Package 'polysat'

September 27, 2010

Version 1.0

Date 2010-09-27

Title Tools for Polyploid Microsatellite Analysis

Author Lindsay V. Clark <lvclark@ucdavis.edu>

Maintainer Lindsay V. Clark <lvclark@ucdavis.edu>

Depends combinat, methods

Suggests ade4, adegenet

Description polysat is 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, and binary presence/absence formats. It can calculate pairwise distances between individuals using a stepwise mutation model or infinite alleles model. It can assist the user in estimating the ploidy of samples, and lastly it can estimate allele frequencies in populations and calculate pairwise Fst values based on those frequencies.

License GPL (>=2)

URL http://openwetware.org/wiki/Polysat

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)
Loci(object) <- value
PopInfo(object)
PopInfo(object) <- value
PopNames(object)
PopNames(object) <- value
PopNum(object, popname)
PopNum(object, popname) <- value
Ploidies(object)
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 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. (optional)
`loci`	A character or numeric vector indicating loci for which to return genotypes. (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 (integer or can be coerced to integer) indicating the ploidy of each sample. • 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. `Samples` and `Loci` return character vectors taken from the `names` of other slots (`Ploidies`/`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) <- c(4,4,4)
# or
Ploidies(mygen) <- rep(4, times = length(Samples(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="lH") <- 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", "lH") <- c(200, 204, 210)
# retrieve a single genotype
Genotype(mygen, "c", "lG")
```

```
# change a genotype to being missing
Genotype(mygen, "c", "lH") <- 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")
```

`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=9, 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.

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`

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))
```

---

calcFst *Calculate Wright's Pairwise FST*

Description

Given a data frame of allele frequencies and population sizes, `calcFst` calculates a matrix of pairwise Fst values.

Usage

```
calcFst(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 and population sizes such as that produced by simpleFreq or deSilvaFreq. Each population is in one row, and a column called Genomes 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.
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.

Details

calcFst works by calculating HS and HT for each locus for each pair of populations, then averaging HS and HT across loci. FST is then calculated for each pair of populations as (HT-HS)/HT.

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 HT, allele frequencies between two populations are averaged before the calculation. To calculate HS, 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).

Value

A square matrix containing FST values. The rows and columns of the matrix are both named by population.

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.

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)
Ploidies(mygenotypes) <- c(2,2,4,4,2,4)

# calculate allele frequencies
myfreq <- simpleFreq(mygenotypes)

# calculate pairwise FST
myfst <- calcFst(myfreq)

# examine the results
myfst
```

`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` and `@Ploidies` or `@Usatnts` slots, respectively. 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)
```

Arguments

object	A genambig or genbinary object containing the dataset of interest. Ploidies must be filled in for samples. All ploidies for samples 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` (which is similar 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`.

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

Examples

```
# 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))
Ploidies(mygen) <- rep(4, 200)
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
```

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

```
# 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))
Ploidies(mygen) <- rep(4, times = 3)

# 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.

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

```
# 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 and Jasieniuk, unpublished data

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)
```

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`.

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

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"`. For other slots, vectors filled with `NA` will be generated and will be named by samples (for `PopInfo` and `Ploidies`) 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 `"integer"`. This vector must be named by sample names. Each element represents the ploidy of that sample. `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")`: 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")`: 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")`: 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")`: Returns the ploidies of samples in the dataset (`object@Ploidies`).

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 are ignored; sample 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.

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.

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)

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

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,
                    ...)
```

Arguments

`object`	A `genambig` object containing the genotypes to be analyzed. If `distmetric = Bruvo.distance`, the `Usatnts` slot should be filled in.
`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`.
`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`) 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.

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`, `Lynch.distance`, `meandist.from.array`

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]]
```

`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.
...	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.

`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. All ploidies in the `genambig` object are 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`

Examples

```
# create a file to be read
# (this would normally be done in a text editor or with ATetra's Excel template)
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 = "atetraexample.txt", sep = "\n")

# Read the file and examine the data
exampledata <- read.ATetra("atetraexample.txt")
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)
```

Arguments

`infiles`	A character vector of paths to the files to be read.

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. 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.

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.

