "gendata" or one of its subclasses.
y	The other object to be merged. Should be of the same class as x. y@Ploidies must also be of the same class as x@Ploidies.
...	Additional arguments specific to the method.

Methods

The methods for merge in polysat have four additional arguments: objectm, samples, loci, overwrite.

The samples and loci arguments can specify, using character vectors, a subset of the samples and loci found in x and y to write to the object that is returned.

If overwrite = "x", data from the second object will be used wherever there is contradicting data. Likewise if overwrite = "y", data from the first object will be used wherever there is contradicting data. If no overwrite argument is given, then any contradicting data between the two objects will produce an error indicating where the contradicting data were found.

The objectm argument is primarily for internal use (most users will not need it). If this argument is not provided, a new genotype object is created and data from x and y are written to it. If objectm is provided, this is the object to which data will be written, and the object that will be returned.

signature(x = "genambig", y = "genambig") This method merges the genotype data from x and y. If the missing data symbols differ between the objects, overwrite is used to determine which missing data symbol to use, and all missing data symbols in the overwritten object are converted. If overwrite is not provided and the missing data symbols differ between the objects, an error will be given. The genotypes are then filled in. If certain sample*locus combinations do not exist in either object (x and y have different samples as well as different loci), missing data symbols are left in these positions. Again, for genotypes, overwrite determines which object to preferentially use for data and whether to give an error if there is a disagreement.

The merge method for gendata is then called.

signature(x = "genbinary", y = "genbinary") This method also merges genotype data for x and y, then calls the method for gendata. Missing, Present, and Absent are checked for consistency between objects similarly to what happens with Missing in the genambig method. Genotypes are then written to the merged object, and consistency between genotypes is checked.

`signature(x = "gendata", y = "gendata")` This method merges data about ploidy, repeat length, and population identity, as well as writing one or both dataset descriptions to the merged object.

The same population numbers can have different meanings in `PopInfo(x)` and `PopInfo(y)`. The unique `PopNames` are used instead to determine population identity, and the `PopInfo` numbers are changed if necessary. Therefore, it is important for identical populations to be named the same way in both objects, but not important for identical populations to have the same number in both objects.

`mergeAlleleAssignments`

Merge Allele Assignment Matrices

Description

In cases where multiple populations are used separately to assign alleles to homeologous loci, `mergeAlleleAssignments` is used to consolidate the results.

Usage

```
mergeAlleleAssignments(x)
```

Arguments

`x`	A list, where each element is in the format of results produced by `catalanAlleles` or `testAlGroups`; see example.

Value

A list in similar format to `x`, but with only one element per locus.

Author(s)

Lindsay V. Clark

See Also

`testAlGroups`, `catalanAlleles`, `recodeAllopoly`

Examples

```
# List of allele assignment results for this example; normally these
# would be produced by other functions evaluating a genetic dataset.
# The example below is for an allotetraploid.

# Locus L1 is a well-behaved locus with no homoplasy.
myresults <- list(list(locus="L1", SGploidy=2,
                  assignments=matrix(c(1,0,0,1,1,0,0,1,0,1), nrow=2,
```

```
                    ncol=5, dimnames=list(NULL,
                    c("124","128","130","134","138"))),
                    list(locus="L1", SGploidy=2,
                    assignments=matrix(c(0,1,1,0,1,0,0,1), nrow=2, ncol=4,
                    dimnames=list(NULL, c("124","128","132","140"))),
                    list(locus="L1", SGploidy=2,
                    assignments=matrix(c(0,1,1,0,0,1,1,0), nrow=2, ncol=4,
                    dimnames=list(NULL, c("126","128","130","132")))),
# Locus L2 is unresolvable because there are no shared alleles between
# populations.
                    list(locus="L2", SGploidy=2,
                    assignments=matrix(c(1,0,1,0,0,1), nrow=2, ncol=3,
                    dimnames=list(NULL, c("205","210","225"))),
                    list(locus="L2", SGploidy=2,
                    assignments=matrix(c(1,0,0,1,0,1,1,0), nrow=2, ncol=4,
                    dimnames=list(NULL, c("195","215","220","230"))),
# Locus L3 has homoplasy that makes it unresolvable.
                    list(locus="L3", SGploidy=2,
                    assignments=matrix(c(1,0,0,1,0,1,1,0), nrow=2, ncol=4,
                    dimnames=list(NULL, c("153","159","168","171"))),
                    list(locus="L3", SGploidy=2,
                    assignments=matrix(c(1,0,0,1,1,0,0,1), nrow=2, ncol=4,
                    dimnames=list(NULL, c("153","156","165","171"))),
# Locus L4 has homoplasy, but the results can still be merged.
                    list(locus="L4", SGploidy=2,
                    assignments=matrix(c(1,0,1,0,0,1,0,1,0,1), nrow=2, ncol=5,
                    dimnames=list(NULL, c("242","246","254","260","264"))),
                    list(locus="L4", SGploidy=2,
                    assignments=matrix(c(1,0,0,1,1,0,0,1,0,1), nrow=2, ncol=5,
                    dimnames=list(NULL, c("242","246","250","254","260"))))
                    )
myresults

# merge within loci
mergedresults <- mergeAlleleAssignments(myresults)
mergedresults
```

PIC *Polymorphic Information Content*

Description

Given a set of allele frequencies, this function estimates the Polymorphic Information Content (PIC) for each locus, within and/or across populations.

Usage

```
PIC(freqs, pops = row.names(freqs), loci = unique(as.matrix(as.data.frame(
    strsplit(names(freqs), split = ".", fixed = TRUE),
    stringsAsFactors = FALSE))[1, ]), bypop = TRUE, overall = TRUE)
```

Arguments

freqs	A data frame of allele frequencies, such as those output by simpleFreq and deSilvaFreq.
pops	An optional characer vector containing names of populations to include.
loci	An optional character vector containing names of loci to include.
bypop	If TRUE, PIC will be estimated separately for each population.
overall	If TRUE, mean allele frequencies will be estimated across all populations (weighted by population size) and used to estimate overall PIC values for each locus.

Details

PIC is estimated as:

$$1 - (\sum_{i=1}^{n} p_i^2) - \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2p_i^2 p_j^2$$

according to Botstein et al. (1980), where p_i and p_j are allele frequencies at alleles *i* and *j*, respectively, and n is the number of alleles.

The higher this value is, the more useful a marker is for distinguishing individuals and understanding relationships among them.

Value

A matrix, with loci in columns, and populations and/or "Overall" in rows. Each element of the matrix contains a PIC value.

Author(s)

Lindsay V. Clark

References

Botstein, M., White, R. L., Skolnick, M. and Davis, R. W. (1980) Construction of a genetic linkage map in man using restriction fragment length polymorphisms. *American Journal of Human Genetics* **32**, 314–331.

See Also

alleleDiversity

Examples

```
# generate allele frequencies for this example
myfreq <- data.frame(row.names = c("pop1", "pop2"),
                     Genomes = c(20,30),
                     loc1.124 = c(0.1, 0.25),
                     loc1.126 = c(0.2, 0),
                     loc1.128 = c(0.05, 0.4),
```

```
                    loc1.130 = c(0.3, 0.1),
                    loc1.132 = c(0.1, 0.1),
                    loc1.134 = c(0.25, 0.15),
                    loc2.150 = c(0.4, 0.5),
                    loc2.155 = c(0.3, 0.2),
                    loc2.160 = c(0.3, 0.3))

# estimate PIC
PIC(myfreq)
```

`pld` *Accessor, Replacement, and Manipulation Functions for* `"ploidysuper"` *Objects*

Description

`pld` accesses and replaces the `pld` slot of objects of `"ploidysuper"` subclasses. `plCollapse` tests whether an object of one of these classes can be converted to an object of a simpler one of these classes, and optionally returns the converted object. These are generic functions with methods for the subclasses of `"ploidysuper"`. These functions are primarily for internal use.

Usage

```
pld(object, samples, loci)
pld(object) <- value
plCollapse(object, na.rm, returnvalue)
```

Arguments

`object`	A `"ploidysuper"` object.
`samples`	An optional character or numeric vector indexing the samples for which to return ploidy values.
`loci`	An optional character or numeric vector indexing the loci for which to return ploidy values.
`value`	A numeric vector or matrix that can be coerced to integers. These represent the ploidies to store in the `object@pld` slot.
`na.rm`	Boolean. If `TRUE`, `NA` values are ignored when testing to see if the ploidy format can be simplified. If the sample, locus, or entire dataset all has one ploidy aside from `NA`, the `NA` values will be overwritten by that ploidy when simplifying the ploidy format. If `FALSE`, `NA` is treated as a unique ploidy.
`returnvalue`	Boolean. If `TRUE`, a `"ploidysuper"` object will be returned if the ploidy format can be simplified, and `FALSE` will be returned if it cannot be simplified. If `FALSE`, only `TRUE` or `FALSE` will be returned to indicate if the ploidy format can be simplified or not.

Value

`pld` returns the vector or matrix containing the ploidy values. This is the contents of `object@pld`.

`plCollapse` either returns a Boolean value indicating whether the ploidy can be changed to a simpler format, or a new `"ploidysuper"` object with all of the ploidy data of `object` put into a simpler format. If `object` is a `"ploidymatrix"` object, a `"ploidysample"`, `"ploidylocus"`, or `"ploidyone"` object can be returned depending on how many unique ploidy values there are and how they are distributed. If `object` is a `"ploidysample"` or `"ploidylocus"` object, a `"ploidyone"` object can be returned.

Author(s)

Lindsay V. Clark

See Also

`reformatPloidies`, `Ploidies`

Examples

```
test <- new("ploidymatrix", samples=c("a","b","c"),
            loci=c("l1","l2","l3"))
pld(test)       # view the ploidies
pld(test) <- 2  # make it diploid at all samples and loci
pld(test)["a",] <- c(2,4,4) # change the ploidies for sample a
pld(test, samples=c("a","b")) # view ploidies at a subset of samples

# test to see if the ploidies can be simplified
p <- plCollapse(test, na.rm=FALSE, returnvalue=TRUE)
p
# now change a ploidy and repeat the test
pld(test)["a","l1"] <- 4
p <- plCollapse(test, na.rm=FALSE, returnvalue=TRUE)
p
# change something else and collapse it further
pld(p)["a"] <- 2
p2 <- plCollapse(p, na.rm=FALSE, returnvalue=TRUE)
p2

# if na.rm=FALSE, NA values are not ignored:
pld(test)["a","l1"] <- NA
pld(test)
plCollapse(test, na.rm=FALSE, returnvalue=TRUE)
# NA values are ignored with na.rm=TRUE
plCollapse(test, na.rm=TRUE, returnvalue=TRUE)
```

ploidysuper-class *Class* "ploidysuper" *and Subclasses*

Description

These classes contain ploidy data indexed by sample, locus, both, or neither. They are intended to go in the Ploidies slot of "gendata" objects.

Objects from the Class

"ploidysuper" is a virtual class: No objects may be created from it.

Objects of the subclasses "ploidymatrix", "ploidysample", "ploidylocus", and "ploidyone" can be created with the call new(ploidyclass, samples, loci, ...), where ploidyclass is a character string of one of the class names, and samples and loci are character vectors naming samples and loci, respectively. The latter two arguments are optional depending on the class (whether ploidies are indexed by sample and/or locus). The typical user will not have to create an object in this way, because other functions in **polysat** will do it for you.

Slots

pld**:** The only slot for objects of these classes. For "ploidymatrix", this is a matrix of integers, indexed in the first dimension by sample name and in the second dimension by locus name. Each element represents the ploidy at a given sample and locus. For "ploidysample" and "ploidylocus", the slot is an integer vector, named by sample or locus and indicating the ploidy at each sample or locus, respectively. For "ploidyone", the slot contains a single integer representing the ploidy for the entire dataset.

Methods

pld signature(object= "ploidymatrix"): Returns the contents of object@pld: a matrix of ploidies indexed by sample and locus. The samples and loci arguments can be used, optionally, to only return a subset of ploidies.

Author(s)

Lindsay V. Clark

See Also

reformatPloidies, pld, plCollapse, gendata

Examples

```
showClass("ploidysuper")
```

plotSSAllo *Perform Allele Assignments across Entire Dataset*

Description

processDatasetAllo runs alleleCorrelations on every locus in a "genambig" object, then runs testAlGroups on every locus using several user-specified parameter sets. It chooses a single best set of allele assignments for each locus, and produces plots to help the user evaluate assignment quality. plotSSAllo assists the user in evaluating the quality of allele assignments by plotting the results of K-means clustering. plotParamHeatmap assists the user in choosing the best parameter set for testAlGroups for each locus.

