Package 'adegenet'

June 22, 2011

Version 1.3-0

Date 2011/06/22

Title adegenet: a R package for the multivariate analysis of genetic markers.

Author Thibaut Jombart <t.jombart@imperial.ac.uk>
with contributions of: Ismail Ahmed, Peter Solymos
and contributed datasets from: Katayoun Moazami-
Goudarzi, Denis Laloe,Dominique Pontier, Daniel Maillard, Francois Balloux

Maintainer Thibaut Jombart <t.jombart@imperial.ac.uk>

Suggests genetics, spdep, tripack, ape, pegas, seqinr, multicore

Depends methods, MASS, ade4

Description Classes and functions for genetic data analysis within the multivariate framework.

Collate classes.R basicMethods.R handling.R auxil.R setAs.R SNPbin.R glHandle.R glFunctions.R glSim.R find.clust.R hybridize.R scale.R fstat.R import.R seqTrack.R chooseCN.R genind2genpop.R loadingplot.R sequences.R gstat.randtest.R makefreq.R colorplot.R monmonier.R spca.R coords.monmonier.R haploGen.R old2new.R spca.rtests.R dapc.R haploPop.R PCtest.R dist.genpop.R Hs.R propShared.R export.R HWE.R propTyped.R inbreeding.R glPlot.R zzz.R

License GPL (>=2)

LazyLoad yes

R topics documented:

`adegenet-package` *The adegenet package*

Description

This package is devoted to the multivariate analysis of genetic markers data. These data can be codominant markers (e.g. microsatellites) or presence/absence data (e.g. AFLP), and have any level of ploidy. 'adegenet' defines three formal (S4) classes:
- genind: a class for data of individuals ("genind" stands for genotypes-individuals).
- genpop: a class for data of groups of individuals ("genpop" stands for genotypes-populations)
- genlight: a class for genome-wide SNP data

For more information about these classes, type "class ? genind", "class ? genpop", or "?genlight".

Essential functionalities of the package are presented througout 4 tutorial vignettes, accessible using `vignette("name-below", package="adegenet")`:
- adegenet-basics: introduction to the package.
- adegenet-spca: multivariate analysis of spatial genetic patterns.
- adegenet-dapc: population structure and group assignment using DAPC.
- adegenet-genomics: introduction to the class genlight for the handling and analysis of genome-wide SNP data.

Important functions are also summarized below.

=== IMPORTING DATA ===
= TO GENIND OBJECTS =
`adegenet` imports data to genind object from the following softwares:
- STRUCTURE: see `read.structure`
- GENETIX: see `read.genetix`
- FSTAT: see `read.fstat`
- Genepop: see `read.genepop`
To import data from any of these formats, you can also use the general function `import2genind`.

In addition, it can extract polymorphic sites from nucleotide and amino-acid alignments:
- DNA files: use `read.dna` from the ape package, and then extract SNPs from DNA alignments using `DNAbin2genind`.

- protein sequences alignments: polymorphic sites can be extracted from protein sequences alignments in `alignment` format (package `seqinr`, see `as.alignment`) using the function `alignment2genind`.

It is also possible to read genotypes coded by character strings from a data.frame in which genotypes are in rows, markers in columns. For this, use `df2genind`. Note that `df2genind` can be used for any level of ploidy.

= TO GENLIGHT OBJECTS =
SNP data can be read from the following formats:
- PLINK: see function `read.PLINK`
- .snp (adegenet's own format): see function `read.snp`

SNP can also be extracted from aligned DNA sequences with the fasta format, using `fasta2genlight`

=== EXPORTING DATA ===
`adegenet` exports data from genind object to formats recognized by other R packages:
- the genetics package: see `genind2genotype`
- the hierfstat package: see `genind2hierfstat`

Genotypes can also be recoded from a genind object into a data.frame of character strings, using any separator between alleles. This covers formats from many softwares like GENETIX or STRUCTURE. For this, see `genind2df`.

=== MANIPULATING DATA ===
Several functions allow one to manipulate genind or genpop objects
- `genind2genpop`: convert a genind object to a genpop
- `seploc`: creates one object per marker; for genlight objects, creates blocks of SNPs.
- `seppop`: creates one object per population
- `na.replace`: replaces missing data (NA) in an approriate way
- `truenames`: restores true names of an object (genind and genpop use generic labels)
- x[i,j]: create a new object keeping only genotypes (or populations) indexed by 'i' and the alleles indexed by 'j'.
- `makefreq`: returns a table of allelic frequencies from a genpop object.
- `repool` merges genoptypes from different gene pools into one single genind object.
- `propTyped` returns the proportion of available (typed) data, by individual, population, and/or locus.
- `selPopSize` subsets data, retaining only genotypes from a population whose sample size is above a given level.
- `pop` sets the population of a set of genotypes.

=== ANALYZING DATA ===
Several functions allow to use usual, and less usual analyses:
- `HWE.test.genind`: performs HWE test for all populations and loci combinations
- `pairwise.fst`: computes simple pairwise Fst between populations
- `dist.genpop`: computes 5 genetic distances among populations.
- `monmonier`: implementation of the Monmonier algorithm, used to seek genetic boundaries among individuals or populations. Optimized boundaries can be obtained using `optimize.monmonier`. Object of the class `monmonier` can be plotted and printed using the corresponding methods.
- `spca`: implements Jombart et al. (in revision) spatial Principal Component Analysis
- `global.rtest`: implements Jombart et al. (2008) test for global spatial structures
- `local.rtest`: implements Jombart et al. (2008) test for local spatial structures
- `propShared`: computes the proportion of shared alleles in a set of genotypes (i.e. from a genind object)
- `propTyped`: function to investigate missing data in several ways

- scaleGen: generic method to scale genind or genpop before a principal component analysis
- Hs: computes the average expected heterozygosity by population in a genpop. Classically Used as a measure of genetic diversity.
- find.clusters and dapc: implement the Discriminant Analysis of Principal Component (DAPC, Jombart et al., 2010).
- seqTrack: implements the SeqTrack algorithm for recontructing transmission trees of pathogens (Jombart et al., 2010) .
glPca: implements PCA for genlight objects.

=== GRAPHICS ===
- colorplot: plots points with associated values for up to three variables represented by colors using the RGB system; useful for spatial mapping of principal components.
- loadingplot: plots loadings of variables. Useful for representing the contribution of alleles to a given principal component in a multivariate method.
- scatter.dapc: scatterplots for DAPC results.
- compoplot: plots membership probabilities from a DAPC object.

=== SIMULATING DATA ===
- hybridize: implements hybridization between two populations.
- haploGen: simulates genealogies of haplotypes, storing full genomes.
- haploPop: simulates populations of haplotypes, using different population dynamics, storing SNPs (under development).
- glSim: simulates simple genlight objects.

=== DATASETS ===
- H3N2: Seasonal influenza (H3N2) HA segment data.
- dapcIllus: Simulated data illustrating the DAPC.
- eHGDP: Extended HGDP-CEPH dataset.
- microbov: Microsatellites genotypes of 15 cattle breeds.
- nancycats: Microsatellites genotypes of 237 cats from 17 colonies of Nancy (France).
- rupica: Microsatellites genotypes of 335 chamois (Rupicapra rupicapra) from the Bauges mountains (France).
- sim2pop: Simulated genotypes of two georeferenced populations.
- spcaIllus: Simulated data illustrating the sPCA.

For more information, visit the adegenet website by typing `adegenetWeb()`.

To cite adegenet, please use the reference given by `citation("adegenet")` (or see reference below).

Details

Package:	adegenet
Type:	Package
Version:	1.3-0
Date:	2011-06-22
License:	GPL (>=2)

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>
with contributions of: Ismail Ahmed, Peter Solymos
and contributed datasets from: Katayoun Moazami-Goudarzi, Denis Laloë, Dominique Pontier, Daniel Maillard, Francois Balloux.

References

Jombart T. (2008) adegenet: a R package for the multivariate analysis of genetic markers *Bioinformatics* 24: 1403-1405. doi: 10.1093/bioinformatics/btn129

Jombart T, Devillard S and Balloux F (2010) Discriminant analysis of principal components: a new method for the analysis of genetically structured populations. BMC Genetics 11:94. doi:10.1186/1471-2156-11-94

Jombart T, Eggo R, Dodd P, Balloux F (2010) Reconstructing disease outbreaks from genetic data: a graph approach. *Heredity*. doi: 10.1038/hdy.2010.78.

Jombart, T., Devillard, S., Dufour, A.-B. and Pontier, D. Revealing cryptic spatial patterns in genetic variability by a new multivariate method. *Heredity*, **101**, 92–103.

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

Please post your questions on 'the adegenet forum': adegenet-forum@lists.r-forge.r-project.org

See Also

adegenet is related to several packages, in particular:
- `ade4` for multivariate analysis
- `ape` for phylogenetics and DNA data handling
- `pegas` for population genetics tools
- `seqinr` for handling nucleic and proteic sequences

---

a-score *Compute and optimize a-score for Discriminant Analysis of Principal Components (DAPC)*

Description

These functions are under development. Please email the author before using them for published results.

Usage

```
a.score(x, n.sim=10, ...)

optim.a.score(x, n.pca=1:ncol(x$tab), smart=TRUE, n=10, plot=TRUE, n.sim=10, n.d
```

Arguments

x	a dapc object.
n.pca	a vector of integers indicating the number of axes retained in the Principal Component Analysis (PCA) steps of DAPC. nsim DAPC will be run for each value in n.pca, unless the smart approach is used (see details).
smart	a logical indicating whether a smart, less computer-intensive approach should be used (TRUE, default) or not (FALSE). See details section.
n	an integer indicating the numbers of values spanning the range of n.pca to be used in the smart approach.
plot	a logical indicating whether the results should be displayed graphically (TRUE, default) or not (FALSE).
n.sim	an integer indicating the number of simulations to be performed for each number of retained PC.
n.da	an integer indicating the number of axes retained in the Discriminant Analysis step.
...	further arguments passed to other methods; currently unused..

Details

The Discriminant Analysis of Principal Components seeks a reduced space inside which observations are best discriminated into pre-defined groups. One way to assess the quality of the discrimination is looking at re-assignment of individuals to their prior group, successful re-assignment being a sign of strong discrimination.

However, when the original space is very large, ad hoc solutions can be found, which discriminate very well the sampled individuals but would perform poorly on new samples. In such a case, DAPC re-assignment would be high even for randomly chosen clusters. The a-score measures this bias. It is computed as (Pt-Pr), where Pt is the reassignment probability using the true cluster, and Pr is the reassignment probability for randomly permuted clusters. A a-score close to one is a sign that the DAPC solution is both strongly discriminating and stable, while low values (toward 0 or lower) indicate either weak discrimination or instability of the results.