Author(s)

Lindsay V. Clark

References

http://www.appliedbiosystems.com/genemapper

See Also

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

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)
write.table(gentable, file="readGMtest.txt", quote=FALSE, sep="\t",
            na="", row.names=FALSE, col.names=TRUE)

# read the file
mygenotypes <- read.GeneMapper("readGMtest.txt")

# 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

Examples

```
# create data file (normally done in a text editor or spreadsheet software)
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 = "genodiveExample.txt", sep = "\n")

# import file data
exampledata <- read.GenoDive("genodiveExample.txt")

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

---

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.

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

http://ebe.ulb.ac.be/ebe/Software_files/manual_SPAGeDi_1-3.pdf

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`

Examples

```
# create a file to read (usually done with spreadsheet software or a
# text editor):
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="SpagInputExample.txt")

# display the file
cat(readLines("SpagInputExample.txt"), sep="\n")

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

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

---

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)
```

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.

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.

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

http://pritch.bsd.uchicago.edu/structure_software/release_versions/v2.3.3/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

Examples

```
# create a file to read (normally done in a text editor or spreadsheet
# software)
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="structtest.txt")

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

# read the structure file into genotypes and populations
testdata <- read.Structure("structtest.txt", 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, because all individuals should be tetraploid. The first line of the file is put into the `Description` slot.

Value

A `genambig` object containing data from the file.

Author(s)

Lindsay V. Clark

References

http://markwith.freehomepage.com/tetrasat.html

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.

http://ecology.bnu.edu.cn/zhangdy/TETRA/TETRA.htm

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`

Examples

```
## Not run:
# example from the Tetrasat website
mydata <- read.Tetrasat("http://markwith.freehomepage.com/sample.txt")
summary(mydata)
viewGenotypes(mydata, loci="A1_Gtype")

## End(Not run)

# example with defined data:
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="TetrasatExample.txt")
mydata2 <- read.Tetrasat("TetrasatExample.txt")

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

`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]. (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 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. The first column is called `Genomes` and contains the number of genomes in each population. 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"))
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
```

---

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 and Jasieniuk, unpublished data

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`

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)
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) <- c(4,4,4,4)
Description(mygendata) <- "Example for write.ATetra."

# write an ATetra file
write.ATetra(mygendata, file="atetratest.txt")

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

`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. The first column is called "Genomes" and contains the sizes of the populations in number of genomes. 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

http://ebe.ulb.ac.be/ebe/Software_files/manual_SPAGeDi_1-3.pdf

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

```
# 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))
Ploidies(mygen) <- rep(4, 80)
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")
```

`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 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]. 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

http://www.appliedbiosystems.com/genemapper

See Also

read.GeneMapper, write.Structure, write.GenoDive, write.Tetrasat, write.ATetra, 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)
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) <- c(6,6,6,6)

# write a GeneMapper file
write.GeneMapper(mygendata, "exampleGMoutput.txt")

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

---

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.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)
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"

# 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")
```

`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, 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

http://ebe.ulb.ac.be/ebe/Software_files/manual_SPAGeDi_1-3.pdf

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`

Examples

```
# set up data to write (usually read from a file)
mygendata <- new("genambig", samples = c("ind1","ind2","ind3","ind4"),
                 loci = c("loc1", "loc2"))
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"))

# write a file
write.SPAGeDi(mygendata, spatcoord = myspatcoord,
              file="SpagOutExample.txt")
```

---

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 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] 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] 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]` 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]` is less than `ploidy`, and `length(Genotype(object, x, loc))` is equal to `Ploidies(object)[x]`, the genotype data are used as is and missing data symbols are inserted in the extra rows.

5) If `Ploidies(object)[x]` is less than `ploidy`, and `length(Genotype(object, x, loc))` is less than `Ploidies(object)[x]`, the first allele is duplicated as many times as necessary for there to be as many alleles as `Ploidies(object)[x]`, and missing data symbols are inserted in the extra rows.

6) If `Ploidies(object)[x]` is less than `ploidy`, and `length(Genotype(object, x, loc))` is greater than `Ploidies(object)[x]`, 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.

The output file requires a few small modifications, done in a text editor or spreadsheet software, in order to be read by Structure. In the upper left corner the words "rowlabel" and "missing" should be deleted. Likewise the first and second rows for any non-locus columns should be deleted if the `extracols` argument was used and/or if `writepopinfo == TRUE`. These should include "PopInfo" and the second dimension names used in `extracols`, and zeros, respectively.

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

http://pritch.bsd.uchicago.edu/structure_software/release_versions/v2.3.3/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`

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

http://markwith.freehomepage.com/tetrasat.html

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.

http://ecology.bnu.edu.cn/zhangdy/TETRA/TETRA.htm

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`,

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)
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) <- c(4,4,4,4)

# write a Tetrasat file
```

```
write.Tetrasat(mygendata, file="tetrasattest.txt")

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

Index