Usage

```
plotSSAllo(AlCorrArray)
plotParamHeatmap(propMat, popname = "AllInd", col = grey.colors(12)[12:1], main = "")
processDatasetAllo(object, samples = Samples(object), loci = Loci(object),
                   n.subgen = 2, SGploidy = 2, n.start = 50, alpha = 0.05,
                parameters = data.frame(tolerance    = c(0.05, 0.05,  0.05, 0.05),
                                        swap         = c(TRUE, FALSE, TRUE, FALSE),
                                          null.weight   = c(0.5,  0.5,   0,    0)),
                  plotsfile = "alleleAssignmentPlots.pdf", usePops = FALSE, ...)
```

Arguments

AlCorrArray	A two-dimensional list, where each item in the list is the output of alleleCorrelations. The first dimension represents loci, and the second dimension represents populations. Both dimensions are named. This is the $AlCorrArray output of processDatasetAllo.
propMat	A two-dimensional array, with loci in the first dimension and parameter sets in the second dimension, indicating the proportion of alleles that were found to be homoplasious by testAlGroups or the proportion of genotypes that could not be recoded using a given set of allele assignments. This can be the $propHomoplasious output of processDatasetAllo, indexed by a single population. If a three-dimensional array is provided, it will be indexed in the second dimension by popname. The $propHomoplMerged or $missRate output of processDatasetAllo may also be passed to this argument.
popname	The name of the population corresponding to the data in propMat.
col	The color scale for representing the proportion of loci that are homoplasious or the proportion of genotypes that are missing.
main	A title for the plot.
object	A "genambig" object.
samples	An optional character vector indicating which samples to include in analysis.
loci	An optional character vector indicating which loci to include in analysis.

n.subgen	The number of isoloci into which each locus should be split. Passed directly to alleleCorrelations.
SGploidy	The ploidy of each isolocus. Passed directly to testAlGroups.
n.start	Passed directly to the nstart argument of kmeans. See alleleCorrelations.
alpha	The significance threshold for determining whether two alleles are significantly correlated. Used primarily for identifying potentially problematic positive correlations. Passed directly to alleleCorrelations.
parameters	Data frame indicating parameter sets to pass to testAlGroups. Each row is one set of parameters.
plotsfile	A PDF output file name for drawing plots to help assess assignment quality. Can be NULL if no plots are desired.
usePops	If TRUE, population assignments are taken from the PopInfo slot of object, and populations are analyzed separately with alleleCorrelations and testAlGroups, before merging the results with mergeAlleleAssignments.
...	Additional parameters to pass to testAlGroups for adjusting the simulated annealing algorithm.

Details

plotSSAllo produces a plot of loci by population, with the sums-of-squares ratio on the x-axis and the evenness of allele distribution on the y-axis (see Value). Locus names are written directly on the plot. If there are multiple population names, locus names are colored by population, and a legend is provided for colors. Loci with high-quality allele clustering are expected to be in the upper-right quadrant of the plot. If locus names are in italics, it indicates that positive correlations were found between some alleles, indicating population structure or scoring error that could interfere with assignment quality.

plotParamHeatmap produces an image to indicate the proportion of alleles found to be homoplasious, or the proportion of genotypes that could not be unambiguously recoded using allele assignments, for each locus and parameter set for a given population (when looking at homoplasy) or merged across populations (for homoplasy or the proportion of non-recodeable genotypes). Darker colors indicate more homoplasy or more genotypes that could not be recoded. The word "best" indicates, for each locus, the parameter set that found the least homoplasy or smallest number of non-recodeable genotypes.

By default, processDatasetAllo generates a PDF file containing output from plotSSAllo and plotParamHeatmap, as well as heatmaps of the $heatmap.dist output of alleleCorrelations for each locus and population. Heatmaps are not plotted for loci where an allele is present in all individuals. processDatasetAllo also generates a list of R objects containing allele assignments under different parameters, as well as statistics for evaluating clustering quality and choosing the optimal parameter sets, as described below.

Value

plotSSAllo draws a plot and invisibly returns a list:

ssratio	A two-dimensional array with loci in the first dimension and populations in the second dimension. Each value is the sums-of-squares between isoloci divided

by the total sums-of-squares, as output by K-means clustering. If K-means clustering was not performed, the value is zero.

`evenness` An array of the same dimensions as `$ssratio`, containing values to indicate how evenly alleles are distributed among isoloci as determined by K-means clustering. This is:

$$1 - \sum_{1}^{i} (\frac{a_i}{A})^2$$

where i is the number of isoloci, a_i is the number of alleles for a given isolocus, and A is the total number of alleles for the locus.

`max.evenness` The maximum possible value for `$evenness`, given the number of isoloci.

`min.evenness` The minimum possible value for `$evenness`, given the number of isoloci and alleles.

`posCor` An array of the same dimensions as `$ssratio`, containing `TRUE` if there were any positive correlations between alleles, and `FALSE` if not.

`processDatasetAllo` returns a list:

`AlCorrArray` A two-dimensional list with loci in the first dimension and populations in the second dimension, giving the results of `alleleCorrelations`.

`TAGarray` A three-dimensional list with loci in the first dimension, populations in the second dimension, and parameter sets in the third dimension, giving the results of `testAlGroups`.

`plotSS` The output of `plotSSAllo`.

`propHomoplasious`
A three-dimensional array, with the same dimensions as `$TAGarray`, indicating the proportion of alleles that were found to be homoplasious for each locus, population, and parameter set.

`mergedAssignments`
A two-dimensional list, with loci in the first dimension and parameter sets in the second dimension, containing allele assignments merged across populations. This is the output of `mergeAlleleAssignments`.

`propHomoplMerged`
A two-dimensional array, of the same dimensions as `$mergedAssignments`, indicating the proportion of alleles that were homoplasious, for each locus and parameter set, for allele assignments that were merged across populations.

`missRate` A matrix with the same dimensions as `$mergedAssignments` indicating the proportion of non-missing genotypes from the original dataset that cannot be unambiguously recoded, without invoking aneuploidy, using the merged allele assignments from each parameter set for each locus.

`bestAssign` A one-dimensional list with a single best set of allele assignments, from `$mergedAssignments`, for each locus. The best set of assignments is chosen using `$missRate`, then in the case of a tie using `$propHomoplMerged`, then in the case of a tie using the parameter set that was listed first.

`plotParamHeatmap` draws a plot and does not return anything.

Author(s)

Lindsay V. Clark

References

Clark, L. V. and Drauch Schreier, A. (2017) Resolving microsatellite genotype ambiguity in populations of allopolyploid and diploidized autopolyploid organisms using negative correlations between allelic variables. *Molecular Ecology Resources*, **17**, 1090–1103. DOI: 10.1111/1755-0998.12639.

See Also

alleleCorrelations, recodeAllopoly

Examples

```
# get example dataset
data(AllopolyTutorialData)

# data cleanup
mydata <- deleteSamples(AllopolyTutorialData, c("301", "302", "303"))
PopInfo(mydata) <- rep(1:2, each = 150)
Genotype(mydata, 43, 2) <- Missing(mydata)

# allele assignments
# R is set to 10 here to speed processing for example.  It should typically be left at the default.
myassign <- processDatasetAllo(mydata, loci = c("Loc3", "Loc6"),
                               plotsfile = NULL, usePops = TRUE, R = 10,
                               parameters = data.frame(tolerance = c(0.5, 0.5),
                               swap = c(TRUE, FALSE),
                               null.weight = c(0.5, 0.5)))

# view best assignments for each locus
myassign$bestAssign

# plot K-means results
plotSSAllo(myassign$AlCorrArray)

# plot proportion of homoplasious alleles
plotParamHeatmap(myassign$propHomoplasious, "Pop1")
plotParamHeatmap(myassign$propHomoplasious, "Pop2")
plotParamHeatmap(myassign$propHomoplMerged, "Merged across populations")

# plot proportion of missing data, after recoding, for each locus and parameter set
plotParamHeatmap(myassign$missRate, main = "Missing data:")
```

---

read.ATetra *Read File in ATetra Format*

Description

Given a file formatted for the software ATetra, `read.ATetra` produces a `genambig` object containing genotypes, population identities, population names, and a dataset description from the file. Ploidy in the `genambig` object is automatically set to `4`.

Usage

```
read.ATetra(infile)
```

Arguments

`infile`	Character string. A file path to the file to be read.

Details

`read.ATetra` reads text files in the exact format specified by the ATetra documentation. Note that this format only allows tetraploid data and that there can be no missing data.

Value

A `genambig` object as described above.

Author(s)

Lindsay V. Clark

References

http://www.vub.ac.be/APNA/ATetra_Manual-1-1.pdf

van Puyvelde, K., van Geert, A. and Triest, L. (2010) ATETRA, a new software program to analyze tetraploid microsatellite data: comparison with TETRA and TETRASAT. *Molecular Ecology Resources* **10**, 331–334.

See Also

`write.ATetra`, `read.Tetrasat`, `read.GeneMapper`, `read.Structure`, `read.GenoDive`, `read.SPAGeDi`, `read.POPDIST`, `read.STRand`

Examples

```
# create a file to be read
# (this would normally be done in a text editor or with ATetra's Excel template)
myfile <- tempfile()

cat("TIT,Sample Rubus Data for ATetra", "LOC,1,CBA15",
"POP,1,1,Commonwealth", "IND,1,1,1,CMW1,197,208,211,213",
"IND,1,1,2,CMW2,197,207,211,212", "IND,1,1,3,CMW3,197,208,212,219",
"IND,1,1,4,CMW4,197,208,212,219", "IND,1,1,5,CMW5,197,208,211,212",
"POP,1,2,Fall Creek Lake", "IND,1,2,6,FCR4,197,207,211,212",
"IND,1,2,7,FCR7,197,208,212,218", "IND,1,2,8,FCR14,197,207,212,218",
```

```
"IND,1,2,9,FCR15,197,208,211,212", "IND,1,2,10,FCR16,197,208,211,212",
"IND,1,2,11,FCR17,197,207,212,218","LOC,2,CBA23","POP,2,1,Commonwealth",
"IND,2,1,1,CMW1,98,100,106,125","IND,2,1,2,CMW2,98,125,,",
"IND,2,1,3,CMW3,98,126,,","IND,2,1,4,CMW4,98,106,119,127",
"IND,2,1,5,CMW5,98,106,125,","POP,2,2,Fall Creek Lake",
"IND,2,2,6,FCR4,98,125,,","IND,2,2,7,FCR7,98,106,126,",
"IND,2,2,8,FCR14,98,127,,","IND,2,2,9,FCR15,98,108,117,",
"IND,2,2,10,FCR16,98,125,,","IND,2,2,11,FCR17,98,126,,","END",
file = myfile, sep = "\n")

# Read the file and examine the data
exampledata <- read.ATetra(myfile)
summary(exampledata)
PopNames(exampledata)
viewGenotypes(exampledata)
```

`read.GeneMapper` *Read GeneMapper Genotypes Tables*

Description

Given a vector of filepaths to tab-delimited text files containing genotype data in the ABI GeneMapper Genotypes Table format, `read.GeneMapper` produces a `genambig` object containing the genotype data.

Usage

```
read.GeneMapper(infiles, forceInteger=TRUE)
```

Arguments

`infiles`	A character vector of paths to the files to be read.
`forceInteger`	Boolean. If `TRUE`, alleles will be coerced to integers. This is particularly useful for stripping any white space from allels and preventing alleles from being imported as character strings. If `FALSE`, alleles will be imported as numeric or character values, depending on the content of the input file(s).

Details

`read.GeneMapper` can read the genotypes tables that are exported by the Applied Biosystems GeneMapper software. The only alterations to the files that the user may have to make are 1) delete any rows with missing data or fill in `-9` in the first allele slot for that row, 2) make sure that all allele names are numeric representations of fragment length (no question marks or dashes), and 3) put sample names into the Sample Name column, if the names that you wish to use in analysis are not already there. Each file should have the standard header row produced by the software. If any sample has more than one genotype listed for a given locus, only the last genotype listed will be used.

The file format is simple enough that the user can easily create files manually if GeneMapper is not the software used in allele calling. The files are tab-delimited text files. There should be a header row with column names. The column labeled "Sample Name" should contain the names of the samples, and the column labeled "Marker" should contain the names of the loci. You can have as many or as few columns as needed to contain the alleles, and each of these columns should be labeled "Allele X" where X is a number unique to each column. Row labels and any other columns are ignored. For any given sample, each allele is listed only once and is given as an integer that is the length of the fragment in nucleotides. Duplicate alleles in the same row are ignored by `read.GeneMapper`. Alleles are separated by tabs. If you have more allele columns than alleles for any given sample, leave the extra cells blank so that `read.table` will read them as `NA`. Example data files in this format are included in the package.