The a-score can serve as a criterion for choosing the optimal number of PCs in the PCA step of DAPC, i.e. the number of PC maximizing the a-score. Two procedures are implemented in optim.a.score. The smart procedure selects evenly distributed number of PCs in a pre-defined range, compute the a-score for each, and then interpolate the results using splines, predicting an approximate optimal number of PCs. The other procedure (when smart is FALSE) performs the computations for all number of PCs request by the user. The 'optimal' number is then the one giving the highest mean a-score (computed over the groups).

Value

=== a.score ===
a.score returns a list with the following components:

tab	a matrix of a-scores with groups in columns and simulations in row.
pop.score	a vector giving the mean a-score for each population.
mean	the overall mean a-score.

=== optim.a.score ===
`optima.score` returns a list with the following components:

`pop.score`	a list giving the mean a-score of the populations for each number of retained PC (each element of the list corresponds to a number of retained PCs).
`mean`	a vector giving the overall mean a-score for each number of retained PCs.
`pred`	(only when `smart` is TRUE) the predictions of the spline, given in x and y coordinates.
`best`	the optimal number of PCs to be retained.

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

References

Jombart T, Devillard S and Balloux F (2010) Discriminant analysis of principal components: a new method for the analysis of genetically structured populations. BMC Genetics11:94. doi:10.1186/1471-2156-11-94

See Also

- `find.clusters`: to identify clusters without prior.
- `dapc`: the Discriminant Analysis of Principal Components (DAPC)

Accessors *Accessors for adegenet objects*

Description

An accessor is a function that allows to interact with slots of an object in a convenient way. Several accessors are available for genind or genpop objects. The operator "\$" and "\$<-" are used to access the slots, being equivalent to "@" and "@<-".

The operator "[" can be used to access components of the matrix slot "@tab", returning a genind or genpop object. This syntax is the same as for a matrix; for instance:
- "obj[,]" returns "obj"
- "obj[1:10,]" returns an object with only the first 10 genotypes (if "obj" is a genind) or the first 10 populations (if "obj" is a genpop) of "obj"
- "obj[1:10, 5:10]" returns an object keeping the first 10 entities and the alleles 5 to 10.
- "obj[loc=c("L1","L3")]" returns an object keeping only the loci specified in the `loc` argument (using generic names, not true names; in this example, only the first and the third locus would be retained)
- "obj[1:3, drop=TRUE]" returns the first 3 genotypes/populations of "obj", but retaining only alleles that are present in this subset (as opposed to keeping all alleles of "obj", which is the default behavior).

The argument `treatOther` handles the treatment of objects in the `@other` slot (see details). The argument `drop` can be set to TRUE to drop alleles that are no longer represented in the subset.

Usage

```
nInd(x, ...)
nLoc(x, ...)
pop(x)
indNames(x, ...)
## S4 method for signature 'genind'
indNames(x, ...)
locNames(x, ...)
## S4 method for signature 'genind'
locNames(x, withAlleles=FALSE, ...)
## S4 method for signature 'genpop'
locNames(x, withAlleles=FALSE, ...)
ploidy(x, ...)
## S4 method for signature 'genind'
ploidy(x, ...)
## S4 method for signature 'genpop'
ploidy(x, ...)
## S4 method for signature 'genind'
other(x, ...)
## S4 method for signature 'genpop'
other(x, ...)
```

Arguments

`x`	a genind or a genpop object.
`withAlleles`	a logical indicating whether the result should be of the form [locus name].[allele name], instead of [locus name].
`...`	further arguments to be passed to other methods (currently not used).

Details

The "[" operator can treat elements in the `@other` slot as well. For instance, if `obj@other$xy` contains spatial coordinates, the `obj[1:3,]@other$xy` will contain the spatial coordinates of the genotypes (or population) 1,2 and 3. This is handled through the argument `treatOther`, a logical defaulting to TRUE. If set to FALSE, the `@other` returned unmodified.

Note that only matrix-like, vector-like and lists can be proceeded in `@other`. Other kind of objects will issue a warning an be returned as they are, unless the argument `quiet` is left to TRUE, its default value.

The `drop` argument can be set to TRUE to retain only alleles that are present in the subset. To achieve better control of polymorphism of the data, see `isPoly`.

Value

A genind or genpop object.

Methods

nInd returns the number of individuals in the `genind` object

nLoc returns the number of loci of the object

pop returns the population factor of the object, using true (as opposed to generic) levels.

pop<- replacement method for the `@pop` slot of an object. The content of `@pop` and `@pop.names` is updated automatically.

indNames returns the true names of individuals.

indNames<- sets the true names of individuals using a vector of length nInd(x).

locNames returns the true names of markers and/or alleles.

locNames<- sets the true names of markers using a vector of length nLoc(x).

ploidy returns the ploidy of the data.

ploidy<- sets the ploidy of the data using an integer.

alleles returns the alleles of each locus.

alleles<- sets the alleles of each locus using a list with one character vector for each locus.

other returns the content of the `@other` slot (misc. information); returns `NULL` if the slot is empty or of length zero.

other<- sets the content of the `@other` slot (misc. information); the provided value needs to be a list; it not, provided value will be stored within a list.

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

Examples

```
data(nancycats)
nancycats
pop(nancycats) # get the populations
indNames(nancycats) # get the labels of individuals
locNames(nancycats) # get the labels of the loci
alleles(nancycats) # get the alleles

# let's isolate populations 4 and 8
temp <- nancycats@pop=="P04" | nancycats@pop=="P08"
obj <- nancycats[temp,]
obj

pop(obj)

# let's isolate two markers, fca23 and fca90
locNames(nancycats)

# they correspond to L2 and L7
nancycats$loc.fac
temp <- nancycats$loc.fac=="L2" | nancycats$loc.fac=="L7"
obj <- nancycats[,temp]
obj

obj$loc.fac
locNames(obj)

# or more simply
nancycats[loc=c("L2","L7")]
obj$loc.fac
locNames(obj)
```

```
# using 'drop':
truenames(nancycats[1:2])$tab
truenames(nancycats[1:2, drop=TRUE])$tab

# illustrate how 'other' slot is handled
colonies <- genind2genpop(nancycats)
colonies@other$aChar <- "This will not be proceeded"
colonies123 <- colonies[1:3]
colonies
colonies@other$xy

# illustrate pop
obj <- nancycats[sample(1:100,10)]
obj$pop
obj$pop.names
pop(obj)
pop(obj) <- rep(c('b','a'), each=5)
obj$pop
obj$pop.names
pop(obj)

# illustrate locNames
locNames(obj)
locNames(obj, withAlleles=TRUE)
```

`as methods in adegenet`

Converting genind/genpop objects to other classes

Description

These S3 and S4 methods are used to coerce genind and genpop objects to matrix-like objects. In most cases, this is equivalent to calling the `@tab` slot. An exception to this is the convertion to `ktab` objects used in the ade4 package as inputs for K-tables methods (e.g. Multiple Coinertia Analysis).

Usage

```
as(object, Class)
```

Arguments

`object` a genind or a genpop object.

`Class` the name of the class to which the object should be coerced, for instance `"data.frame"` or `"matrix"`.

Methods

coerce from one object class to another using `as(object,"Class")`, where the `object` is of the old class and the returned object is of the new class `"Class"`.

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

Examples

```
data(microbov)
x <- na.replace(microbov,method="0")
as(x[1:3],"data.frame")

## dudi functions attempt to convert their first argument
## to a data.frame; so they can be used on genind/genpop objects.
if(require(ade4)){
## perform a PCA
pca1 <- dudi.pca(x, scale=FALSE, scannf=FALSE)
pca1

x <- genind2genpop(microbov,miss="chi2")
x <- as(x,"ktab")
class(x)
## perform a STATIS analysis
statis1 <- statis(x, scannf=FALSE)
statis1
plot(statis1)
}
```

---

as.genlight *Conversion to class "genlight"*

Description

The class genlight is a formal (S4) class for storing a genotypes of binary SNPs in a compact way, using a bit-level coding scheme. New instances of this class are best created using new; see the manpage of genlight for more information on this point.

As a shortcut, conversion methods can be used to convert various objects into a genlight object. Conversions can be achieved using S3-style (as.genlight(x)) or S4-style (as(x,"genlight") procedures. All of them call upon the constructor (new) of genlight objects.

Conversion is currently available from the following objects: - matrix of type integer/numeric - data.frame with integer/numeric data - list of vectors of integer/numeric type

Author(s)

Thibaut Jombart (<t.jombart@imperial.ac.uk>)

See Also

Related class:
- SNPbin, for storing individual genotypes of binary SNPs

- genind

Examples

```
## data to be converted
dat <- list(toto=c(1,1,0,0,2,2,1,2,NA), titi=c(NA,1,1,0,1,1,1,0,0), tata=c(NA,0,3, NA,1,1

## using the constructor
x1 <- new("genlight", dat)
x1

## using 'as' methods
x2 <- as.genlight(dat)
x3 <- as(dat, "genlight")

identical(x1,x2)
identical(x1,x3)
```

`as.SNPbin` *Conversion to class "SNPbin"*

Description

The class SNPbin is a formal (S4) class for storing a genotype of binary SNPs in a compact way, using a bit-level coding scheme. New instances of this class are best created using `new`; see the manpage of SNPbin for more information on this point.

As a shortcut, conversion methods can be used to convert various objects into a SNPbin object. Conversions can be achieved using S3-style (`as.SNPbin(x)`) or S4-style (`as(x,"SNPbin")`) procedures. All of them call upon the constructor (`new`) of SNPbin objects.

Conversion is currently available from the following objects: - integer vectors - numeric vectors

Author(s)

Thibaut Jombart (<t.jombart@imperial.ac.uk>)

See Also

Related class:
- `SNPbin` - `genlight`, for storing multiple binary SNP genotypes.

Examples

```
## data to be converted
dat <- c(1,0,0,2,1,1,1,2,2,1,1,0,0,1)

## using the constructor
x1 <- new("SNPbin", dat)
x1

## using 'as' methods
x2 <- as.SNPbin(dat)
x3 <- as(dat, "SNPbin")
```

```
identical(x1,x2)
identical(x1,x3)
```

`Auxiliary functions`

Utilities functions for adegenet

Description

These functions are mostly auxiliary procedures used internally in adegenet, with the exception of `adegenetWeb`, which opens the adegenet website in the default navigator.

The other functions are:
- `checkType`: checks the type of markers being used in a function and issues an error if appropriate.
- `.rmspaces`: remove peripheric spaces in a character string.
- `.genlab`: generate labels in a correct alphanumeric ordering.
- `.readExt`: read the extension of a given file.
- `corner`: adds text to a corner of a figure.
- `num2col`: translates a numeric vector into colors.
- `transp`: adds transparency to a vector of colors. Note that transparent colors are not supported on some graphical devices.