`read.GeneMapper` will read all of your data at once. It takes as its first argument a character vector containing paths to all of the files to be read. How the data are distributed over these files does not matter. The function finds all unique sample names and all unique markers across all the files, and automatically puts a missing data symbol into the list if a particular sample and locus combination is not found. Rows in which all allele cells are blank should NOT be included in the input files; either delete these rows or put the missing data symbol into the first allele cell.

Sample and locus names must be consistent within and across the files. The object that is produced is indexed by these names.

If `forceInteger=FALSE`, alleles can be non-numeric values. Some functionality of **polysat** will be lost in this case, but it could allow for the import of SNP data, for example.

Value

A `genambig` object containing genotypes from the files, stored as vectors of unique alleles in its `Genotypes` slot. Other slots are left at the default values.

Note

A 'subscript out of bounds' error may mean that a sample name or marker was left blank in one of the input files. A 'NAs introduced by coercion' warning when `forceInteger=TRUE` means that a non-numeric, non-whitespace character was included in one of the allele fields of the file(s), in which case the file(s) should be carefully checked and re-imported.

Author(s)

Lindsay V. Clark

References

GeneMapper website: https://www.thermofisher.com/order/catalog/product/4475073

See Also

`genambig`, `read.Structure`, `read.GenoDive`, `read.SPAGeDi`, `read.Tetrasat`, `read.ATetra`, `write.GeneMapper`, `read.POPDIST`, `read.STRand`

Examples

```
# create a table of data
gentable <- data.frame(Sample.Name=rep(c("ind1","ind2","ind3"),2),
                       Marker=rep(c("loc1","loc2"), each=3),
                       Allele.1=c(202,200,204,133,133,130),
                       Allele.2=c(206,202,208,136,142,136),
                       Allele.3=c(NA,208,212,145,148,NA),
                       Allele.4=c(NA,216,NA,151,157,NA)
                       )
# create a file (inspect this file in a text editor or spreadsheet
# software to see the required format)
myfile <- tempfile()
write.table(gentable, file=myfile, quote=FALSE, sep="\t",
            na="", row.names=FALSE, col.names=TRUE)

# read the file
mygenotypes <- read.GeneMapper(myfile)

# inspect the results
viewGenotypes(mygenotypes)
```

---

read.GenoDive *Import Genotype Data from GenoDive File*

Description

read.GenoDive takes a text file in the format for the software GenoDive and produces a genambig object.

Usage

```
read.GenoDive(infile)
```

Arguments

infile	A character string. The path to the file to be read.

Details

GenoDive is a Mac-only program for population genetic analysis that allows for polyploid data. read.GenoDive imports data from text files formatted for this program.

The first line of the file is a comment line, which is written to the Description slot of the genambig object. On the second line, separated by tabs, are the number of individuals, number of populations, number of loci, maximum ploidy (ignored), and number of digits used to code alleles.

The following lines contain the names of populations, which are written to the PopNames slot of the genambig object. After that is a header line for the genotype data. This line contains, separated by tabs, column headers for populations, clones (optional), and individuals, followed by the name of each locus. The locus names for the genotype object are derived from this line.

Each individual is on one line following the genotype header line. Separated by tabs are the population number, the clone number (optional), the individual name (used as the sample name in the output) and the genotypes at each locus. Alleles at one locus are concatenated together in one string without any characters to separate them. Each allele must have the same number of digits, although leading zeros can be omitted.

If the only alleles listed for a particular individual and locus are zeros, this is interpreted by `read.GenoDive` as missing data, and `Missing(object)` (the default, `-9`) is written in that genotype slot in the `genambig` object. GenoDive allows for a genotype to be partially missing but **polysat** does not; therefore, if an allele is coded as zero but other alleles are recorded for that sample and locus, the output genotype will just contain the alleles that are present, with the zeros thrown out.

Value

A `genambig` object containing the data from the file.

Author(s)

Lindsay V. Clark

References

Meirmans, P. G. and Van Tienderen, P. H. (2004) GENOTYPE and GENODIVE: two programs for the analysis of genetic diversity of asexual organisms. *Molecular Ecology Notes* **4**, 792-794.

http://www.bentleydrummer.nl/software/software/GenoDive.html

See Also

`read.GeneMapper`, `write.GenoDive`, `read.Tetrasat`, `read.ATetra`, `read.Structure`, `read.SPAGeDi`, `read.POPDIST`, `read.STRand`

Examples

```
# create data file (normally done in a text editor or spreadsheet software)
myfile <- tempfile()
cat(c("example comment line", "5\t2\t2\t3\t2", "pop1", "pop2",
      "pop\tind\tloc1\tloc2", "1\tJohn\t102\t1214",
      "1\tPaul\t202\t0", "2\tGeorge\t101\t121213",
      "2\tRingo\t10304\t131414","1\tYoko\t10303\t120014"),
    file = myfile, sep = "\n")

# import file data
exampledata <- read.GenoDive(myfile)

# view data
summary(exampledata)
viewGenotypes(exampledata)
exampledata
```

`read.POPDIST` *Read Genotype Data in POPDIST Format*

Description

`read.POPDIST` reads one or more text files formatted for the software POPDIST and produces a `"genambig"` object containing genotypes, ploidies, and population identities from the file(s).

Usage

```
read.POPDIST(infiles)
```

Arguments

`infiles`	A character vector of file paths to be read.

Details

The format for the software POPDIST is a modified version of the popular Genepop format. The first line is a comment line, followed by a list of locus names, each on a separate line or on one line separated by commas. A line starting with the string "Pop" ("pop" and "POP" are also recognized) indicates the beginning of data for one population. Each individual is then represented on one line, with the population name and individual genotype separated by a tab followed by a comma. Genotypes for different loci are separated by a tab or space. Each allele must be coded by two digits. Zeros ("00") indicate missing data, either for an entire locus or for a partially heterozygous genotype. Partially heterozygous genotypes can also be represented by the arbitrary duplication of alleles.

If more than one file is read at once, locus names must be consistent across all files. Locus and population names should not start with "Pop", "pop", or "POP", as `read.POPDIST` searches for these character strings in order to identify the lines that delimit populations.

Value

A `"genambig"` object. The `Description` slot of the object is taken from the comment line of the first file. Locus names are taken from the files, and samples are given numbers instead of names. Each genotype consists of all unique non-zero integers for a given sample and locus. The `Ploidies` slot is filled in based on how many alleles are present at each locus of each sample (the number of characters for the genotype, divided by two). `reformatPloidies` is used internally by the function to collapse the ploidies to the simplest format. Population names are taken from the individual genotype lines, and population identities are recorded based on how the individuals are delimited by "Pop" lines.

Author(s)

Lindsay V. Clark

References

Tomiuk, J., Guldbrandtsen, B. and Loeschcke, B. (2009) Genetic similarity of polyploids: a new version of the computer program POPDIST (version 1.2.0) considers intraspecific genetic differentiation. *Molecular Ecology Resources* **9**, 1364-1368.

Guldbrandtsen, B., Tomiuk, J. and Loeschcke, B. (2000) POPDIST version 1.1.1: A program to calculate population genetic distance and identity measures. *Journal of Heredity* **91**, 178-179.

See Also

write.POPDIST, read.Tetrasat, read.ATetra, read.Structure, read.SPAGeDi, read.GeneMapper, read.GenoDive, read.STRand

Examples

```
# Create a file to read (this is typically done in a text editor)
myfile <- tempfile()
cat("An example for the read.POPDIST documentation.",
"abcR",
"abcQ",
"Pop",
"Piscataqua\t, 0204 0505",
"Piscataqua\t, 0404 0307",
"Piscataqua\t, 050200 030509",
"Pop",
"Salmon Falls\t, 1006\t0805",
"Salmon Falls\t, 0510\t0308",
"Pop",
"Great Works\t, 050807 030800",
"Great Works\t, 0000 0408",
"Great Works\t, 0707 0305",
file=myfile, sep="\n")

# View the file in the R console (or open it in a text editor)
cat(readLines(myfile), sep="\n")

# Read the file into a "genambig" object
fishes <- read.POPDIST(myfile)

# View the data in the object
summary(fishes)
PopNames(fishes)
PopInfo(fishes)
Ploidies(fishes)
viewGenotypes(fishes)
```

---

read.SPAGeDi *Read Genotypes in SPAGeDi Format*

Description

`read.SPAGeDi` can read a text file formatted for the SPAGeDi software and return a `genambig` object, as well as optionally returning a data frame of spatial coordinates. The `genambig` object includes genotypes, ploidies, and population identities (from the category column, if present) from the file.

Usage

```
read.SPAGeDi(infile, allelesep = "/", returnspatcoord = FALSE)
```

Arguments

`infile`	A character string indicating the path of the file to read.
`allelesep`	The character that is used to delimit alleles within genotypes, or `""` if alleles have a fixed number of digits and are not delimited by any character. Other examples shown in section 3.2.1 of the SPAGeDi 1.3 manual include `"/"`, `" "`, `"_"`, `":"`, `"."`, and `"--"`.
`returnspatcoord`	Boolean. Indicates whether a data frame should be returned containing the spatial coordinates columns.

Details

SPAGeDi offers a lot of flexibility in how data files are formatted. `read.SPAGeDi` accomodates most of that flexibility. The primary exception is that alleles must be delimited in the same way across all genotypes, as specified by `allelesep`. Comment lines beginning with `//`, as well as blank lines, are ignored by `read.SPAGeDi` just as they are by SPAGeDi.

`read.SPAGeDi` is not designed to read dominant data (see section 3.2.2 of the SPAGeDi 1.3 manual). However, see `genbinary.to.genambig` for a way to read this type of data after some simple manipulation in a spreadsheet program.

The first line of a SPAGeDi file contains information that is used by `read.SPAGeDi`. The ploidy as specified in the 6th position of the first line is ignored, and is instead calculated by counting alleles for each individual (including zeros on the right, but not the left, side of the genotype). The number of digits specified in the 5th position of the first line is only used if `allelesep=""`. All other values in the first line are important for the function.

If the only alleles found for a particular individual and locus are zeros, the genotype is interpreted as missing. Otherwise, zeros on the left side of a genotype are ignored, and zeros on the right side of a genotype are used in calculating the ploidy but are not included in the genotype object that is returned. If `allelesep=""`, `read.SPAGeDi` checks that the number of characters in the genotype can be evenly divided by the number of digits per allele. If not, zeros are added to the left of the genotype string before splitting it into alleles.

The `Ploidies` slot of the `"genambig"` object that is created is initially indexed by both sample and locus, with ploidy being written to the slot on a per-genotype basis. After all genotypes have been imported, `reformatPloidies` is used to convert `Ploidies` to the simplest possible format before the object is returned.

Value

Under the default where `returnspatcoord=FALSE`, a `genambig` object is returned. Alleles are formatted as integers. The `Ploidies` slot is filled in according to the number of alleles per genotype, ignoring zeros on the left. If the first line of the file indicates that there are more than zero categories, the category column is used to fill in the `PopNames` and `PopInfo` slots.

Otherwise, a list is returned:

`SpatCoord`	A data frame of spatial coordinates, unchanged from the file. The format of each column is determined under the default `read.table` settings. Row names are individual names from the file. Column names are the same as in the file.
`Dataset`	A `genambig` object as described above.

Author(s)

Lindsay V. Clark

References

https://ebe.ulb.ac.be/ebe/SPAGeDi.html

Hardy, O. J. and Vekemans, X. (2002) SPAGeDi: a versatile computer program to analyse spatial genetic structure at the individual or population levels. *Molecular Ecology Notes* **2**, 618–620.

See Also

`write.SPAGeDi`, `genbinary.to.genambig`, `read.table`, `read.GeneMapper`, `read.GenoDive`, `read.Structure`, `read.ATetra`, `read.Tetrasat`, `read.POPDIST`, `read.STRand`

Examples

```
# create a file to read (usually done with spreadsheet software or a
# text editor):
myfile <- tempfile()
cat("// here's a comment line at the beginning of the file",
"5\t0\t-2\t2\t2\t4",
"4\t5\t10\t50\t100",
"Ind\tLat\tLong\tloc1\tloc2",
"ind1\t39.5\t-120.8\t00003133\t00004040",
"ind2\t39.5\t-120.8\t3537\t4246",
"ind3\t42.6\t-121.1\t5083332\t40414500",
"ind4\t38.2\t-120.3\t00000000\t41430000",
"ind5\t38.2\t-120.3\t00053137\t00414200",
"END",
sep="\n", file=myfile)

# display the file
cat(readLines(myfile), sep="\n")

# read the file
mydata <- read.SPAGeDi(myfile, allelesep = "",
returnspatcoord = TRUE)
```

```
# view the data
mydata
viewGenotypes(mydata[[2]])
```

`read.STRand` *Read Genotypes Produced by STRand Software*

Description

This function reads in data in a format derived from the "BTH" format for exporting genotypes from the allele calling software STRand.

Usage

```
read.STRand(file, sep = "\t", popInSam = TRUE)
```

Arguments

`file`	A text string indicating the file to read.
`sep`	Field delimiter for the file. Tab by default.
`popInSam`	Boolean. If `TRUE`, fields from the "Pop" and "Ind" columns will be concatenated to create a sample name. If `FALSE`, only the "Ind" column will be used for sample names.

Details

This function does not read the files directly produced from STRand, but requires some simple clean-up in spreadsheet software. The BTH format in STRand produces two columns per locus. One of these columns should be deleted so that there is just one column per locus. Loci names should remain in the column headers. The column containing sample names should be deleted or renamed "Ind". A "Pop" column will need to be added, containing population names. An "Ind" column is also necessary, containing either full sample names or a sample suffix to be concatenated with the population name (see `popInSam` argument).

STRand adds an asterisk to the end of any genotype with more than two alleles. `read.STRand` will automatically strip this asterisk out of the genotype.

Missing data is indicated by a zero in the file.

Value

A `"genambig"` object containing genotypes, locus and sample names, population names, and population identities from the file.

Author(s)

Lindsay V. Clark

References

https://vgl.ucdavis.edu/STRand

Toonen, R. J. and Hughes, S. (2001) Increased Throughput for Fragment Analysis on ABI Prism 377 Automated Sequencer Using a Membrane Comb and STRand Software. *Biotechniques* **31**, 1320–1324.

See Also

read.table, read.GeneMapper, read.GenoDive, read.Structure, read.ATetra, read.Tetrasat, read.POPDIST, read.SPAGeDi

Examples

```
# generate file to read
strtemp <- data.frame(Pop=c("P1","P1","P2","P2"),
                      Ind=c("a","b","a","b"),
                      LocD=c("0","172/174","170/172/178*","172/176"),
                      LocG=c("130/136/138/142*","132/136","138","132/140/144*"))
myfile <- tempfile()
write.table(strtemp, file=myfile, sep="\t",
            row.names=FALSE, quote=FALSE)

# read the file
mydata <- read.STRand(myfile)
viewGenotypes(mydata)
PopNames(mydata)

# alternative example with popInSam=FALSE
strtemp$Ind <- c("OH1","OH5","MT4","MT7")
write.table(strtemp, file=myfile, sep="\t",
            row.names=FALSE, quote=FALSE)
mydata <- read.STRand(myfile, popInSam=FALSE)
Samples(mydata)
PopNames(mydata)
```

---

read.Structure *Read Genotypes and Other Data from a Structure File*

Description

read.Structure creates a genambig object by reading a text file formatted for the software Structure. Ploidies and PopInfo (if available) are also written to the object, and data from additional columns can optionally be extracted as well.

Usage

```
read.Structure(infile, ploidy, missingin = -9, sep = "\t",
               markernames = TRUE, labels = TRUE, extrarows = 1,
               popinfocol = 1, extracols = 1, getexcols = FALSE,
               ploidyoutput="one")
```

Arguments

infile	Character string. The file path to be read.
ploidy	Integer. The ploidy of the file, *i.e.* how many rows there are for each individual.
missingin	The symbol used to represent missing data in the Structure file.
sep	The character used to delimit the fields of the Structure file (tab by default).
markernames	Boolean, indicating whether the file has a header containing marker names.
labels	Boolean, indicating whether the file has a column containing sample names.
extrarows	Integer. The number of extra rows that the file has, not counting marker names. This could include rows for recessive alleles, inter-marker distances, or phase information.
popinfocol	Integer. The column number (after the labels column, if present) where the data to be used for PopInfo are stored. Can be NA to indicate that PopInfo should not be extracted from the file.
extracols	Integer. The number of extra columns that the file has, not counting sample names (labels) but counting the column to be used for PopInfo. This could include PopData, PopFlag, LocData, Phenotype, or any other extra columns.
getexcols	Boolean, indicating whether the function should return the data from any extra columns.
ploidyoutput	This argument determines what is assigned to the Ploidies slot of the "genambig" dataset that is output. It should be a string, either "one", "samplemax", or "matrix". This indicates, respectively, that ploidy should be used as the ploidy of the entire dataset, that the maximum number of alleles for each sample should be used as the ploidy of that sample, or that ploidies should be stored as a matrix of allele counts for each sample*locus.

Details

The current version of read.Structure does not support the ONEROWPERIND option in the file format. Each locus must only have one column. If your data are in ONEROWPERIND format, it should be fairly simple to manipulate it in a spreadsheet program so that it can be read by read.GeneMapper instead.

read.Structure uses read.table to initially read the file into a data frame, then extracts information from the data frame. Because of this, any header rows (particularly the one containing marker names) should have leading tabs (or spaces if sep=" ") so that the marker names align correctly with their corresponding genotypes. You should be able to open the file in a spreadsheet program and have everything align correctly.

If the file does not contain sample names, set labels=FALSE. The samples will be numbered instead, and if you like you can use the Samples<- function to edit the sample names of the genotype object after import. Likewise, if markernames=FALSE, the loci will be numbered automatically by the column names that read.table creates, but these can also be edited after the fact.

The Ploidies slot of the "genambig" object that is created is initially indexed by both sample and locus, with ploidy being written to the slot on a per-genotype basis. After all genotypes have been imported, reformatPloidies is used to convert Ploidies to the simplest possible format before the object is returned.

Value

If `getexcols=FALSE`, the function returns only a `genambig` object.

If `getexcols=TRUE`, the function returns a list with two elements. The first, named `ExtraCol`, is a data frame, where the row names are the sample names and each column is one of the extra columns from the file (but with each sample only once instead of being repeated `ploidy` number of times). The second element is named `Dataset` and is the genotype object described above.

Author(s)

Lindsay V. Clark

References

https://web.stanford.edu/group/pritchardlab/structure_software/release_versions/v2.3.4/structure_doc.pdf

Hubisz, M. J., Falush, D., Stephens, M. and Pritchard, J. K. (2009) Inferring weak population structure with the assistance of sample group information. *Molecular Ecology Resources* **9**, 1322–1332.

Falush, D., Stephens, M. and Pritchard, J. K. (2007) Inferences of population structure using multilocus genotype data: dominant markers and null alleles. *Molecular Ecology Notes* **7**, 574–578.

See Also

`write.Structure`, `read.GeneMapper`, `read.Tetrasat`, `read.ATetra`, `read.GenoDive`, `read.SPAGeDi`, `read.POPDIST`, `read.STRand`

Examples

```
# create a file to read (normally done in a text editor or spreadsheet
# software)
myfile <- tempfile()
cat("\t\tRhCBA15\tRhCBA23\tRhCBA28\tRhCBA14\tRUB126\tRUB262\tRhCBA6\tRUB26",
    "\t\t-9\t-9\t-9\t-9\t-9\t-9\t-9\t-9",
    "WIN1B\t1\t197\t98\t152\t170\t136\t208\t151\t99",
    "WIN1B\t1\t208\t106\t174\t180\t166\t208\t164\t99",
    "WIN1B\t1\t211\t98\t182\t187\t184\t208\t174\t99",
    "WIN1B\t1\t212\t98\t193\t170\t203\t208\t151\t99",
    "WIN1B\t1\t-9\t-9\t-9\t-9\t-9\t-9\t-9\t-9",
    "WIN1B\t1\t-9\t-9\t-9\t-9\t-9\t-9\t-9\t-9",
    "WIN1B\t1\t-9\t-9\t-9\t-9\t-9\t-9\t-9\t-9",
    "WIN1B\t1\t-9\t-9\t-9\t-9\t-9\t-9\t-9\t-9",
    "MCD1\t2\t208\t100\t138\t160\t127\t202\t151\t124",
    "MCD1\t2\t208\t102\t153\t168\t138\t207\t151\t134",
    "MCD1\t2\t208\t106\t157\t180\t162\t211\t151\t137",
    "MCD1\t2\t208\t110\t159\t187\t127\t215\t151\t124",
    "MCD1\t2\t208\t114\t168\t160\t127\t224\t151\t124",
    "MCD1\t2\t208\t124\t193\t160\t127\t228\t151\t124",
    "MCD1\t2\t-9\t-9\t-9\t-9\t-9\t-9\t-9\t-9",
    "MCD1\t2\t-9\t-9\t-9\t-9\t-9\t-9\t-9\t-9",
    "MCD2\t2\t208\t98\t138\t160\t136\t202\t150\t120",
```

```
    "MCD2\t2\t208\t102\t144\t174\t145\t214\t150\t132",
    "MCD2\t2\t208\t105\t148\t178\t136\t217\t150\t135",
    "MCD2\t2\t208\t114\t151\t184\t136\t227\t150\t120",
    "MCD2\t2\t208\t98\t155\t160\t136\t202\t150\t120",
    "MCD2\t2\t208\t98\t157\t160\t136\t202\t150\t120",
    "MCD2\t2\t208\t98\t163\t160\t136\t202\t150\t120",
    "MCD2\t2\t208\t98\t138\t160\t136\t202\t150\t120",
    "MCD3\t2\t197\t100\t172\t170\t159\t213\t174\t134",
    "MCD3\t2\t197\t106\t174\t178\t193\t213\t176\t132",
    "MCD3\t2\t-9\t-9\t-9\t-9\t-9\t-9\t-9\t-9",
    "MCD3\t2\t-9\t-9\t-9\t-9\t-9\t-9\t-9\t-9",
    "MCD3\t2\t-9\t-9\t-9\t-9\t-9\t-9\t-9\t-9",
    "MCD3\t2\t-9\t-9\t-9\t-9\t-9\t-9\t-9\t-9",
    "MCD3\t2\t-9\t-9\t-9\t-9\t-9\t-9\t-9\t-9",
    "MCD3\t2\t-9\t-9\t-9\t-9\t-9\t-9\t-9\t-9",
    sep="\n",file=myfile)

# view the file
cat(readLines(myfile), sep="\n")

# read the structure file into genotypes and populations
testdata <- read.Structure(myfile, ploidy=8)

# examine the results
testdata
```

`read.Tetrasat` *Read Data from a TETRASAT Input File*

Description

Given a file containing genotypes in the TETRASAT format, `read.Tetrasat` produces a `genambig` object containing genotypes and population identities from the file.

Usage

```
read.Tetrasat(infile)
```

Arguments

`infile`	A character string of the file path to be read.

Details

`read.Tetrasat` reads text files that are in the exact format specified by the software TETRASAT and TETRA (see references for more information). This is similar to the file format for GenePop but allows for up to four alleles per locus. All alleles must be coded by two digits. Another difference between the TETRASAT and GenePop formats is that in TETRASAT the sample name and genotypes are not separated by a comma, because the columns of data have fixed widths.

Since TETRASAT files also contain information about which samples belong to which populations, this information is put into the PopInfo slot of the genambig object. Population names are not taken from the file. The Ploidies slot is filled with the number 4 (using the "ploidyone" class), because all individuals should be tetraploid. The first line of the file is put into the Description slot.

Locus names should not contain the letters "pop", uppercase or lowercase, adjacent to each other.

Value

A genambig object containing data from the file.

Author(s)

Lindsay V. Clark

References

Markwith, S. H., Stewart, D. J. and Dyer, J. L. (2006) TETRASAT: a program for the population analysis of allotetraploid microsatellite data. *Molecular Ecology Notes* **6**, 586-589.

Liao, W. J., Zhu, B. R., Zeng, Y. F. and Zhang, D. Y. (2008) TETRA: an improved program for population genetic analysis of allotetraploid microsatellite data. *Molecular Ecology Resources* **8**, 1260–1262.