Usage

```
adegenetWeb()
.genlab(base, n)
corner(text, posi="topleft",  inset=0.1, ...)
num2col(x, col.pal=heat.colors, reverse=FALSE,
               x.min=min(x), x.max=max(x), na.col="green")
transp(col, alpha=.5)
```

Arguments

`base`	a character string forming the base of the labels
`n`	the number of labels to generate
`text`	a character string to be added to the plot
`posi`	a character matching any combinations of "top/bottom" and "left/right".
`inset`	a vector of two numeric values (recycled if needed) indicating the inset, as a fraction of the plotting region.
`...`	further arguments to be passed to `text`
`x`	a numeric vector
`col.pal`	a function generating colors according to a given palette.
`reverse`	a logical stating whether the palette should be inverted (TRUE), or not (FALSE, default).
`x.min`	the minimal value from which to start the color scale

x.max	the maximal value from which to start the color scale
na.col	the color to be used for missing values (NAs)
col	a vector of colors
alpha	a numeric value between 0 and 1 representing the alpha coefficient; 0: total transparency; 1: no transparency.

Value

For .genlab, a character vector of size "n".

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

Examples

```
## Not run:
## this opens the adegenet website
adegenetWeb()


## End(Not run)

.genlab("Locus-",11)

## transparent colors using "transp"
plot(rnorm(1000), rnorm(1000), col=transp("blue",.3), pch=20, cex=4)


## numeric values to color using num2col
plot(1:100, col=num2col(1:100), pch=20, cex=4)
plot(1:100, col=num2col(1:100, col.pal=rainbow), pch=20, cex=4)
```

---

chooseCN *Function to choose a connection network*

Description

The function chooseCN is a simple interface to build a connection network (CN) from xy coordinates. The user chooses from 6 types of graph and one additional weighting scheme. chooseCN calls functions from appropriate packages, handles non-unique coordinates and returns a connection network either with classe nb or listw. For graph types 1-4, duplicated locations are not accepted and will issue an error.

Usage

```
chooseCN(xy, ask = TRUE, type = NULL, result.type = "nb", d1 = NULL,
    d2 = NULL, k = NULL,  a=NULL, dmin=NULL, plot.nb = TRUE, edit.nb = FALSE)
```

Arguments

xy	an matrix or data.frame with two columns for x and y coordinates.
ask	a logical stating whether graph should be chosen interactively (TRUE,default) or not (FALSE). Set to FALSE if type is provided.
type	an integer giving the type of graph (see details).
result.type	a character giving the class of the returned object. Either "nb" (default) or "listw", both from spdep package. See details.
d1	the minimum distance between any two neighbours. Used if type=5.
d2	the maximum distance between any two neighbours. Used if type=5. Can also be a character: "dmin" for the minimum distance so that each site has at least one connection, or "dmax" to have all sites connected (despite the later has no sense).
k	the number of neighbours per point. Used if type=6.
a	the exponent of the inverse distance matrix. Used if type=7.
dmin	the minimum distance between any two distinct points. Used to avoid infinite spatial proximities (defined as the inversed spatial distances). Used if type=7.
plot.nb	a logical stating whether the resulting graph should be plotted (TRUE, default) or not (FALSE).
edit.nb	a logical stating whether the resulting graph should be edited manually for corrections (TRUE) or not (FALSE, default).

Details

There are 7 kinds of graphs proposed:
Delaunay triangulation (type 1)
Gabriel graph (type 2)
Relative neighbours (type 3)
Minimum spanning tree (type 4)
Neighbourhood by distance (type 5)
K nearests neighbours (type 6)
Inverse distances (type 7)

The last option (type=7) is not a true neighbouring graph: all sites are neighbours, but the spatial weights are directly proportional to the inversed spatial distances.
Also not that in this case, the output of the function is always a listw object, even if nb was requested.

The choice of the connection network has been discuted on the adegenet forum. Please search the archives from adegenet website (section 'contact') using 'graph' as keyword.

Value

Returns a connection network having the class nb or listw. The xy coordinates are passed as attribute to the created object.

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

See Also

spca

Examples

```
data(nancycats)
if(require(spdep) & require(ade4)){

par(mfrow=c(2,2))
cn1 <- chooseCN(nancycats@other$xy,ask=FALSE,type=1)
cn2 <- chooseCN(nancycats@other$xy,ask=FALSE,type=2)
cn3 <- chooseCN(nancycats@other$xy,ask=FALSE,type=3)
cn4 <- chooseCN(nancycats@other$xy,ask=FALSE,type=4)
par(mfrow=c(1,1))
}
```

---

colorplot *Represents a cloud of points with colors*

Description

The colorplot function represents a cloud of points with colors corresponding to a combination of 1,2 or 3 quantitative variables, assigned to RGB (Red, Green, Blue) channels. For instance, this can be useful to represent up to 3 principal components in space. Note that the property of such representation to convey multidimensional information has not been investigated.

colorplot is a S3 generic function. Methods are defined for particular objects, like spca objects.

Usage

```
colorplot(...)

## Default S3 method:
colorplot(xy, X, axes=NULL, add.plot=FALSE, defaultLevel=0, transp=FALSE, alpha=
```

Arguments

xy	a numeric matrix with two columns (e.g. a matrix of spatial coordinates.
X	a matrix-like containing numeric values that are translated into the RGB system. Variables are considered to be in columns.
axes	the index of the columns of X to be represented. Up to three axes can be chosen. If null, up to the first three columns of X are used.
add.plot	a logical stating whether the colorplot should be added to the existing plot (defaults to FALSE).
defaultLevel	a numeric value between 0 and 1, giving the default level in a color for which values are not specified. Used whenever less than three axes are specified.
transp	a logical stating whether the produced colors should be transparent (TRUE) or not (FALSE, default).

alpha	the alpha level for transparency, between 0 (fully transparent) and 1 (not transparent); see ?rgb for more details.
...	further arguments to be passed to other methods. In colorplot.default, these arguments are passed to plot/points functions. See ?plot.default and ?points.

Value

Invisibly returns a vector of colours used in the plot.

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

Examples

```
# a toy example
xy <- expand.grid(1:10,1:10)
df <- data.frame(x=1:100, y=100:1, z=runif(100,0,100))
colorplot(xy,df,cex=10,main="colorplot: toy example")

# a genetic example using a sPCA
if(require(spdep) & require(ade4)){
data(spcaIllus)
dat3 <- spcaIllus$dat3
spca3 <- spca(dat3,xy=dat3$other$xy,ask=FALSE,type=1,plot=FALSE,scannf=FALSE,nfposi=1,nfn
colorplot(spca3, cex=4, main="colorplot: a sPCA example")
text(spca3$xy[,1], spca3$xy[,2], dat3$pop)
mtext("P1-P2 in cline\tP3 random \tP4 local repulsion")
}
```

---

coords.monmonier *Returns original points in results paths of an object of class 'monmonier'*

Description

The original implementation of monmonier in package **adegenet** returns path coordinates, coords.monmonier additionally displays identities of the original points of the network, based on original coordinates.

Usage

```
coords.monmonier(x)
```

Arguments

x	an object of class monmonier.

Value

Returns a list with elements according to the x$nrun result of the monmonier object. Corresponding path points are in the same order as in the original object.

run1 (run2, ...): for each run, a list containing a matrix giving the original points in the network (first and second, indicating pairs of neighbours). Path coordinates are stored in columns x.hw and y.hw. first and second are integers referring to the row numbers in the x$xy matrix of the original monmonier object.

Author(s)

Peter Solymos, <Solymos.Peter@aotk.szie.hu>, http://www.univet.hu/users/psolymos/personal/

See Also

monmonier

Examples

```
## Not run:
if(require(spdep) & require(ade4)){

load(system.file("files/mondata1.rda",package="adegenet"))
cn1 <- chooseCN(mondata1$xy,type=2,ask=FALSE)
mon1 <- monmonier(mondata1$xy,dist(mondata1$x1),cn1,threshold=2,nrun=3)

mon1$run1
mon1$run2
mon1$run3
path.coords <- coords.monmonier(mon1)
path.coords
}

## End(Not run)
```

---

dapc *Discriminant Analysis of Principal Components (DAPC)*

Description

These functions implement the Discriminant Analysis of Principal Components (DAPC, Jombart et al. 2010). This method descibes the diversity between pre-defined groups. When groups are unknown, use find.clusters to infer genetic clusters. See 'details' section for a succint description of the method, and vignette("adegenet-dapc") for a tutorial. Graphical methods for DAPC are documented in scatter.dapc (see ?scatter.dapc).

dapc is a generic function performing the DAPC on the following types of objects:
- data.frame (only numeric data)
- matrix (only numeric data)
- genind objects (genetic markers)
- genlight objects (genome-wide SNPs)

These methods all return an object with class dapc.

Functions that can be applied to these objects are (the ".dapc" can be ommitted):

- print.dapc: prints the content of a dapc object.
- summary.dapc: extracts useful information from a dapc object.
- predict.dapc: predicts group memberships based on DAPC results.

DAPC implementation calls upon dudi.pca from the ade4 package (except for genlight objects) and lda from the MASS package. The predict procedure uses predict.lda from the MASS package.

as.lda is a generic with a method for dapc object which converts these objects into outputs similar to that of lda.default.

Usage

```
## S3 method for class 'data.frame'
dapc(x, grp, n.pca=NULL, n.da=NULL, center=TRUE,
     scale=FALSE,var.contrib=TRUE, pca.info=TRUE, pca.select=c("nbEig","percVar"
     perc.pca=NULL, ..., dudi=NULL)

## S3 method for class 'matrix'
dapc(x, ...)

## S3 method for class 'genind'
dapc(x, pop=NULL, n.pca=NULL, n.da=NULL, scale=FALSE,
     scale.method=c("sigma", "binom"), truenames=TRUE, var.contrib=TRUE,
     pca.info=TRUE, pca.select=c("nbEig","percVar"), perc.pca=NULL, ...)

## S3 method for class 'genlight'
dapc(x, pop = NULL, n.pca = NULL, n.da = NULL, scale
    = FALSE, var.contrib = TRUE, pca.info=TRUE, pca.select = c("nbEig", "percVar
    perc.pca = NULL, glPca = NULL, ...)

## S3 method for class 'dudi'
dapc(x, grp, ...)

## S3 method for class 'dapc'
print(x, ...)

## S3 method for class 'dapc'
summary(object, ...)

## S3 method for class 'dapc'
predict(object, newdata, prior = object$prior, dimen,
         method = c("plug-in", "predictive", "debiased"), ...)
```

Arguments

x	a data.frame, matrix, or genind object. For the data.frame and matrix arguments, only quantitative variables should be provided.
grp,pop	a factor indicating the group membership of individuals; for scatter, an optional grouping of individuals.

n.pca — an integer indicating the number of axes retained in the Principal Component Analysis (PCA) step. If NULL, interactive selection is triggered.

n.da — an integer indicating the number of axes retained in the Discriminant Analysis step. If NULL, interactive selection is triggered.

center — a logical indicating whether variables should be centred to mean 0 (TRUE, default) or not (FALSE). Always TRUE for genind objects.

scale — a logical indicating whether variables should be scaled (TRUE) or not (FALSE, default). Scaling consists in dividing variables by their (estimated) standard deviation to account for trivial differences in variances. Further scaling options are available for genind objects (see argument scale.method).

var.contrib — a logical indicating whether the contribution of original variables (alleles, for genind objects) should be provided (TRUE, default) or not (FALSE). Such output can be useful, but can also create huge matrices when there is a lot of variables.

pca.info — a logical indicating whether information about the prior PCA should be stored (TRUE, default) or not (FALSE). This information is required to predict group membership of new individuals using predict, but makes the object slightly bigger.

pca.select — a character indicating the mode of selection of PCA axes, matching either "nbEig" or "percVar". For "nbEig", the user has to specify the number of axes retained (interactively, or via n.pca). For "percVar", the user has to specify the minimum amount of the total variance to be preserved by the retained axes, expressed as a percentage (interactively, or via perc.pca).

perc.pca — a numeric value between 0 and 100 indicating the minimal percentage of the total variance of the data to be expressed by the retained axes of PCA.