See Also

read.GeneMapper, write.Tetrasat, read.ATetra, read.GenoDive, read.Structure, read.SPAGeDi, read.POPDIST, read.STRand

Examples

```
# example with defined data:
myfile <- tempfile()
cat("Sample Data", "A1_Gtype", "A10_Gtype", "B1_Gtype", "D7_Gtype",
"D9_Gtype", "D12_Gtype", "Pop",
"BCRHE 1             0406     04040404 0208     02020202 03030303 0710",
"BCRHE 10            0406     04040404 07070707 02020202 0304     0710",
"BCRHE 2             04040404 04040404 0708     02020202 010305   0710",
"BCRHE 3             04040404 04040404 02020202 0203     03030303 0809",
"BCRHE 4             04040404 04040404 0608     0203     03030303 070910",
"BCRHE 5             04040404 04040404 0208     02020202 03030303 050710",
"BCRHE 6             0304     04040404 0207     02020202 03030303 07070707",
"BCRHE 7             0406     04040404 0708     02020202 03030303 07070707",
"BCRHE 8             0304     04040404 0203     0203     03030303 0709",
"BCRHE 9             0406     04040404 0708     02020202 03030303 0710",
"Pop",
"BR 1                0406     04040404 05050505 02020202 03030303 1012",
"BR 10               030406   04040404 0607     02020202 03030303 1011",
"BR 2                030406   04040404 07070707 02020202 03030303 09090909",
"BR 3                010304   04040404 07070707 02020202 03030303 09090909",
"BR 4                030406   04040404 07070707 0203     03030303 10101010",
"BR 5                030406   04040404 07070707 02020202 03030303 10101010",
"BR 6                0406     04040404 0507     0203     03030303 10101010",
```

```
"BR 7               0304     04040404 0809     02020202 03030303 070910",
"BR 8               030406   04040404 07070707 02020202 03030303 070910",
"BR 9               0406     04040404 07070707 02020202 03030303 07070707",
sep="\n", file=myfile)
mydata2 <- read.Tetrasat(myfile)

summary(mydata2)
viewGenotypes(mydata2, loci="B1_Gtype")
```

---

recodeAllopoly *Create a New* genambig *Dataset with Loci Split into Isoloci*

Description

Given a "genambig" object and a list of allele assignments such as those produced by testAlGroups or catalanAlleles, recodeAllopoly will generate a new "genambig" object, with genotypes split according to which alleles belong to which isoloci.

Usage

```
recodeAllopoly(object, x, allowAneuploidy = TRUE,
               samples = Samples(object), loci = Loci(object))
```

Arguments

object	A "genambig" object containing the dataset that needs to be re-coded.
x	A list. Each item in the list should itself be a list, in the format output by testAlGroups, catalanAlleles, or mergeAlleleAssignments. Each sub-list has three items: $locus is the name of the locus, $SGploidy is an integer indicating the ploidy of each subgenome (*e.g.* 2 for an allotetraploid), and $assignments is a matrix of ones and zeros indicating which alleles belong to which isoloci.
allowAneuploidy	Boolean. This controls what happens when the function encounters genotypes that have more alleles than are possible for a given isolocus. (For example, the genotype has four alleles, but three belong to isolocus 1 and one belongs to isolocus 2.) If TRUE, the individual is assumed to be aneuploid at that locus, and its ploidy is adjusted only for that locus. If FALSE, missing data are recorded.
samples	An optional character vector indicating which samples to analyze and output.
loci	An optional character vector indicating which loci to analyze and output.

Details

The same locus may appear more than once in x, for example if distinct populations were analyzed separately to produce the allele assignments. If this is the case, recodeAllopoly will internally use mergeAlleleAssignments to consolidate items in x with the same locus name. Loci that are in x

but not `object` are ignored with a warning. Loci that are in `object` but not `x` are retained in the output of the function, but not re-coded.

This function allows homoplasy, and uses process-of-elimination to try to determine which isoloci the homoplasious alleles belong to. In cases where genotypes cannot be determined for certain due to homoplasy, missing data are inserted.

If a genotype has more alleles than should be possible (*e.g.* five alleles in an allotetraploid), the genotype is skipped and will be output as missing data for all corresponding isoloci.

Value

A `"genambig"` object, with loci that are in `x` split into the appropriate number of isoloci.

Author(s)

Lindsay V. Clark

References

Clark, L. V. and Drauch Schreier, A. (2017) Resolving microsatellite genotype ambiguity in populations of allopolyploid and diploidized autopolyploid organisms using negative correlations between alleles. *Molecular Ecology Resources*, **17**, 1090–1103. DOI: 10.1111/1755-0998.12639.

Examples

```
# generate a dataset for this example
testdata <- new("genambig", samples = paste("S", 1:9, sep = ""),
                loci = c("L1", "L2","L3"))
Genotypes(testdata, loci="L1") <-
    list(c(120,124),c(124,126,130),c(120,126),c(126,132,134),
         c(120,124,130,132),c(120,126,130),c(120,132,134),
         c(120,124,126,130),c(120,132,138))
Genotypes(testdata, loci="L2") <-
    list(c(210,219,222,225),c(216,228),c(210,213,219,222),c(213,222,225,228),
         c(210,213,216,219),c(222,228),c(213),c(210,216),c(219,222,228))
Genotypes(testdata, loci="L3") <-
    list(c(155,145,153),c(157,155),c(151,157,159,165),c(147,151),c(149,153,157),
         c(149,157),c(153,159,161),c(163,165),c(147,163,167))
viewGenotypes(testdata)

# generate allele assignments for this example
myAssign <- list(list(locus="L1", SGploidy=2,
             assignments=matrix(c(1,0,0,1,1,1,0,1,1,0,1,1), nrow=2,
                                ncol=6, dimnames=list(NULL,
                                  c("120","124","126","130","132","134")))),
    list(locus="L2", SGploidy=2,
         assignments=matrix(c(1,1,1,1,1,1,1,0,1,0,1,0,0,1), nrow=2, ncol=7,
    dimnames=list(NULL,c("210","213","216","219","222","225","228")))),
    list(locus="L3", SGploidy=2, assignments="No assignment"))
myAssign

# recode the dataset
```

```
splitdata <- recodeAllopoly(testdata, myAssign)

# view results
viewGenotypes(splitdata)
Ploidies(splitdata)
```

`reformatPloidies` *Convert Ploidy Format of a Dataset*

Description

This function changes the class of the object in the `Ploidies` slot of a `"gendata"` object. (See the four subclasses described in `"ploidysuper"`.) Existing ploidy data can either be erased or, if possible, used in the new format.

Usage

```
reformatPloidies(object, output = "collapse", na.rm = FALSE, erase = FALSE)
```

Arguments

`object`	A `"gendata"` object.
`output`	A character string indicating the desired result of the conversion: `"matrix"` if ploidies should be indexed by both sample and locus, `"sample"` if ploidies should be indexed only by sample, `"locus"` if ploidies should be indexed only by locus, `"one"` if there should be one ploidy for the entire dataset, or `"collapse"` if ploidies should be converted to the simplest possible format.
`na.rm`	Boolean. If `FALSE`, `NA` is treated as a unique ploidy. If `TRUE`, `NA` values are ignored assuming that each sample and/or locus has only one ploidy otherwise. This argument is passed directly to `plCollapse`.
`erase`	Boolean. If `TRUE`, the new `Ploidies` slot is simply filled with `NA` instead of existing ploidy values from `object`.

Details

This is a versatile function that can accomplish several tasks relating to the format of ploidies in the dataset:

If you wish to change how ploidy is indexed, but don't care about keeping any data in the `Ploidies` slot, set `erase=TRUE` and `output` to `"matrix"`, `"sample"`, `"locus"`, or `"one"`.

If you wish to keep ploidy data while moving from a simpler format to a more complex format (*i.e.* from `"one"` to any other format, or from `"sample"` or `"locus"` to `"matrix"`), leave `erase=FALSE` and set `output` to the desired format. Existing data will be duplicated to fill out the new format. For example, if ploidies were indexed by sample and you change to matrix format, the ploidy that had previously been recorded for each sample will be duplicated for each locus.

If you wish to keep ploidy data while performing any other format conversions (*e.g.* `"matrix"` to `"sample"` or `"sample"` to `"locus"`), the function will check that there is one unique ploidy for each sample, locus, or the entire dataset (as appropriate), and will produce an error if the conversion cannot be done without a loss of information.

If you wish to keep ploidy data and convert to the simplest possible format, set `output="collapse"`. The function will automatically determine the simplest format for conversion without loss of data. (The `read` functions in **polysat** that take ploidy data from the input file use this option.)

Value

A `"gendata"` object that is a copy of `object` but with the `Ploidies` slot converted to a new class.

Author(s)

Lindsay V. Clark

See Also

`plCollapse`, `"ploidysuper"`

Examples

```
# Make a new "genambig" object for this example
testdata <- new("genambig")
Ploidies(testdata)

## If you need to reformat before you have entered any ploidy
## information:

# convert from matrix to sample format
testdata <- reformatPloidies(testdata, output="sample")
Ploidies(testdata)

## If you have entered ploidy information but realized you can use a
## simpler format:

# Enter some ploidies
Ploidies(testdata)[1] <- 2
Ploidies(testdata)

# Convert from "sample" to "one" with na.rm=TRUE
testdata <- reformatPloidies(testdata, na.rm=TRUE, output="one")
Ploidies(testdata)

## If you change your mind and want to go back to a more complex format

# Convert from "one" to "locus"
testdata <- reformatPloidies(testdata, output="locus")
Ploidies(testdata)
```

simAllopoly *Generate Simulated Datasets*

Description

Given the number of subgenomes, the ploidy of each subgenome, and optionally, allele frequencies, simAllopoly will generate a "genambig" object containing simulated data for one locus.

Usage

```
simAllopoly(ploidy = c(2, 2), n.alleles = c(4, 4), n.homoplasy = 0,
            n.null.alleles=rep(0, length(ploidy)), alleles = NULL,
            freq = NULL, meiotic.error.rate=0, nSam = 100, locname = "L1")
```

Arguments

ploidy	A vector of integers, with one value for each subgenome, indicating the ploidy of that subgenome. For example, c(2,2) indicates an allotetraploid. An allohexaploid, with three diploid subgenomes, would be coded as c(2,2,2).
n.alleles	A vector, in similar format to the previous argument, indicating how many different unique alleles there are for each isolocus. Ignored if alleles is provided.
n.homoplasy	A single value indicating how many homoplasious alleles there are. Ignored if alleles is provided. This value should not be greater than any value in n.alleles. If freq is provided, the frequency or frequencies at the end of each vector will be the frequencies of homopolasious alleles.
n.null.alleles	A vector, in similar format to ploidy and n.alleles, indicating how many null alleles there are for each isolocus. Ignored if alleles is provided. These values should not be greater than n.alleles. If freq is provided, the frequency or frequencies at the beginning of each vector will be the null allele frequencies.
alleles	Optional. A list of vectors of allele names (which are usually expressed as integers, but can also be character strings if desired). Each element of the list contains the allele names for the corresponding isolocus. Zero indicates a null allele. Allele names that are identical between isoloci will be treated as homoplasious. If this argument is not provided, alleles will be named as described in "Details".
freq	Optional. A list of vectors of allele frequencies. If alleles is provided, all of the vectors must match in length between the two lists. Otherwise, the lengths of the vectors much match the values in n.alleles. If freq is not provided, it will be randomly generated.
meiotic.error.rate	A single value ranging from 0 to 0.5. The probability of a gamete containing a meiotic error involving this locus. See "Details".
nSam	A single value indicating the number of samples to generate.
locname	The name for the locus.

Details

If `alleles=NULL`, allele names will be generated in the format `A-1`, `A-2`, `B-1`, `B-2` etc., where A and B refer to separate subgenomes. Homoplasious alleles will be named `H-1`, `H-2`, etc.

Meiotic errors, as simulated by `simAllopoly`, always result in balanced aneuploidy, *i.e.* one copy of an isolocus will be replaced by an additional copy of a different isolocus. This is simulated on a per-gamete basis, so each gamete can have a maximum of one meiotic error per locus, but an individual could potentially be derived from two error-containing gametes. Note that in homozygotes and partial heterozygotes, it may not be possible to detect aneuploidy by examining the genotype; this phenomenon lowers the apparent rate of aneuploidy in the dataset.

If alleles are provided by the user with the `alleles` argument, zero (for sets of numeric alleles) or `"N"` (for sets of character alleles) indicates a null allele. The null allele will be simulated at the frequency specified, but will not be shown in the output dataset. Genotypes with no non-null alleles are recorded as missing.

Value

A `"genambig"` object.

Note

Unlike the code supplied in the file `extdata/simgen.R`, all genotypes in a dataset generated by this function will be of the same ploidy.

Author(s)

Lindsay V. Clark

References

Clark, L. V. and Drauch Schreier, A. (2017) Resolving microsatellite genotype ambiguity in populations of allopolyploid and diploidized autopolyploid organisms using negative correlations between alleles. *Molecular Ecology Resources*, **17**, 1090–1103. DOI: 10.1111/1755-0998.12639.

See Also

`alleleCorrelations`, `catalanAlleles`, `simgen`

Examples

```
# Generate an allotetraploid dataset with no homoplasy.
# One isolocus has five alleles, while the other has eight.
test <- simAllopoly(n.alleles=c(5,8))

# Generate an allo-octoploid dataset with two tetraploid subgenomes, ten
# alleles per subgenome, including one homoplasious allele.
test2 <- simAllopoly(ploidy=c(4,4), n.alleles=c(10,10), n.homoplasy=1)

# Generate an allotetraploid dataset, and manually define allele names
# and frequencies.
```

```
test3 <- simAllopoly(alleles=list(c(120,124,126),c(130,134,138,140)),
                         freq=list(c(0.4,0.3,0.3),c(0.25,0.25,0.25,0.25)))

# Generate an autotetraploid dataset with seven alleles
test4 <- simAllopoly(ploidy=4, n.alleles=7)

# Generate an allotetraploid dataset with a null allele at high frequency
test5 <- simAllopoly(n.null.alleles=c(1,0),
                     freq=list(c(0.5,0.1,0.1,0.3), c(0.25,0.25,0.4,0.1)))
```

`simgen` *Randomly Generated Data for Learning polysat*

Description

`genambig` object containing simulated data from three populations with 100 individuals each, at three loci. Individuals are a random mixture of diploids and tetraploids. Genotypes were generated according to pre-set allele frequencies.

Usage

```
data(simgen)
```

Format

A `genambig` object with data in the `Genotypes`, `PopInfo`, `PopNames`, `Ploidies` and `Usatnts` slots. This is saved as an .RData file. `simgen` was created using the code found in the "simgen.R" file in the "extdata" directory of the polysat package installation. This code may be useful for inspiration on how to create a simulated dataset.

Source

simulated data

See Also

`testgenotypes`, `genambig`

simpleFreq *Simple Allele Frequency Estimator*

Description

Given genetic data, allele frequencies by population are calculated. This estimation method assumes polysomic inheritance. For genotypes with allele copy number ambiguity, all alleles are assumed to have an equal chance of being present in multiple copies. This function is best used to generate initial values for more complex allele frequency estimation methods.

Usage

```
simpleFreq(object, samples = Samples(object), loci = Loci(object))
```

Arguments

object	A genbinary or genambig object containing genotype data. No NA values are allowed for PopInfo(object)[samples] or Ploidies(object,samples,loci). (Population identity and ploidy are needed for allele frequency calculation.)
samples	An optional character vector of samples to include in the calculation.
loci	An optional character vector of loci to include in the calculation.

Details

If object is of class genambig, it is converted to a genbinary object before allele frequency calculations take place. Everything else being equal, the function will work more quickly if it is supplied with a genbinary object.

For each sample*locus, a conversion factor is generated that is the ploidy of the sample (and/or locus) as specified in Ploidies(object) divided by the number of alleles that the sample has at that locus. Each allele is then considered to be present in as many copies as the conversion factor (note that this is not necessarily an integer). The number of copies of an allele is totaled for the population and is divided by the total number of genomes in the population (minus missing data at the locus) in order to calculate allele frequency.

A major assumption of this calculation method is that each allele in a partially heterozygous genotype has an equal chance of being present in more than one copy. This is almost never true, because common alleles in a population are more likely to be partially homozygous in an individual. The result is that the frequency of common alleles is underestimated and the frequency of rare alleles is overestimated. Also note that the level of inbreeding in the population has an effect on the relationship between genotype frequencies and allele frequencies, but is not taken into account in this calculation.

Value

Data frame, where each population is in one row. If each sample in object has only one ploidy, the first column of the data frame is called Genomes and contains the number of genomes in each population. Otherwise, there is a Genomes column for each locus. Each remaining column contains

frequencies for one allele. Columns are named by locus and allele, separated by a period. Row names are taken from `PopNames(object)`.

Author(s)

Lindsay V. Clark

See Also

genbinary, genambig

Examples

```
# create a data set for this example
mygen <- new("genambig", samples = paste("ind", 1:6, sep=""),
             loci = c("loc1", "loc2"))
mygen <- reformatPloidies(mygen, output="sample")
Genotypes(mygen, loci="loc1") <- list(c(206),c(208,210),c(204,206,210),
    c(196,198,202,208),c(196,200),c(198,200,202,204))
Genotypes(mygen, loci="loc2") <- list(c(130,134),c(138,140),c(130,136,140),
    c(138),c(136,140),c(130,132,136))
PopInfo(mygen) <- c(1,1,1,2,2,2)
Ploidies(mygen) <- c(2,2,4,4,2,4)

# calculate allele frequencies
myfreq <- simpleFreq(mygen)

# look at the results
myfreq

# an example where ploidy is indexed by locus instead
mygen2 <- new("genambig", samples = paste("ind", 1:6, sep=""),
             loci = c("loc1", "loc2"))
mygen2 <- reformatPloidies(mygen2, output="locus")
PopInfo(mygen2) <- 1
Ploidies(mygen2) <- c(2,4)
Genotypes(mygen2, loci="loc1") <- list(c(198), c(200,204), c(200),
                                       c(198,202), c(200), c(202,204))
Genotypes(mygen2, loci="loc2") <- list(c(140,144,146), c(138,144),
                                       c(136,138,144,148), c(140),
                                       c(140,142,146,150),
                                       c(142,148,150))
myfreq2 <- simpleFreq(mygen2)
myfreq2
```

`testgenotypes` *Rubus Genotype Data for Learning polysat*

Description

`genambig` object containing alleles of 20 *Rubus* samples at three microsatellite loci.

Usage

```
data(testgenotypes)
```

Format

A genambig object with data in the Genotypes, PopInfo, PopNames, and Usatnts slots. This is saved as a .RData file. Population identities are used here to indicate two different species.

Source

Clark, L. V. and Jasieniuk, M. (2012) Spontaneous hybrids between native and exotic *Rubus* in the Western United States produce offspring both by apomixis and by sexual recombination. *Heredity* **109**, 320–328. Data available at: doi:10.5061/dryad.m466f

See Also

FCRinfo, simgen, genambig

---

viewGenotypes *Print Genotypes to the Console*

Description

viewGenotypes prints a tab-delimited table of samples, loci, and alleles to the console so that genotypes can be easily viewed.

Usage

```
viewGenotypes(object, samples = Samples(object), loci = Loci(object))
```

Arguments

object	An object of one of the gendata subclasses, containing genotypes to be viewed.
samples	A numerical or character vector indicating which samples to display.
loci	A numerical or character vector indicating which loci to display.

Details

viewGenotypes is a generic function with methods for genambig and genbinary objects.

For a genambig object, a header line indicating sample, locus, and allele columns is printed. Genotypes are printed below this. Genotypes are ordered first by locus and second by sample.

For a genbinary object, the presence/absence matrix is printed, organized by locus. After the matrix for one locus is printed, a blank line is inserted and the matrix for the next locus is printed.

Value

No value is returned.

Author(s)

Lindsay V. Clark

See Also

Genotypes

Examples

```
# create a dataset for this example
mygen <- new("genambig", samples=c("ind1", "ind2", "ind3", "ind4"),
             loci=c("locA", "locB"))
Genotypes(mygen) <- array(list(c(98, 104, 108), c(100, 104, 110, 114),
                               c(102, 108, 110), Missing(mygen),
                               c(132, 135), c(138, 141, 147),
                               c(135, 141, 144), c(129, 150)),
                          dim=c(4,2))

# view the genotypes
viewGenotypes(mygen)
```

---

write.ATetra *Write Genotypes in ATetra Format*

Description

`write.ATetra` uses genotype and population information contained in a `genambig` object to create a text file of genotypes in the ATetra format.

Usage

```
write.ATetra(object, samples = Samples(object),
             loci = Loci(object), file = "")
```

Arguments

`object`	A `genambig` object containing the dataset of interest. Genotypes, population identities, population names, and the dataset description are used for creating the file. Ploidies must be set to 4.
`samples`	A character vector of samples to write to the file. This is a subset of `Samples(object)`.
`loci`	A character vector of loci to write to the file. This is a subset of `Loci(object)`.
`file`	A character string indicating the path and name to which to write the file.

Details

Note that missing data are not allowed in ATetra, although `write.ATetra` will still process missing data. When it does so, it leaves all alleles blank in the file for that particular sample and locus, and also prints a warning indicating which sample and locus had missing data.

Value

A file is written but no value is returned.

Author(s)

Lindsay V. Clark

References

http://www.vub.ac.be/APNA/ATetra_Manual-1-1.pdf

van Puyvelde, K., van Geert, A. and Triest, L. (2010) ATETRA, a new software program to analyze tetraploid microsatellite data: comparison with TETRA and TETRASAT. *Molecular Ecology Resources* **10**, 331-334.

See Also

read.ATetra, write.Tetrasat, write.GeneMapper, write.POPDIST

Examples

```
# set up sample data (usually done by reading files)
mysamples <- c("ind1", "ind2", "ind3", "ind4")
myloci <- c("loc1", "loc2")
mygendata <- new("genambig", samples=mysamples, loci=myloci)
mygendata <- reformatPloidies(mygendata, output="one")
Genotypes(mygendata, loci="loc1") <- list(c(202,204), c(204),
                                          c(200,206,208,212),
                                          c(198,204,208))
Genotypes(mygendata, loci="loc2") <- list(c(78,81,84),
                                          c(75,90,93,96,99),
                                          c(87), c(-9))
PopInfo(mygendata) <- c(1,2,1,2)
PopNames(mygendata) <- c("this pop", "that pop")
Ploidies(mygendata) <- 4
Description(mygendata) <- "Example for write.ATetra."

## Not run:
# write an ATetra file
write.ATetra(mygendata, file="atetratest.txt")

# view the file
cat(readLines("atetratest.txt"),sep="\n")

## End(Not run)
```

`write.freq.SPAGeDi` *Create a File of Allele Frequencies for SPAGeDi*

Description

A table of allele frequencies such as that produced by `simpleFreq` or `deSilvaFreq` is used to calculate average allele frequencies for the entire dataset. These are then written in a format that can be read by the software SPAGeDi.

Usage

```
write.freq.SPAGeDi(freqs, usatnts, file = "", digits = 2,
                   pops = row.names(freqs),
  loci = unique(as.matrix(as.data.frame(strsplit(names(freqs), split =
  ".", fixed = TRUE), stringsAsFactors = FALSE))[1, ]))
```

Arguments

`freqs`	A data frame of population sizes and allele frequencies, such as that produced by `simpleFreq` or `deSilvaFreq`. Populations are in rows, and alleles are in columns. A column is needed containing population sizes in number of genomes; this may either be a single column called "Genomes" or multiple columns named by the locus and "Genomes", sepearated by a period. All other columns contain allele frequencies. The column names for these should be the locus and allele separated by a period.
`usatnts`	An integer vector containing the lengths of the microsatellite repeats for the loci in the table. In most cases, if `object` is the `"gendata"` object used to generate `freqs`, then you should set `usatnts = Usatnts(object)`. This is needed to convert allele names in the same way that `write.SPAGeDi` converts allele names.
`file`	The name of the file to write.
`digits`	The number of digits to use to represent each allele. This should be the same as that used in `write.SPAGeDi`, so that allele names are consistent between the two files.
`pops`	An optional character vector indicating a subset of populations from the table to use in calculating mean allele frequencies.
`loci`	An optional character vector indicating a subset of loci to write to the file.

Details

For some calculations of inter-individual relatedness and kinship coefficients, SPAGeDi can read a file of allele frequencies to use in the calculation. `write.freq.SPAGeDi` puts allele frequencies from **polysat** into this format.

A weighted average of allele frequencies is calculated across all populations (or those specified by `pops`). The average is weighted by population size as specified in the "Genomes" column of `freqs`.

Allele names are converted to match those produced by `write.SPAGeDi`. Alleles are divided by the numbers in `usatnts` in order to convert fragment length in nucleotides to repeat numbers. If necessary, `10^(digits-1)` is repeatedly subtracted from all alleles until they can be represented using the right number of digits.

The file produced is tab-delimited and contains two columns per locus. The first column contains the locus name followed by all allele names, and the second column contains the number of alleles followed by the allele frequencies.

Value

A file is written but no value is returned.

Note

SPAGeDi can already estimate allele frequencies in a way that is identical to that of `simpleFreq`. Therefore, if you have allele frequencies produced by `simpleFreq`, there is not much sense in exporting them to SPAGeDi. `deSilvaFreq`, however, is a more advanced and accurate allele frequency estimation than what is available in SPAGeDi v1.3. `write.freq.SPAGeDi` exists primarily to export allele frequencies from `deSilvaFreq`.