... — further arguments to be passed to other functions. For dapc.matrix, arguments are to match those of dapc.data.frame; for dapc.genlight, arguments passed to glPca

glPca — an optional glPca object; if provided, dimension reduction is not performed (saving computational time) but taken directly from this object.

object — a dapc object.

scale.method — a character specifying the scaling method to be used for allele frequencies, which must match "sigma" (usual estimate of standard deviation) or "binom" (based on binomial distribution). See scaleGen for further details.

truenames — a logical indicating whether true (i.e., user-specified) labels should be used in object outputs (TRUE, default) or not (FALSE).

dudi — optionally, a multivariate analysis with the class dudi (from the ade4 package). If provided, prior PCA will be ignored, and this object will be used as a prior step for variable orthogonalisation.

newdata — an optional dataset of individuals whose membership is seeked; can be a data.frame, a matrix, a genind or a genlight object, but object class must match the original ('training') data. In particular, variables must be exactly the same as in the original data. For genind objects, see repool to ensure matching of alleles.

prior,dimen,method — see ?predict.lda.

Details

The Discriminant Analysis of Principal Components (DAPC) is designed to investigate the genetic structure of biological populations. This multivariate method consists in a two-steps procedure. First, genetic data are transformed (centred, possibly scaled) and submitted to a Principal Component Analysis (PCA). Second, principal components of PCA are submitted to a Linear Discriminant Analysis (LDA). A trivial matrix operation allows to express discriminant functions as linear combination of alleles, therefore allowing one to compute allele contributions. More details about the computation of DAPC are to be found in the indicated reference.

DAPC does not infer genetic clusters ex nihilo; for this, see the `find.clusters` function.

Value

=== dapc objects ===
The class `dapc` is a list with the following components:

`call`	the matched call.
`n.pca`	number of PCA axes retained
`n.da`	number of DA axes retained
`var`	proportion of variance conserved by PCA principal components
`eig`	a numeric vector of eigenvalues.
`grp`	a factor giving prior group assignment
`prior`	a numeric vector giving prior group probabilities
`assign`	a factor giving posterior group assignment
`tab`	matrix of retained principal components of PCA
`loadings`	principal axes of DAPC, giving coefficients of the linear combination of retained PCA axes.
`ind.coord`	principal components of DAPC, giving the coordinates of individuals onto principal axes of DAPC; also called the discriminant functions.
`grp.coord`	coordinates of the groups onto the principal axes of DAPC.
`posterior`	a data.frame giving posterior membership probabilities for all individuals and all clusters.
`var.contr`	(optional) a data.frame giving the contributions of original variables (alleles in the case of genetic data) to the principal components of DAPC.

=== other outputs ===
Other functions have different outputs:
- `summary.dapc` returns a list with 6 components: `n.dim` (number of retained DAPC axes), `n.pop` (number of groups/populations), `assign.prop` (proportion of overall correct assignment), `assign.per.pop` (proportion of correct assignment per group), `prior.grp.size` (prior group sizes), and `post.grp.size` (posterior group sizes).

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

References

Jombart T, Devillard S and Balloux F (2010) Discriminant analysis of principal components: a new method for the analysis of genetically structured populations. BMC Genetics11:94. doi:10.1186/1471-2156-11-94

See Also

- scatter.dapc, assignplot, compoplot: graphics for DAPC.
- find.clusters: to identify clusters without prior.
- dapcIllus: a set of simulated data illustrating the DAPC
- eHGDP, H3N2: empirical datasets illustrating DAPC

Examples

```
## data(dapcIllus), data(eHGDP), and data(H3N2) illustrate the dapc
## see ?dapcIllus, ?eHGDP, ?H3N2
##

example(dapcIllus)
example(eHGDP)
example(H3N2)


## showing different scatter options ##
## !! more in ?scatter.dapc !! ##
data(H3N2)
pop(H3N2) <- factor(H3N2$other$epid)
dapc1 <- dapc(H3N2, var.contrib=FALSE, scale=FALSE, n.pca=150, n.da=5)

## remove internal segments and ellipses, different pch, add MStree
scatter(dapc1, cell=0, pch=18:23, cstar=0, mstree=TRUE, lwd=2, lty=2)

## only ellipse, custom labels
scatter(dapc1, cell=2, pch="", cstar=0, posi.da="top",
lab=paste("year\n",2001:2006), axesel=FALSE, col=terrain.colors(10))



## example using genlight objects ##
## simulate data
x <- glSim(50,4e3-50, 50, ploidy=2)
x
plot(x)

## perform DAPC
dapc1 <- dapc(x, n.pca=10, n.da=1)
dapc1

## plot results
scatter(dapc1, scree.da=FALSE)

## SNP contributions
loadingplot(dapc1$var.contr)
loadingplot(tail(dapc1$var.contr, 100), main="Loading plot - last 100 SNPs")


## USE "PREDICT" TO PREDICT GROUPS OF NEW INDIVIDUALS ##
## load data
data(sim2pop)

## we make a dataset of:
```

```
## 30 individuals from pop A
## 30 individuals from pop B
## 30 hybrids

## separate populations and make F1
temp <- seppop(sim2pop)
temp <- lapply(temp, function(e) hybridize(e,e,n=30)) # force equal popsizes

## make hybrids
hyb <- hybridize(temp[[1]], temp[[2]], n=30)

## repool data - needed to ensure allele matching
newdat <- repool(temp[[1]], temp[[2]], hyb)
pop(newdat) <- rep(c("pop A", "popB", "hyb AB"), c(30,30,30))

## perform the DAPC on the first 2 pop (60 first indiv)
dapc1 <- dapc(newdat[1:60],n.pca=5,n.da=1)

## plot results
scatter(dapc1)

## make prediction for the 30 hybrids
hyb.pred <- predict(dapc1, newdat[61:90])
hyb.pred

## plot the inferred coordinates (circles are hybrids)
points(hyb.pred$ind.scores, rep(.1, 30))

## look at assignment using assignplot
assignplot(dapc1, new.pred=hyb.pred)
title("30 indiv popA, 30 indiv pop B, 30 hybrids")

## image using compoplot
compoplot(dapc1, new.pred=hyb.pred, ncol=2)
title("30 indiv popA, 30 indiv pop B, 30 hybrids")


## show compoplot on microbov data ##
data(microbov)
dapc1 <- dapc(microbov, n.pca=20, n.da=15)
compoplot(dapc1, lab="")
```

`dapc graphics` *Graphics for Discriminant Analysis of Principal Components (DAPC)*

Description

These functions provide graphic outputs for Discriminant Analysis of Principal Components (DAPC, Jombart et al. 2010). See `?dapc` for details about this method. DAPC graphics are detailed in the DAPC tutorial accessible using `vignette("adegenet-dapc")`.

These functions all require an object of class `dapc` (the ".dapc" can be ommitted when calling the functions):
- `scatter.dapc`: produces scatterplots of principal components (or 'discriminant functions'), with a screeplot of eigenvalues as inset.

- `assignplot`: plot showing the probabilities of assignment of individuals to the different clusters.
- `compoplot`: barplot showing the probabilities of assignment of individuals to the different clusters.

Usage

```
## S3 method for class 'dapc'
scatter(x, xax=1, yax=2, grp=x$grp, col=rainbow(length(levels(grp))),
                         pch=20, bg="lightgrey", solid=.7, scree.da=TRUE,
                         scree.pca=FALSE, posi.da="bottomright",
                         posi.pca="bottomleft", bg.inset="white", ratio.da=.25,
                         ratio.pca=.25, inset.da=0.02, inset.pca=0.02,
                         inset.solid=.5, onedim.filled=TRUE, mstree=FALSE, lwd=1
                         lty=1, segcol="black", legend=FALSE, posi.leg="topright
                         cleg=1, txt.leg=levels(grp), cstar = 1, cellipse = 1.5,
                         axesell = FALSE, label = levels(grp), clabel = 1, xlim
                         NULL, ylim = NULL, grid = FALSE, addaxes = TRUE, origin
                         c(0,0), include.origin = TRUE, sub = "", csub = 1, poss
                         "bottomleft", cgrid = 1, pixmap = NULL, contour = NULL,
                         = NULL, ...)

assignplot(x, only.grp=NULL, subset=NULL, new.pred=NULL, cex.lab=.75,pch=3)

compoplot(x, only.grp=NULL, subset=NULL, new.pred=NULL, col=NULL, lab=NULL,
                  legend=TRUE, txt.leg=NULL, ncol=4, posi=NULL, cleg=.8, bg=tran
```

Arguments

`x`	a `dapc` object.
`xax,yax`	`integers` specifying which principal components of DAPC should be shown in x and y axes.
`grp`	a factor defining group membership for the individuals. The scatterplot is optimal only for the default group, i.e. the one used in the DAPC analysis.
`col`	a suitable color to be used for groups. The specified vector should match the number of groups, not the number of individuals.
`pch`	a `numeric` indicating the type of point to be used to indicate the prior group of individuals (see `points` documentation for more details); one value is expected for each group; recycled if necessary.
`bg`	the color used for the background of the scatterplot.
`solid`	a value between 0 and 1 indicating the alpha level for the colors of the plot; 0=full transparency, 1=solid colours.
`scree.da`	a logical indicating whether a screeplot of Discriminant Analysis eigenvalues should be displayed in inset (TRUE) or not (FALSE).
`scree.pca`	a logical indicating whether a screeplot of Principal Component Analysis eigenvalues should be displayed in inset (TRUE) or not (FALSE); retained axes are displayed in black.
`posi.da`	the position of the inset of DA eigenvalues; can match any combination of "top/bottom" and "left/right".

posi.pca — the position of the inset of PCA eigenvalues; can match any combination of "top/bottom" and "left/right".

bg.inset — the color to be used as background for the inset plots.

ratio.da — the size of the inset of DA eigenvalues as a proportion of the current plotting region.

ratio.pca — the size of the inset of PCA eigenvalues as a proportion of the current plotting region.

inset.da — a vector with two numeric values (recycled if needed) indicating the inset to be used for the screeplot of DA eigenvalues as a proportion of the current plotting region; see ?add.scatter for more details.

inset.pca — a vector with two numeric values (recycled if needed) indicating the inset to be used for the screeplot of PCA eigenvalues as a proportion of the current plotting region; see ?add.scatter for more details.

inset.solid — a value between 0 and 1 indicating the alpha level for the colors of the inset plots; 0=full transparency, 1=solid colours.

onedim.filled — a logical indicating whether curves should be filled when plotting a single discriminant function (TRUE), or not (FALSE).

mstree — a logical indicating whether a minimum spanning tree linking the groups and based on the squared distances between the groups inside the entire space should added to the plot (TRUE), or not (FALSE).

lwd, lty, segcol — the line width, line type, and segment colour to be used for the minimum spanning tree.

legend — a logical indicating whether a legend for group colours should added to the plot (TRUE), or not (FALSE).

posi.leg — the position of the legend for group colours; can match any combination of "top/bottom" and "left/right", or a set of x/y coordinates stored as a list (locator can be used).

cleg — a size factor used for the legend.

cstar, cellipse, axesell, label, clabel, xlim, ylim, grid, addaxes, origin, include.origin — arguments passed to s.class; see ?s.class for more informations

only.grp — a character vector indicating which groups should be displayed. Values should match values of x$grp. If NULL, all results are displayed

subset — integer or logical vector indicating which individuals should be displayed. If NULL, all results are displayed

new.pred — an optional list, as returned by the predict method for dapc objects; if provided, the individuals with unknown groups are added at the bottom of the plot. To visualize these individuals only, specify only.grp="unknown".

cex.lab — a numeric indicating the size of labels.

lab — a vector of characters (recycled if necessary) of labels for the individuals; if left to NULL, the row names of x$tab are used.

txt.leg — a character vector indicating the text to be used in the legend; if not provided, group names stored in x$grp are used.

ncol — an integer indicating the number of columns of the legend, defaulting to 4.

posi — a characther string indicating the position of the legend; can match any combination of "top/bottom" and "left/right". See ?legend.

... — further arguments to be passed to other functions. For scatter, arguments passed to points; for compoplot, arguments passed to barplot.

Details

See the documentation of dapc for more information about the method.

Value

All functions return the matched call.

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

References

Jombart T, Devillard S and Balloux F (2010) Discriminant analysis of principal components: a new method for the analysis of genetically structured populations. BMC Genetics11:94. doi:10.1186/1471-2156-11-94

See Also

- dapc: implements the DAPC.
- find.clusters: to identify clusters without prior.
- dapcIllus: a set of simulated data illustrating the DAPC
- eHGDP, H3N2: empirical datasets illustrating DAPC

Examples

```
data(H3N2)
dapc1 <- dapc(H3N2, pop=H3N2$other$epid, n.pca=30,n.da=6)

## defautl plot ##
scatter(dapc1)

## showing different scatter options ##
## remove internal segments and ellipses, different pch, add MStree
scatter(dapc1, pch=18:23, cstar=0, mstree=TRUE, lwd=2, lty=2, posi.da="topleft")

## only ellipse, custom labels, use insets
scatter(dapc1, cell=2, pch="", cstar=0, posi.pca="topleft", posi.da="topleft", scree.pca=
inset.pca=c(.01,.3), lab=paste("year\n",2001:2006), axesel=FALSE, col=terrain.colors(10))

## without ellipses, use legend for groups
scatter(dapc1, cell=0, cstar=0, scree.da=FALSE, clab=0, cex=3, solid=.4, bg="white", leg=

## only one axis
scatter(dapc1,1,1,scree.da=FALSE, legend=TRUE, solid=.4,bg="white")



## example using genlight objects ##
## simulate data
x <- glSim(50,4e3-50, 50, ploidy=2)
x
```

```
plot(x)

## perform DAPC
dapc2 <- dapc(x, n.pca=10, n.da=1)
dapc2

## plot results
scatter(dapc2, scree.da=FALSE, leg=TRUE, txt.leg=paste("group", c('A','B')), col=c("red",

## SNP contributions
loadingplot(dapc2$var.contr)
loadingplot(tail(dapc2$var.contr, 100), main="Loading plot - last 100 SNPs")



## assignplot / compoplot ##
assignplot(dapc1, only.grp=2006)

data(microbov)
dapc3 <- dapc(microbov, n.pca=20, n.da=15)
compoplot(dapc3, lab="")
```

`dapcIllus` *Simulated data illustrating the DAPC*

Description

Datasets illustrating the Discriminant Analysis of Principal Components (DAPC, Jombart et al. submitted).

These data were simulated using various models using Easypop (2.0.1). The `dapcIllus` is a list containing the following genind objects:
- "a": island model with 6 populations
- "b": hierarchical island model with 6 populations (3,2,1)
- "c": one-dimensional stepping stone with 2x6 populations, and a boundary between the two sets of 6 populations
- "d": one-dimensional stepping stone with 24 populations

See "source" for a reference providing simulation details.

Usage

```
data(dapcIllus)
```

Format

`dapcIllus` is list of 4 components being all genind objects.

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

Source

Jombart, T., Devillard, S. and Balloux, F. Discriminant analysis of principal components: a new method for the analysis of genetically structured populations. Submitted to *BMC genetics*.

References

Jombart, T., Devillard, S. and Balloux, F. Discriminant analysis of principal components: a new method for the analysis of genetically structured populations. Submitted to *Genetics*.

See Also

- `dapc`: implements the DAPC.
- `eHGDP`: dataset illustrating the DAPC and `find.clusters`.
- `H3N2`: dataset illustrating the DAPC.
- `find.clusters`: to identify clusters without prior.

Examples

```
if(require(MASS) & require(ade4)){

data(dapcIllus)
attach(dapcIllus)
a # this is a genind object, like b, c, and d.


## FINS CLUSTERS EX NIHILO
clust.a <- find.clusters(a, n.pca=100, n.clust=6)
clust.b <- find.clusters(b, n.pca=100, n.clust=6)
clust.c <- find.clusters(c, n.pca=100, n.clust=12)
clust.d <- find.clusters(d, n.pca=100, n.clust=24)

## examin outputs
names(clust.a)
lapply(clust.a, head)


## PERFORM DAPCs
dapc.a <- dapc(a, pop=clust.a$grp, n.pca=100, n.da=5)
dapc.b <- dapc(b, pop=clust.b$grp, n.pca=100, n.da=5)
dapc.c <- dapc(c, pop=clust.c$grp, n.pca=100, n.da=11)
dapc.d <- dapc(d, pop=clust.d$grp, n.pca=100, n.da=23)


## LOOK AT ONE RESULT
dapc.a
summary(dapc.a)

## FORM A LIST OF RESULTS FOR THE 4 DATASETS
lres <- list(dapc.a, dapc.b, dapc.c, dapc.d)


## DRAW 4 SCATTERPLOTS
par(mfrow=c(2,2))
lapply(lres, scatter)
```

```
# detach data
detach(dapcIllus)
}
```

`df2genind` *Convert a data.frame of genotypes to a genind object, and conversely.*

Description

The function `df2genind` converts a data.frame (or a matrix) into a genind object. The data.frame must meet the following requirements:
- genotypes are in row (one row per genotype)
- markers are in columns
- each element is a string of characters coding alleles with or without separator. If no separator is used, the function tries to find how many characters code each genotypes at a locus, but it is safer to state it (`ncode` argument). Uncomplete strings are filled with "0" at the begining.

The function `genind2df` converts a genind back to such a data.frame. Alleles of a given locus can be coded as a single character string (with specified separators), or provided on different columns (see `oneColPerAll` argument).

Usage

```
df2genind(X, sep=NULL, ncode=NULL, ind.names=NULL, loc.names=NULL,
 pop=NULL, missing=NA, ploidy=2, type=c("codom","PA"))
genind2df(x,pop=NULL, sep="", usepop=TRUE, oneColPerAll=FALSE)
```

Arguments

`X`	a matrix or a data.frame (see decription)
`sep`	a character string separating alleles. See details.
`ncode`	an optional integer giving the number of characters used for coding one genotype at one locus. If not provided, this is determined from data.
`ind.names`	an optional character vector giving the individuals names; if NULL, taken from rownames of X.
`loc.names`	an optional character vector giving the markers names; if NULL, taken from colnames of X.
`pop`	an optional factor giving the population of each individual.
`missing`	can be NA, 0 or "mean". See details section.
`ploidy`	an integer indicating the degree of ploidy of the genotypes.
`type`	a character string indicating the type of marker: 'codom' stands for 'codominant' (e.g. microstallites, allozymes); 'PA' stands for 'presence/absence' markers (e.g. AFLP, RAPD).
`x`	a genind object
`usepop`	a logical stating whether the population (argument `pop` or `x@pop` should be used (TRUE, default) or not (FALSE)).
`oneColPerAll`	a logical stating whether alleles of one locus should be provided on separate columns (TRUE) rather than as a single character string (FALSE, default).

Details

=== There are 3 treatments for missing values ===
- NA: kept as NA.

- 0: allelic frequencies are set to 0 on all alleles of the concerned locus. Recommended for a PCA on compositionnal data.

- "mean": missing values are replaced by the mean frequency of the corresponding allele, computed on the whole set of individuals. Recommended for a centred PCA.

=== Details for the `sep` argument ===
this character is directly used in reguar expressions like `gsub`, and thus require some characters to be preceeded by double backslashes. For instance, "/" works but "|" must be coded as "\\|".

Value

an object of the class genind for `df2genind`; a matrix of biallelic genotypes for `genind2df`

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

See Also

`import2genind`, `read.genetix`, `read.fstat`, `read.structure`

Examples

```
## simple example
df <- data.frame(locusA=c("11","11","12","32"),
locusB=c(NA,"34","55","15"),locusC=c("22","22","21","22"))
row.names(df) <- .genlab("genotype",4)
df

obj <- df2genind(df, ploidy=2)
obj
truenames(obj)

## converting a genind as data.frame
genind2df(obj)
genind2df(obj, sep="/")
genind2df(obj, oneColPerAll=TRUE)
```

`dist.genpop` *Genetic distances between populations*

Description

This function computes measures of genetic distances between populations using a `genpop` object. Currently, five distances are available, some of which are euclidian (see details).