Author(s)

Lindsay V. Clark

References

https://ebe.ulb.ac.be/ebe/SPAGeDi.html

Hardy, O. J. and Vekemans, X. (2002) SPAGeDi: a versatile computer program to analyse spatial genetic structure at the individual or population levels. *Molecular Ecology Notes* **2**, 618-620.

See Also

`write.SPAGeDi`, `deSilvaFreq`

Examples

```
## Not run:
# set up a genambig object to use in this example
mygen <- new("genambig", samples=c(paste("G", 1:30, sep=""),
                                   paste("R", 1:50, sep="")),
             loci=c("afrY", "ggP"))
PopNames(mygen) <- c("G", "R")
PopInfo(mygen) <- c(rep(1, 30), rep(2, 50))
mygen <- reformatPloidies(mygen, output="one")
Ploidies(mygen) <- 4
Usatnts(mygen) <- c(2, 2)

# randomly create genotypes according to pre-set allele frequencies
for(s in Samples(mygen, populations=1)){
    Genotype(mygen, s, "afrY") <-
```

```
        unique(sample(c(140, 142, 146, 150, 152), 4, TRUE,
                      c(.30, .12, .26, .08, .24)))
    Genotype(mygen, s, "ggP") <-
        unique(sample(c(210, 214, 218, 220, 222), 4, TRUE,
                      c(.21, .13, .27, .07, .32)))
}
for(s in Samples(mygen, populations=2)){
    Genotype(mygen, s, "afrY") <-
        unique(sample(c(140, 142, 144, 150, 152), 4, TRUE,
                      c(.05, .26, .17, .33, .19)))
    Genotype(mygen, s, "ggP") <-
        unique(sample(c(212, 214, 220, 222, 224), 4, TRUE,
                      c(.14, .04, .36, .20, .26)))
}

# write a SPAGeDi file
write.SPAGeDi(mygen, file="SPAGdataFreqExample.txt")

# calculate allele frequenies
myfreq <- deSilvaFreq(mygen, self = 0.05)

# write allele frequencies file
write.freq.SPAGeDi(myfreq, usatnts=Usatnts(mygen),
file="SPAGfreqExample.txt")

## End(Not run)
```

`write.GeneMapper` *Write Genotypes to a Table Similarly to ABI GeneMapper*

Description

Given a `genambig` object, `write.GeneMapper` writes a text file of a table containing columns for sample name, locus, and alleles.

Usage

```
write.GeneMapper(object, file = "", samples = Samples(object),
                 loci = Loci(object))
```

Arguments

`object`	A `genambig` object containing genotype data to write to the file. The `Ploidies` slot is used for determining how many allele columns to make.
`file`	Character string. The path to which to write the file.
`samples`	Character vector. Samples to write to the file. This should be a subset of `Samples(object)`.
`loci`	Character vector. Loci to write to the file. This should be a subset of `Loci(object)`.

Details

Although I do not know of any population genetic software other than **polysat** that will read this data format directly, the ABI GeneMapper Genotypes Table format is a convenient way for the user to store microsatellite genotype data and view it in a text editor or spreadsheet software. Each row contains the sample name, locus name, and alleles separated by tabs.

The number of allele columns needed is detected by the maximum value of `Ploidies(object,samples,loci)`. The function will add additional columns if it encounters genotypes with more than this number of alleles.

`write.GeneMapper` handles missing data in a very simple way, in that it writes the missing data symbol directly to the table as though it were an allele. If you want missing data to be represented differently in the table, you can open it in spreadsheet software and use find/replace or conditional formatting to locate missing data.

The file that is produced can be read back into R directly by `read.GeneMapper`, and therefore may be a convenient way to backup genotype data for future analysis and manipulation in **polysat**. (`save` can also be used to backup an R object more directly, including population and other information.) This can also enable the user to edit genotype data in spreadsheet software, if the `editGenotypes` function is not sufficient.

Value

A file is written but no value is returned.

Author(s)

Lindsay V. Clark

References

GeneMapper website: https://www.thermofisher.com/order/catalog/product/4475073

See Also

`read.GeneMapper`, `write.Structure`, `write.GenoDive`, `write.Tetrasat`, `write.ATetra`, `write.POPDIST`, `write.SPAGeDi`, `editGenotypes`

Examples

```
# create a genotype object (usually done by reading a file)
mysamples <- c("ind1", "ind2", "ind3", "ind4")
myloci <- c("loc1", "loc2")
mygendata <- new("genambig", samples=mysamples, loci=myloci)
mygendata <- reformatPloidies(mygendata, output="one")
Genotypes(mygendata, loci="loc1") <- list(c(202,204), c(204),
                                          c(200,206,208,212),
                                          c(198,204,208))
Genotypes(mygendata, loci="loc2") <- list(c(78,81,84),
                                          c(75,90,93,96,99),
                                          c(87), c(-9))
Ploidies(mygendata) <- 6
```

```
## Not run:
# write a GeneMapper file
write.GeneMapper(mygendata, "exampleGMoutput.txt")

# view the file with read.table
read.table("exampleGMoutput.txt", sep="\t", header=TRUE)

## End(Not run)
```

`write.GenoDive` *Write a File in GenoDive Format*

Description

`write.GenoDive` uses data from a `genambig` object to create a file formatted for the software GenoDive.

Usage

```
write.GenoDive(object, digits = 2, file = "",
               samples = Samples(object), loci = Loci(object))
```

Arguments

`object`	A `genambig` object containing genotypes, ploidies, population identities, microsatellite repeat lengths, and description for the dataset of interest.
`digits`	An integer indicating how many digits to use to represent each allele (usually 2 or 3).
`file`	A character string of the file path to which to write.
`samples`	A character vector of samples to include in the file. A subset of `Samples(object)`.
`loci`	A character vector of loci to include in the file. A subset of `Loci(object)`.

Details

The number of individuals, number of populations, number of loci, and maximum ploidy of the sample are calculated automatically and entered in the second line of the file. If the maximum ploidy needs to be reduced by random removal of alleles, it is possible to do this in the software GenoDive after importing the data. The `genambig` object should not have individuals with more alleles than the highest ploidy level listed in its `Ploidies` slot.

Several steps happen in order to convert alleles to the right format. First, all instances of the missing data symbol are replaced with zero. Alleles are then divided by the numbers provided in `Usatnts(object)` (and rounded down if necessary) in order to convert them from nucleotides to repeat numbers. If the alleles are still too big to be represented by the number of digits specified, `write.GenoDive` repeatedly subtracts a number (`10^(digits-1)`; 10 if `digits=2`) from all alleles at a locus until the alleles are small enough. Alleles are then converted to characters, and a leading zero is added to an allele if it does not have enough digits. These alleles are concatenated at each locus so that each sample*locus genotype is an uninterrupted string of numbers.

Value

A file is written but no value is returned.

Author(s)

Lindsay V. Clark

References

Meirmans, P. G. and Van Tienderen P. H. (2004) GENOTYPE and GENODIVE: two programs for the analysis of genetic diversity of asexual organisms. *Molecular Ecology Notes* **4**, 792-794.

http://www.bentleydrummer.nl/software/software/GenoDive.html

See Also

read.GenoDive, write.Structure, write.ATetra, write.Tetrasat, write.GeneMapper, write.POPDIST, write.SPAGeDi

Examples

```
# set up the genotype object (usually done by reading a file)
mysamples <- c("Mal", "Inara", "Kaylee", "Simon", "River", "Zoe",
               "Wash", "Jayne", "Book")
myloci <- c("loc1", "loc2")
mygendata <- new("genambig", samples=mysamples, loci=myloci)
mygendata <- reformatPloidies(mygendata, output="sample")
Genotypes(mygendata, loci="loc1") <- list(c(304,306), c(302,310),
                                          c(306), c(312,314),
    c(312,314), c(308,310), c(312), c(302,308,310), c(-9))
Genotypes(mygendata, loci="loc2") <- list(c(118,133), c(121,130),
                                          c(122,139), c(124,133),
    c(118,124), c(121,127), c(124,136), c(124,127,136), c(121,130))
Usatnts(mygendata) <- c(2,3)
PopNames(mygendata) <- c("Core","Outer Rim")
PopInfo(mygendata) <- c(2,1,2,1,1,2,2,2,1)
Ploidies(mygendata) <- c(2,2,2,2,2,2,2,3,2)
Description(mygendata) <- "Serenity crew"

## Not run:
# write files (use file="" to write to the console instead)
write.GenoDive(mygendata, digits=2, file="testGenoDive2.txt")
write.GenoDive(mygendata, digits=3, file="testGenoDive3.txt")

## End(Not run)
```

`write.POPDIST` *Write Genotypes to a POPDIST File*

Description

`write.POPDIST` uses data from a `"genambig"` object to write a file formatted for the software POPDIST.

Usage

```
write.POPDIST(object, samples = Samples(object),
              loci = Loci(object), file = "")
```

Arguments

`object`	A `"genambig"` object. `Ploidies` and `PopInfo` are required, and if provided `Usatnts` may be used to convert alleles to repeat number in order to represent each allele with two digits. Locus names and `PopNames` are used in the file, but sample names are not.
`samples`	An optional character vector of samples to use. Must be a subset of `Samples(object)`.
`loci`	An optional character vector of loci to use. Must be a subset of `Loci(object)`.
`file`	Character string. File path to which to write.

Details

POPDIST is a program that calculates inter-population distance measures, some of which are available for polyploid samples with allele copy number ambiguity. Each population must be of uniform ploidy, but different populations may have different ploidies.

Two types of warning messages may be printed by `write.POPDIST`. The first indicates that a population contains individuals of more than one ploidy. In this case a file is still written, but POPDIST may not be able to read it. Separate populations with different ploidies are okay. The second type of warning indicates that an individual has more alleles than its ploidy level. If this occurs, alleles are randomly removed from the genotype that is written to the file.

If necessary, `write.POPDIST` converts alleles into a two-digit format, similarly to `write.Tetrasat`. If the value of any allele for a given locus is greater than 99, the function first checks to see if the locus has a `Usatnts` value greater than 1, and if so divides all alleles by this value and rounds down. If the locus still has alleles with more than two digits, a multiple of 10 is subtracted from all alleles. A zero is placed in front of any allele with one digit.

Value

A file is written but no value is returned.

Author(s)

Lindsay V. Clark

References

Tomiuk, J., Guldbrandtsen, B. and Loeschcke, B. (2009) Genetic similarity of polyploids: a new version of the computer program POPDIST (version 1.2.0) considers intraspecific genetic differentiation. *Molecular Ecology Resources* **9**, 1364-1368.

Guldbrandtsen, B., Tomiuk, J. and Loeschcke, B. (2000) POPDIST version 1.1.1: A program to calculate population genetic distance and identity measures. *Journal of Heredity* **91**, 178–179.

See Also

read.POPDIST, write.Tetrasat, write.ATetra, write.SPAGeDi, write.GenoDive, write.Structure, write.GeneMapper

Examples

```
# create a "genambig" object containing the dataset
mygen <- new("genambig", samples=c("a", "b", "c", "d"),
             loci=c("loc1", "loc27"))
mygen <- reformatPloidies(mygen, output="sample")
Description(mygen) <- "Some example data for POPDIST"
PopInfo(mygen) <- c(1,1,2,2)
PopNames(mygen) <- c("Old Orchard Beach", "York Beach")
Ploidies(mygen) <- c(2,2,4,4)
Usatnts(mygen) <- c(2,2)
Genotypes(mygen, loci="loc1") <- list(c(128, 134), c(130),
                                      Missing(mygen), c(126, 128, 132))
Genotypes(mygen, loci="loc27") <- list(c(209,211), c(207,217),
                                       c(207,209,215,221), c(211,223))

## Not run:
# write the file
write.POPDIST(mygen, file="forPOPDIST.txt")

# view the file
cat(readLines("forPOPDIST.txt"), sep="\n")

## End(Not run)
```

---

write.SPAGeDi *Write Genotypes in SPAGeDi Format*

Description

write.SPAGeDi uses data contained in a genambig object to create a file that can be read by the software SPAGeDi. The user controls how the genotypes are formatted, and can provide a data frame of spatial coordinates for each sample.