A non-euclidian distance can be transformed into an Euclidian one using `cailliez` in order to perform a Principal Coordinate Analysis `dudi.pco` (both functions in `ade4`).

The function `dist.genpop` is based on former `dist.genet` function of `ade4` package.

Usage

```
dist.genpop(x, method = 1, diag = FALSE, upper = FALSE)
```

Arguments

`x`	a list of class `genpop`
`method`	an integer between 1 and 5. See details
`diag`	a logical value indicating whether the diagonal of the distance matrix should be printed by `print.dist`
`upper`	a logical value indicating whether the upper triangle of the distance matrix should be printed by `print.dist`

Details

Let **A** a table containing allelic frequencies with *t* populations (rows) and *m* alleles (columns).
Let ν the number of loci. The locus *j* gets *m(j)* alleles. $m = \sum_{j=1}^{\nu} m(j)$

For the row *i* and the modality *k* of the variable *j*, notice the value a_{ij}^k ($1 \leq i \leq t$, $1 \leq j \leq \nu$, $1 \leq k \leq m(j)$) the value of the initial table.

$a_{ij}^+ = \sum_{k=1}^{m(j)} a_{ij}^k$ and $p_{ij}^k = \frac{a_{ij}^k}{a_{ij}^+}$

Let **P** the table of general term p_{ij}^k
$p_{ij}^+ = \sum_{k=1}^{m(j)} p_{ij}^k = 1$, $p_{i+}^+ = \sum_{j=1}^{\nu} p_{ij}^+ = \nu$, $p_{++}^+ = \sum_{j=1}^{\nu} p_{i+}^+ = t\nu$

The option `method` computes the distance matrices between populations using the frequencies p_{ij}^k.

1. Nei's distance (not Euclidian):
$$D_1(a,b) = -\ln\left(\frac{\sum_{k=1}^{\nu}\sum_{j=1}^{m(k)} p_{aj}^k p_{bj}^k}{\sqrt{\sum_{k=1}^{\nu}\sum_{j=1}^{m(k)} (p_{aj}^k)^2}\sqrt{\sum_{k=1}^{\nu}\sum_{j=1}^{m(k)} (p_{bj}^k)^2}}\right)$$

2. Angular distance or Edwards' distance (Euclidian):
$$D_2(a,b) = \sqrt{1 - \frac{1}{\nu}\sum_{k=1}^{\nu}\sum_{j=1}^{m(k)} \sqrt{p_{aj}^k p_{bj}^k}}$$

3. Coancestrality coefficient or Reynolds' distance (Euclidian):
$$D_3(a,b) = \sqrt{\frac{\sum_{k=1}^{\nu}\sum_{j=1}^{m(k)} (p_{aj}^k - p_{bj}^k)^2}{2\sum_{k=1}^{\nu}(1 - \sum_{j=1}^{m(k)} p_{aj}^k p_{bj}^k)}}$$

4. Classical Euclidean distance or Rogers' distance (Euclidian):
$D_4(a,b) = \frac{1}{\nu}\sum_{k=1}^{\nu}\sqrt{\frac{1}{2}\sum_{j=1}^{m(k)}\left(p_{aj}^k - p_{bj}^k\right)^2}$

5. Absolute genetics distance or Provesti 's distance (not Euclidian):
$D_5(a,b) = \frac{1}{2\nu}\sum_{k=1}^{\nu}\sum_{j=1}^{m(k)}\left|p_{aj}^k - p_{bj}^k\right|$

Value

returns a distance matrix of class `dist` between the rows of the data frame

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>
Former dist.genet code by Daniel Chessel <chessel@biomserv.univ-lyon1.fr>
and documentation by Anne B. Dufour <dufour@biomserv.univ-lyon1.fr>

References

To complete informations about distances:

Distance 1:
Nei, M. (1972) Genetic distances between populations. *American Naturalist*, **106**, 283–292.
Nei M. (1978) Estimation of average heterozygosity and genetic distance from a small number of individuals. *Genetics*, **23**, 341–369.
Avise, J. C. (1994) Molecular markers, natural history and evolution. Chapman & Hall, London.

Distance 2:
Edwards, A.W.F. (1971) Distance between populations on the basis of gene frequencies. *Biometrics*, **27**, 873–881.
Cavalli-Sforza L.L. and Edwards A.W.F. (1967) Phylogenetic analysis: models and estimation procedures. *Evolution*, **32**, 550–570.
Hartl, D.L. and Clark, A.G. (1989) Principles of population genetics. Sinauer Associates, Sunderland, Massachussetts (p. 303).

Distance 3:
Reynolds, J. B., B. S. Weir, and C. C. Cockerham. (1983) Estimation of the coancestry coefficient: basis for a short-term genetic distance. *Genetics*, **105**, 767–779.

Distance 4:
Rogers, J.S. (1972) Measures of genetic similarity and genetic distances. *Studies in Genetics*, Univ. Texas Publ., **7213**, 145–153.
Avise, J. C. (1994) Molecular markers, natural history and evolution. Chapman & Hall, London.

Distance 5:
Prevosti A. (1974) La distancia genetica entre poblaciones. *Miscellanea Alcobe*, **68**, 109–118.
Prevosti A., Oca\~na J. and Alonso G. (1975) Distances between populations of Drosophila subobscura, based on chromosome arrangements frequencies. *Theoretical and Applied Genetics*, **45**, 231–241.

For more information on dissimilarity indexes:
Gower J. and Legendre P. (1986) Metric and Euclidian properties of dissimilarity coefficients. *Journal of Classification*, **3**, 5–48

Legendre P. and Legendre L. (1998) *Numerical Ecology*, Elsevier Science B.V. 20, pp274–288.

See Also

cailliez,dudi.pco

Examples

```
if(require(ade4)){
data(microsatt)
obj <- as.genpop(microsatt$tab)

listDist <- lapply(1:5, function(i) cailliez(dist.genpop(obj,met=i)))
for(i in 1:5) {attr(listDist[[i]],"Labels") <- obj@pop.names}
listPco <- lapply(listDist, dudi.pco,scannf=FALSE)

par(mfrow=c(2,3))
for(i in 1:5) {scatter(listPco[[i]],sub=paste("Dist:", i))}
}
```

---

eHGDP *Extended HGDP-CEPH dataset*

Description

This dataset consists of 1350 individuals from native Human populations distributed worldwide typed at 678 microsatellite loci. The original HGDP-CEPH panel [1-3] has been extended by several native American populations [4]. This dataset was used to illustrate the Discriminant Analysis of Principal Components (DAPC, [5]).

Usage

```
data(eHGDP)
```

Format

eHGDP is a genind object with a data frame named popInfo as supplementary component (eHGDP@other$popInfo which contains the following variables:

Population: a character vector indicating populations.

Region: a character vector indicating the geographic region of each population.

Label: a character vector indicating the correspondance with population labels used in the genind object (i.e., as output by pop(eHGDP)).

Latitude,Longitude: geographic coordinates of the populations, indicated as north and east degrees.

Source

Original panel by Human Genome Diversity Project (HGDP) and Centre d'Etude du Polymorphisme Humain (CEPH). See reference [4] for Native American populations.

This copy of the dataset was prepared by Francois Balloux (f.balloux@imperial.ac.uk).

References

[1] Rosenberg NA, Pritchard JK, Weber JL, Cann HM, Kidd KK, et al. (2002) Genetic structure of human populations. *Science* 298: 2381-2385.

[2] Ramachandran S, Deshpande O, Roseman CC, Rosenberg NA, Feldman MW, et al. (2005) Support from the relationship of genetic and geographic distance in human populations for a serial founder effect originating in Africa. *Proc Natl Acad Sci U S A* 102: 15942-15947.

[3] Cann HM, de Toma C, Cazes L, Legrand MF, Morel V, et al. (2002) A human genome diversity cell line panel. *Science* 296: 261-262.

[4] Wang S, Lewis CM, Jakobsson M, Ramachandran S, Ray N, et al. (2007) Genetic Variation and Population Structure in Native Americans. *PLoS Genetics* 3: e185.

[5] Jombart, T., Devillard, S. and Balloux, F. Discriminant analysis of principal components: a new method for the analysis of genetically structured populations. Submitted to *BMC genetics*.

Examples

```
## Not run:
## LOAD DATA
data(eHGDP)
eHGDP


## PERFORM DAPC - USE POPULATIONS AS CLUSTERS
## to reproduce exactly analyses from the paper, use "n.pca=1000"
dapc1 <- dapc(eHGDP, all.contrib=TRUE, scale=FALSE, n.pca=200, n.da=80) # takes 2 minutes
dapc1

## (see ?dapc for details about the output)



## SCREEPLOT OF EIGENVALUES
barplot(dapc1$eig, main="eHGDP - DAPC eigenvalues", col=c("red","green","blue", rep("grey



## SCATTERPLOTS
## (!) Note: colors may be inverted with respect to [5]
## as signs of principal components are arbitrary
## and change from one computer to another
##
## axes 1-2
s.label(dapc1$grp.coord[,1:2], clab=0, sub="Axes 1-2")
par(xpd=T)
colorplot(dapc1$grp.coord[,1:2], dapc1$grp.coord, cex=3, add=TRUE)
add.scatter.eig(dapc1$eig,10,1,2, posi="bottomright", ratio=.3, csub=1.25)

## axes 2-3
s.label(dapc1$grp.coord[,2:3], clab=0, sub="Axes 2-3")
par(xpd=T)
colorplot(dapc1$grp.coord[,2:3], dapc1$grp.coord, cex=3, add=TRUE)
add.scatter.eig(dapc1$eig,10,1,2, posi="bottomright", ratio=.3, csub=1.25)
```

```
## MAP DAPC1 RESULTS
if(require(maps)){

xy <- cbind(eHGDP$other$popInfo$Longitude, eHGDP$other$popInfo$Latitude)

par(mar=rep(.1,4))
map(fill=TRUE, col="lightgrey")
colorplot(xy, -dapc1$grp.coord, cex=3, add=TRUE, trans=FALSE)
}



## LOOK FOR OTHER CLUSTERS
## to reproduce results of the reference paper, use :
## grp <- find.clusters(hgdp, max.n=50, n.pca=200, scale=FALSE)
## and then
## plot(grp$Kstat, type="b", col="blue")

grp <- find.clusters(eHGDP, max.n=30, n.pca=200, scale=FALSE, n.clust=4) # takes about 2
names(grp)

## (see ?find.clusters for details about the output)



## PERFORM DAPC - USE POPULATIONS AS CLUSTERS
## to reproduce exactly analyses from the paper, use "n.pca=1000"
dapc2 <- dapc(eHGDP, pop=grp$grp, all.contrib=TRUE, scale=FALSE, n.pca=200, n.da=80) # ta
dapc2


## PRODUCE SCATTERPLOT
scatter(dapc2) # axes 1-2
scatter(dapc2,2,3) # axes 2-3


## MAP DAPC2 RESULTS
if(require(maps)){
xy <- cbind(eHGDP$other$popInfo$Longitude, eHGDP$other$popInfo$Latitude)

myCoords <- apply(dapc2$ind.coord, 2, tapply, pop(eHGDP), mean)

par(mar=rep(.1,4))
map(fill=TRUE, col="lightgrey")
colorplot(xy, myCoords, cex=3, add=TRUE, trans=FALSE)
}


## End(Not run)
```

`export` *Conversion functions from adegenet to other R packages*

Description

The function genind2genotype and genind2hierfstat convert a genind object into, respectively, a list of genotypes (class genotypes, package genetics), and a data.frame to be used by the functions of the package hierfstat.

Usage

```
genind2genotype(x,pop=NULL,res.type=c("matrix","list"))
genind2hierfstat(x,pop=NULL)
```

Arguments

x	a genind object.
pop	a factor giving the population of each individual. If NULL, it is seeked in x\$pop. If NULL again, all individuals are assumed from the same population.
res.type	a character (if a vector, only the first element is retained), indicating the type of result returned.

Value

The function genind2genotype converts a genind object into genotypes (package genetics). If res.type is set to "matrix" (default), the returned value is a individuals x locus matrix whose columns have the class genotype. Such data can be used by LDheatmap package to compute linkage disequilibrium.

If res.type is set to "list", the returned value is a list of genotypes sorted first by locus and then by population.)

genind2hierfstat returns a data frame where individuals are in rows. The first columns is a population factor (but stored as integer); each other column is a locus. Genotypes are coded as integers (e.g., 44 is an homozygote 4/4, 56 is an heterozygote 5/6).

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

References

Gregory Warnes and Friedrich Leisch (2007). genetics: Population Genetics. R package version 1.2.1.

Jerome Goudet (2005). HIERFSTAT, a package for R to compute and test hierarchical F-statistics. *Molecular Ecology*, **5**:184-186

Fstat (version 2.9.3). Software by Jerome Goudet. http://www2.unil.ch/popgen/softwares/fstat.htm

See Also

import2genind

F statistics *F statistics for genind objects*

Description

The function `fstat` computes a global Fst, while `pairwise.fst` computes Nei's pairwise Fst between all pairs of populations. Both functions are designed for genind objects.

`fstat` is wrapper for `varcomp.glob` from package `hierfstat` for genind objects. It computes F statistics (Fst, Fis, Fit) given a set of genotypes and a grouping factor.

`pairwise.fst` is an implementation of Nei's Fst in which heretozygosities are weighted by group sizes (see details).

Usage

```
fstat(x, pop=NULL, fstonly=FALSE)

pairwise.fst(x, pop=NULL, res.type=c("dist","matrix"), truenames=TRUE)
```

Arguments

`x`	an object of class genind.
`pop`	a factor giving the 'population' of each individual. If NULL, pop is seeked from `pop(x)`. Note that the term population refers in fact to any grouping of individuals'.
`fstonly`	a logical stating whether only the Fst value should be returned (TRUE) instead of all F statistics (FALSE, default).
`res.type`	the type of result to be returned: a `dist` object, or a symmetric matrix
`truenames`	a logical indicating whether true labels (as opposed to generic labels) should be used to name the output.

Details

Let A and B be two populations of population sizes n_A and n_B, with expected heterozygosity (averaged over loci) $Hs(A)$ and $Hs(B)$, respectively. We denote Ht the expected heterozygosity of a population pooling A and B. Then, the pairwise Fst between A and B is computed as:

$$Fst(A,B) = \frac{(Ht-(n_A Hs(A)+n_B Hs(B))/(n_A+n_B))}{Ht}$$

Value

A vector, a matrix, or a dist object containing F statistics.

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

References

Nei, M. (1973) Analysis of gene diversity in subdivided populations. Proc Natl Acad Sci USA, 70: 3321-3323

See Also

Hs

---

fasta2genlight *Extract Single Nucleotide Polymorphism (SNPs) from alignments*

Description

The function fasta2genlight reads alignments with the fasta format (extensions ".fasta", ".fas", or ".fa"), extracts the binary SNPs, and converts the output into a genlight object.

The function reads data by chunks of a few genomes (minimum 1, no maximum) at a time, which allows one to read massive datasets with negligible RAM requirements (albeit at a cost of computational time). The argument chunkSize indicates the number of genomes read at a time. Increasing this value decreases the computational time required to read data in, while increasing memory requirements.

Multiple cores can be used to decrease the overall computational time on multicore architectures (needs the package multicore).

Usage

```
fasta2genlight(file, quiet=FALSE, chunkSize = 1000, saveNbAlleles=FALSE,
               multicore = require("multicore"), n.cores = NULL, ...)
```

Arguments

file	a character string giving the path to the file to convert, with the extension ".snp".
quiet	logical stating whether a conversion messages should be printed (TRUE,default) or not (FALSE).
chunkSize	an integer indicating the number of genomes to be read at a time; larger values require more RAM but decrease the time needed to read the data.
saveNbAlleles	a logical indicating whether the number of alleles for each loci in the original alignment should be saved in the other slot (TRUE), or not (FALSE, default). In large genomes, this takes some space but allows for tracking SNPs with more than 2 alleles, lost during the conversion.
multicore	a logical indicating whether multiple cores -if available- should be used for the computations (TRUE, default), or not (FALSE); requires the package multicore to be installed (see details).
n.cores	if multicore is TRUE, the number of cores to be used in the computations; if NULL, then the maximum number of cores available on the computer is used.
...	other arguments to be passed to other functions - currently not used.

Details

=== Using multiple cores ===

Most recent machines have one or several processors with multiple cores. R processes usually use one single core. The package `multicore` allows for parallelizing some computations on multiple cores, which decreases drastically computational time.

To use this functionality, you need to have the last version of the `multicore` package installed. To install it, type: install.packages("multicore",,"http://rforge.net/",type="source")

DO NOT use the version on CRAN, which is slightly outdated.

Value

an object of the class genlight

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

See Also

- ?genlight for a description of the class genlight.

- read.snp: read SNPs in adegenet's '.snp' format.

- read.PLINK: read SNPs in PLINK's '.raw' format.

- df2genind: convert any multiallelic markers into adegenet genind.

- import2genind: read multiallelic markers from various software into adegenet.

Examples

```
## show the example file ##
## this is the path to the file:
myPath <- system.file("files/usflu.fasta",package="adegenet")
myPath

## read the file
obj <- fasta2genlight(myPath, chunk=10) # process 10 sequences at a time
obj

## look at extracted information
position(obj)
alleles(obj)
locNames(obj)

## plot positions of polymorphic sites
temp <- density(position(obj), bw=10)
plot(temp, xlab="Position in the alignment", lwd=2, main="Location of the SNPs")
points(position(obj), rep(0, nLoc(obj)), pch="|", col="red")
```

`find.clusters` *find.cluster: cluster identification using successive K-means*

Description

These functions implement the clustering procedure used in Discriminant Analysis of Principal Components (DAPC, Jombart et al. 2010). This procedure consists in running successive K-means with an increasing number of clusters (`k`), after transforming data using a principal component analysis (PCA). For each model, a statistical measure of goodness of fit (by default, BIC) is computed, which allows to choose the optimal `k`. See `details` for a description of how to select the optimal `k` and `vignette("adegenet-dapc")` for a tutorial.

Optionally, hierarchical clustering can be sought by providing a prior clustering of individuals (argument `clust`). In such case, clusters will be sought within each prior group.

The K-means procedure used in `find.clusters` is `kmeans` function from the `stats` package. The PCA function is `dudi.pca` from the `ade4` package, except for genlight objects which use the `glPca` procedure from adegenet.

`find.clusters` is a generic function with methods for the following types of objects:
- `data.frame` (only numeric data)
- `matrix` (only numeric data)
- `genind` objects (genetic markers)
- `genlight` objects (genome-wide SNPs)

Usage

```
## S3 method for class 'data.frame'
find.clusters(x, clust=NULL, n.pca=NULL,
              n.clust=NULL, stat=c("BIC","AIC", "WSS"),
              choose.n.clust=TRUE,criterion=c("diffNgroup",
              "min","goesup", "smoothNgoesup", "goodfit"),
              max.n.clust=round(nrow(x)/10), n.iter=1e5, n.start=10,
              center=TRUE, scale=TRUE, pca.select=c("nbEig","percVar"),
              perc.pca=NULL, ..., dudi=NULL)

## S3 method for class 'matrix'
find.clusters(x, ...)

## S3 method for class 'genind'
find.clusters(x, clust=NULL, n.pca=NULL, n.clust=NULL,
              stat=c("BIC","AIC", "WSS"), choose.n.clust=TRUE,
              criterion=c("diffNgroup", "min","goesup", "smoothNgoesup",
              "goodfit"), max.n.clust=round(nrow(x@tab)/10), n.iter=1e5,
              n.start=10, scale=FALSE, scale.method=c("sigma", "binom"),
              truenames=TRUE, ...)

## S3 method for class 'genlight'
find.clusters(x, clust=NULL, n.pca=NULL,
              n.clust=NULL, stat=c("BIC", "AIC",
              "WSS"),choose.n.clust=TRUE, criterion=c("diffNgroup",
              "min","goesup","smoothNgoesup", "goodfit"),
              max.n.clust=round(nInd(x)/10), n.iter=1e5,n.start=10,
```

```
scale=FALSE, pca.select=c("nbEig","percVar"),
perc.pca=NULL,glPca=NULL, ...)
```