Usage

```
write.SPAGeDi(object, samples = Samples(object),
              loci = Loci(object), allelesep = "/",
              digits = 2, file = "",
              spatcoord = data.frame(X = rep(1, length(samples)),
                                     Y = rep(1, length(samples)),
                                     row.names = samples))
```

Arguments

object	A genambig object containing genotypes, ploidies, population identities, and microsatellite repeat lengths for the dataset of interest.
samples	Character vector. Samples to write to the file. Must be a subset of Samples(object).
loci	Character vector. Loci to write to the file. Must be a subset of Loci(object).
allelesep	The character that will be used to separate alleles within a genotype. If each allele should instead be a fixed number of digits, with no characters to delimit alleles, set allelesep = "".
digits	Integer. The number of digits used to represent each allele.
file	A character string indicating the path to which the file should be written.
spatcoord	Data frame. Spatial coordinates of each sample. Column names are used for column names in the file. Row names indicate sample, or if absent it is assumed that the rows are in the same order as samples.

Details

The Categories column of the SPAGeDi file that is produced contains information from the PopNames and PopInfo slots of object; the population name for each sample is written to the column.

The first line of the file contains the number of individuals, number of categories, number of spatial coordinates, number of loci, number of digits for coding alleles, and maximum ploidy, and is generated automatically from the data provided.

The function does not write distance intervals to the file, but instead writes 0 to the second line.

All alleles for a given locus are divided by the Usatnts value for that locus, after all missing data symbols have been replaced with zeros. If necessary, a multiple of 10 is subtracted from all alleles at a locus in order to get the alleles down to the right number of digits.

If a genotype has fewer alleles than the Ploidies value for that sample and locus, zeros are added up to the ploidy. If the genotype has more alleles than the ploidy, a random subset of alleles is used and a warning is printed. If the genotype has only one allele (is fully heterozygous), then that allele is replicated to the ploidy of the individual. Genotypes are then concatenated into strings, delimited by allelesep. If allelesep="", leading zeros are first added to alleles as necessary to make them the right number of digits.

Value

A file is written but no value is returned.

Author(s)

Lindsay V. Clark

References

https://ebe.ulb.ac.be/ebe/SPAGeDi.html

Hardy, O. J. and Vekemans, X. (2002) SPAGeDi: a versatile computer program to analyse spatial genetic structure at the individual or population levels. *Molecular Ecology Notes* **2**, 618–620.

See Also

read.SPAGeDi, write.freq.SPAGeDi, write.GenoDive, write.Structure, write.GeneMapper, write.ATetra, write.Tetrasat, write.POPDIST

Examples

```
# set up data to write (usually read from a file)
mygendata <- new("genambig", samples = c("ind1","ind2","ind3","ind4"),
                 loci = c("loc1", "loc2"))
mygendata <- reformatPloidies(mygendata, output="sample")
Genotypes(mygendata, samples="ind1") <- list(c(102,106,108),c(207,210))
Genotypes(mygendata, samples="ind2") <- list(c(104),c(204,210))
Genotypes(mygendata, samples="ind3") <- list(c(100,102,108),c(201,213))
Genotypes(mygendata, samples="ind4") <- list(c(102,112),c(-9))
Ploidies(mygendata) <- c(3,2,2,2)
Usatnts(mygendata) <- c(2,3)
PopNames(mygendata) <- c("A", "B")
PopInfo(mygendata) <- c(1,1,2,2)
myspatcoord <- data.frame(X=c(27,29,24,30), Y=c(44,41,45,46),
                          row.names=c("ind1","ind2","ind3","ind4"))

## Not run:
# write a file
write.SPAGeDi(mygendata, spatcoord = myspatcoord,
              file="SpagOutExample.txt")

## End(Not run)
```

---

write.Structure *Write Genotypes in Structure 2.3 Format*

Description

Given a dataset stored in a genambig object, write.Structure produces a text file of the genotypes in a format readable by Structure 2.2 and higher. The user specifies the overall ploidy of the file, while the ploidy of each sample is extracted from the genambig object. PopInfo and other data can optionally be written to the file as well.

Usage

```
write.Structure(object, ploidy, file="",
                samples = Samples(object), loci = Loci(object),
                writepopinfo = TRUE, extracols = NULL,
                missingout = -9)
```

Arguments

object	A genambig object containing the data to write to the file. There must be non-NA values of Ploidies (and PopInfo if writepopinfo == TRUE) for samples.
ploidy	PLOIDY for Structure, *i.e.* how many rows per individual to write.
file	A character string specifying where the file should be written.
samples	An optional character vector listing the names of samples to be written to the file.
loci	An optional character vector listing the names of the loci to be written to the file.
writepopinfo	TRUE or FALSE, indicating whether to write values from the PopInfo slot of object to the file.
extracols	An array, with the first dimension names corresponding to samples, of PopData, PopFlag, LocData, Phenotype, or other values to be included in the extra columns in the file.
missingout	The number used to indicate missing data.

Details

Structure 2.2 and higher can process autopolyploid microsatellite data, although 2.3.3 or higher is recommended for its improvements on polyploid handling. The input format of Structure requires that each locus take up one column and that each individual take up as many rows as the parameter PLOIDY. Because of the multiple rows per sample, each sample name must be duplicated, as well as any population, location, or phenotype data. Partially heterozygous genotypes also must have one arbitrary allele duplicated up to the ploidy of the sample, and samples that have a lower ploidy than that used in the file (for mixed polyploid data sets) must have a missing data symbol inserted to fill in the extra rows. Additionally, if some samples have more alleles than PLOIDY (if you are using a lower PLOIDY to save processing time, or if there are extra alleles from scoring errors), some alleles must be randomly removed from the data. write.Structure performs this duplication, insertion, and random deletion of data.

The sample names from samples will be used as row names in the Structure file. Each sample name should only be in the vector samples once, because write.Structure will duplicate the sample names a number of times as dictated by ploidy.

In writing genotypes to the file, write.Structure compares the number of alleles in the genotype, the ploidy of the sample*locus as stored in Ploidies, and the ploidy of the file as stored in ploidy, and does one of six things (for a given sample x and locus loc):

1) If Ploidies(object,x,loc) is greater than or equal to ploidy, and length(Genotype(object, x, loc)) is equal to ploidy, the genotype data are used as is.

2) If Ploidies(object,x,loc) is greater than or equal to ploidy, and length(Genotype(object, x, loc)) is less than ploidy, the first allele is duplicated as many times as necessary for there to be as many alleles as ploidy.

3) If `Ploidies(object,x,loc)` is greater than or equal to `ploidy`, and `length(Genotype(object, x, loc))` is greater than `ploidy`, a random sample of the alleles, without replacement, is used as the genotype.

4) If `Ploidies(object,x,loc)` is less than `ploidy`, and `length(Genotype(object, x, loc))` is equal to `Ploidies(object,x,loc)`, the genotype data are used as is and missing data symbols are inserted in the extra rows.

5) If `Ploidies(object,x,loc)` is less than `ploidy`, and `length(Genotype(object, x, loc))` is less than `Ploidies(object,x,loc)`, the first allele is duplicated as many times as necessary for there to be as many alleles as `Ploidies(object,x,loc)`, and missing data symbols are inserted in the extra rows.

6) If `Ploidies(object,x,loc)` is less than `ploidy`, and `length(Genotype(object, x, loc))` is greater than `Ploidies(object,x,loc)`, a random sample, without replacement, of `Ploidies(object)[x]` alleles is used, and missing data symbols are inserted in the extra rows. (Alleles are removed even though there is room for them in the file.)

Two of the header rows that are optional for Structure are written by `write.Structure`. These are 'Marker Names', containing the names of loci supplied in `gendata`, and 'Recessive Alleles', which contains the missing data symbol once for each locus. This indicates to the program that all alleles are codominant with copy number ambiguity.

Value

No value is returned, but instead a file is written at the path specified.

Note

If `extracols` is a character array, make sure none of the elements contain white space.

Author(s)

Lindsay V. Clark

References

https://web.stanford.edu/group/pritchardlab/structure_software/release_versions/v2.3.4/structure_doc.pdf

Hubisz, M. J., Falush, D., Stephens, M. and Pritchard, J. K. (2009) Inferring weak population structure with the assistance of sample group information. *Molecular Ecology Resources* **9**, 1322-1332.

Falush, D., Stephens, M. and Pritchard, J. K. (2007) Inferences of population structure using multilocus genotype data: dominant markers and null alleles. *Molecular Ecology Notes* **7**, 574-578.

See Also

`read.Structure`, `write.GeneMapper`, `write.GenoDive`, `write.SPAGeDi`, `write.ATetra`, `write.Tetrasat`, `write.POPDIST`

Examples

```
# input genotype data (this is usually done by reading a file)
mygendata <- new("genambig", samples = c("ind1","ind2","ind3",
                                         "ind4","ind5","ind6"),
                 loci = c("locus1","locus2"))
Genotypes(mygendata) <- array(list(c(100,102,106,108,114,118),c(102,110),
                      c(98,100,104,108,110,112,116),c(102,106,112,118),
                      c(104,108,110),c(-9),
                      c(204),c(206,208,210,212,220,224,226),
                      c(202,206,208,212,214,218),c(200,204,206,208,212),
                      c(-9),c(202,206)),
                 dim=c(6,2))
Ploidies(mygendata) <- c(6,6,6,4,4,4)
# Note that some of the above genotypes have more or fewer alleles than
# the ploidy of the sample.

# create a vector of sample names to be used.  Note that this excludes
#  ind6.
mysamples <- c("ind1","ind2","ind3","ind4","ind5")

# Create an array containing data for additional columns to be written
# to the file.  You might also prefer to just read this and the ploidies
# in from a file.
myexcols <- array(data=c(1,2,1,2,1,1,1,0,0,0),dim=c(5,2),
                  dimnames=list(mysamples, c("PopData","PopFlag")))

# Write the Structure file, with six rows per individual.
# Since outfile="", the data will be written to the console instead of a file.
write.Structure(mygendata, 6, "", samples = mysamples, writepopinfo = FALSE,
                extracols = myexcols)
```

`write.Tetrasat` *Write Genotype Data in Tetrasat Format*

Description

Given a `genambig` object, `write.Tetrasat` creates a file that can be read by the software Tetrasat and Tetra.

Usage

```
write.Tetrasat(object, samples = Samples(object),
               loci = Loci(object), file = "")
```

Arguments

`object`	A `genambig` object containing the dataset of interest. Genotypes, population identities, microsatellite repeat lengths, and the dataset description of `object` are used by the function.

samples	A character vector of samples to write to the file. Should be a subset of Samples(object).
loci	A character vector of loci to write to the file. Should be a subset of Loci(object).
file	A character string indicating the file to which to write.

Details

Tetrasat files are space-delimited text files in which all alleles at a locus are concatenated into a string eight characters long. Population names or numbers are not used in the file, but samples are ordered by population, with the line "Pop" delimiting populations.

write.Tetrasat divides each allele by the length of the repeat and rounds down in order to convert alleles to repeat numbers. If necessary, it subtracts a multiple of 10 from all alleles at a locus to make all allele values less than 100, or puts a zero in front of the number if it only has one digit. If the individual is fully homozygous at a locus, the single allele is repeated four times. If any genotype has more than four alleles, write.Tetrasat picks a random sample of four alleles without replacement, and prints a warning. Missing data are represented by blank spaces.

Sample names should be a maximum of 20 characters long in order for the file to be read correctly by Tetrasat or Tetra.

Value

A file is written but no value is returned.

Author(s)

Lindsay V. Clark

References

Markwith, S. H., Stewart, D. J. and Dyer, J. L. (2006) TETRASAT: a program for the population analysis of allotetraploid microsatellite data. *Molecular Ecology Notes* **6**, 586-589.

Liao, W. J., Zhu, B. R., Zeng, Y. F. and Zhang, D. Y. (2008) TETRA: an improved program for population genetic analysis of allotetraploid microsatellite data. *Molecular Ecology Resources* **8**, 1260–1262.

See Also

read.Tetrasat, write.GeneMapper, write.ATetra, write.POPDIST

Examples

```
# set up sample data (usually done by reading files)
mysamples <- c("ind1", "ind2", "ind3", "ind4")
myloci <- c("loc1", "loc2")
mygendata <- new("genambig", samples = mysamples, loci = myloci)
mygendata <- reformatPloidies(mygendata, output="one")
Usatnts(mygendata) <- c(2, 3)
Genotypes(mygendata, loci="loc1") <- list(c(202,204), c(204),
                                          c(200,206,208,212),
                                          c(198,204,208))
```

```
Genotypes(mygendata, loci="loc2") <- list(c(78,81,84),
                                          c(75,90,93,96,99),
                                          c(87), c(-9))
PopInfo(mygendata) <- c(1,2,1,2)
Description(mygendata) <- "An example for write.Tetrasat."
Ploidies(mygendata) <- 4

## Not run:
# write a Tetrasat file
write.Tetrasat(mygendata, file="tetrasattest.txt")

# view the file
cat(readLines("tetrasattest.txt"),sep="\n")

## End(Not run)
```

Index