Arguments

x	a data.frame, matrix, or genind object. For the data.frame and matrix arguments, only quantitative variables should be provided.
clust	an optional factor indicating a prior group membership of individuals. If provided, sub-clusters will be sought within each prior group.
n.pca	an integer indicating the number of axes retained in the Principal Component Analysis (PCA) step. If NULL, interactive selection is triggered.
n.clust	an optinal integer indicating the number of clusters to be sought. If provided, the function will only run K-means once, for this number of clusters. If left as NULL, several K-means are run for a range of k (number of clusters) values.
stat	a character string matching 'BIC', 'AIC', or 'WSS', which indicates the statistic to be computed for each model (i.e., for each value of k). BIC: Bayesian Information Criterion. AIC: Aikaike's Information Criterion. WSS: within-groups sum of squares, that is, residual variance.
choose.n.clust	a logical indicating whether the number of clusters should be chosen by the user (TRUE, default), or automatically, based on a given criterion (argument criterion). It is HIGHLY RECOMMENDED to choose the number of clusters INTERACTIVELY, since i) the decrease of the summary statistics (BIC by default) is informative, and ii) no criteria for automatic selection is appropriate to all cases (see details).
criterion	a character string matching "diffNgroup", "min","goesup", "smoothNgoesup", or "conserv", indicating the criterion for automatic selection of the optimal number of clusters. See details for an explanation of these procedures.
max.n.clust	an integer indicating the maximum number of clusters to be tried. Values of 'k' will be picked up between 1 and max.n.clust
n.iter	an integer indicating the number of iterations to be used in each run of K-means algorithm. Corresponds to iter.max of kmeans function.
n.start	an integer indicating the number of randomly chosen starting centroids to be used in each run of the K-means algorithm. Using more starting points ensures convergence of the algorithm. Corresponds to nstart of kmeans function.
center	a logical indicating whether variables should be centred to mean 0 (TRUE, default) or not (FALSE). Always TRUE for genind objects.
scale	a logical indicating whether variables should be scaled (TRUE) or not (FALSE, default). Scaling consists in dividing variables by their (estimated) standard deviation to account for trivial differences in variances. In allele frequencies, it comes with the risk of giving uninformative alleles more importance while downweighting informative alleles. Further scaling options are available for genind objects (see argument scale.method).
pca.select	a character indicating the mode of selection of PCA axes, matching either "nbEig" or "percVar". For "nbEig", the user has to specify the number of axes retained (interactively, or via n.pca). For "percVar", the user has to specify the minimum amount of the total variance to be preserved by the retained axes, expressed as a percentage (interactively, or via perc.pca).

perc.pca	a numeric value between 0 and 100 indicating the minimal percentage of the total variance of the data to be expressed by the retained axes of PCA.
scale.method	a character specifying the scaling method to be used for allele frequencies, which must match "sigma" (usual estimate of standard deviation) or "binom" (based on binomial distribution). See scaleGen for further details.
truenames	a logical indicating whether true (i.e., user-specified) labels should be used in object outputs (TRUE, default) or not (FALSE), in which case generic labels are used.
...	further arguments to be passed to other functions. For find.clusters.matrix, arguments are to match those of the data.frame method.
dudi	optionally, a multivariate analysis with the class dudi (from the ade4 package). If provided, prior PCA will be ignored, and this object will be used as a prior step for variable orthogonalisation.
glPca	an optional glPca object; if provided, dimension reduction is not performed (saving computational time) but taken directly from this object.

Details

=== ON THE SELECTION OF K ===
(where K is the 'optimal' number of clusters)

So far, the analysis of data simulated under various population genetics models (see reference) suggested an ad hoc rule for the selection of the optimal number of clusters. First important result is that BIC seems for efficient than AIC and WSS to select the appropriate number of clusters (see example). The rule of thumb consists in increasing K until it no longer leads to an appreciable improvement of fit (i.e., to a decrease of BIC). In the most simple models (island models), BIC decreases until it reaches the optimal K, and then increases. In these cases, our rule amounts to choosing the lowest K. In other models such as stepping stones, the decrease of BIC often continues after the optimal K, but is much less steep.

An alternative approach is the automatic selection based on a fixed criterion. Note that, in any case, it is highly recommended to look at the graph of the BIC for different numbers of clusters as displayed during the interactive cluster selection. To use automated selection, set choose.n.clust to FALSE and specify the criterion you want to use, from the following values:

- "diffNgroup": differences between successive values of the summary statistics (by default, BIC) are splitted into two groups using a Ward's clustering method (see ?hclust), to differentiate sharp decrease from mild decreases or increases. The retained K is the one before the first group switch. Appears to work well for island/hierarchical models, and decently for isolation by distance models, albeit with some unstability. Can be impacted by an initial, very sharp decrease of the test statistics. IF UNSURE ABOUT THE CRITERION TO USE, USE THIS ONE.

- "min": the model with the minimum summary statistics (as specified by stat argument, BIC by default) is retained. Is likely to work for simple island model, using BIC. It is likely to fail in models relating to stepping stones, where the BIC always decreases (albeit by a small amount) as K increases. In general, this approach tends to over-estimate the number of clusters.

- "goesup": the selected model is the K after which increasing the number of clusters leads to increasing the summary statistics. Suffers from inaccuracy, since i) a steep decrease might follow a small 'bump' of increase of the statistics, and ii) increase might never happen, or happen after negligible decreases. Is likely to work only for clear-cut island models.

- "smoothNgoesup": a variant of "goesup", in which the summary statistics is first smoothed using a lowess approach. Is meant to be more accurate than "goesup" as it is less prone to stopping to small 'bumps' in the decrease of the statistics.

- "goodfit": another criterion seeking a good fit with a minimum number of clusters. This approach does not rely on differences between successive statistics, but on absolute fit. It selects the model with the smallest K so that the overall fit is above a given threshold.

Value

The class `find.clusters` is a list with the following components:

`Kstat`	a `numeric` vector giving the values of the summary statistics for the different values of K. Is NULL if `n.clust` was specified.
`stat`	a `numeric` value giving the value of the summary statistics for the retained model
`grp`	a `factor` giving group membership for each individual.
`size`	an `integer` vector giving the size of the different clusters.

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

References

Jombart T, Devillard S and Balloux F (2010) Discriminant analysis of principal components: a new method for the analysis of genetically structured populations. BMC Genetics 11:94. doi:10.1186/1471-2156-11-94

See Also

- dapc: implements the DAPC.
- scatter.dapc: graphics for DAPC.
- dapcIllus: dataset illustrating the DAPC and `find.clusters`.
- eHGDP: dataset illustrating the DAPC and `find.clusters`.
- kmeans: implementation of K-means in the stat package.
- dudi.pca: implementation of PCA in the ade4 package.

Examples

```
## Not run:
## THIS ONE TAKES A FEW MINUTES TO RUN ##
data(eHGDP)

## here, n.clust is specified, so that only on K value is used
grp <- find.clusters(eHGDP, max.n=30, n.pca=200, scale=FALSE, n.clust=4) # takes about 2
names(grp)
grp$Kstat
grp$stat


## to try different values of k (interactive)
grp <- find.clusters(hgdp, max.n=50, n.pca=200, scale=FALSE)

## and then, to plot BIC values:
plot(grp$Kstat, type="b", col="blue")
```

```
## End(Not run)


## ANOTHER SIMPLE EXAMPLE ##
data(sim2pop) # this actually contains 2 pop

## DETECTION WITH BIC (clear result)
foo.BIC <- find.clusters(sim2pop, n.pca=100, choose=FALSE)
plot(foo.BIC$Kstat, type="o", xlab="number of clusters (K)", ylab="BIC",
col="blue", main="Detection based on BIC")
points(2, foo.BIC$Kstat[2], pch="x", cex=3)
mtext(3, tex="'X' indicates the actual number of clusters")


## DETECTION WITH AIC (less clear-cut)
foo.AIC <- find.clusters(sim2pop, n.pca=100, choose=FALSE, stat="AIC")
plot(foo.AIC$Kstat, type="o", xlab="number of clusters (K)", ylab="AIC", col="purple", ma
points(2, foo.AIC$Kstat[2], pch="x", cex=3)
mtext(3, tex="'X' indicates the actual number of clusters")


## DETECTION WITH WSS (less clear-cut)
foo.WSS <- find.clusters(sim2pop, n.pca=100, choose=FALSE, stat="WSS")
plot(foo.WSS$Kstat, type="o", xlab="number of clusters (K)", ylab="WSS
(residual variance)", col="red", main="Detection based on WSS")
points(2, foo.WSS$Kstat[2], pch="x", cex=3)
mtext(3, tex="'X' indicates the actual number of clusters")


## TOY EXAMPLE FOR GENLIGHT OBJECTS ##
x <- glSim(100,500,500)
x
plot(x)
grp <- find.clusters(x, n.pca=100, choose=FALSE, stat="BIC")
plot(grp$Kstat, type="o", xlab="number of clusters (K)",ylab="BIC",main="find.clusters on
```

`genind class` *adegenet formal class (S4) for individual genotypes*

Description

The S4 class `genind` is used to store individual genotypes.
It contains several components described in the 'slots' section).
The `summary` of a `genind` object invisibly returns a list of component. The function `.valid.genind` is for internal use. The function `genind` creates a genind object from a valid table of alleles corresponding to the `@tab` slot. Note that as in other S4 classes, slots are accessed using @ instead of \$.

Slots

`tab`**:** matrix of genotypes (in rows) for all alleles (in columns). The table differs depending on the `@type` slot:

- 'codom': values are frequencies ; '0' if the genotype does not have the corresponding allele, '1' for an homozygote and 0.5 for an heterozygte.
- 'PA': values are presence/absence of alleles.
In all cases, rows and columns are given generic names.

`loc.names`: character vector containing the real names of the loci

`loc.fac`: locus factor for the columns of `tab`

`loc.nall`: integer vector giving the number of alleles per locus

`all.names`: list having one component per locus, each containing a character vector of alleles names

`call`: the matched call

`ind.names`: character vector containing the real names of the individuals. Note that as Fstat does not store these names, objects converted from .dat files will contain empty `ind.names`.

`ploidy`: an integer indicating the degree of ploidy of the genotypes. Beware: 2 is not an integer, but as.integer(2) is.

`type`: a character string indicating the type of marker: 'codom' stands for 'codominant' (e.g. microstallites, allozymes); 'PA' stands for 'presence/absence' (e.g. AFLP).

`pop`: (optional) factor giving the population of each individual

`pop.names`: (optional) vector giving the real names of the populations

`other`: (optional) a list containing other information

Extends

Class "`gen`", directly. Class "`indInfo`", directly.

Methods

names `signature(x = "genind")`: give the names of the components of a genind object

print `signature(x = "genind")`: prints a genind object

show `signature(object = "genind")`: shows a genind object (same as print)

summary `signature(object = "genind")`: summarizes a genind object, invisibly returning its content

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

See Also

`as.genind`, `is.genind`, `genind2genpop`, `genpop`, `import2genind`, `read.genetix`, `read.genepop`, `read.fstat`, `na.replace`

Related classes:
- genpop for storing data per populations

- genlight for an efficient storage of binary SNPs genotypes

Examples

```
showClass("genind")

obj <- read.genetix(system.file("files/nancycats.gtx",package="adegenet"),missing="mean")
obj
validObject(obj)
summary(obj)

# test inter-colonies structuration
if(require(hierfstat)){
gtest <- gstat.randtest(obj,nsim=99)
gtest
plot(gtest)
}

# perform an inter-class PCA
if(require(ade4)){
pca1 <- dudi.pca(obj@tab,scannf=FALSE,scale=FALSE)
pcabet1 <- between(pca1,obj@pop,scannf=FALSE)
pcabet1

s.class(pcabet1$ls,obj@pop,sub="Inter-class PCA",possub="topleft",csub=2)
add.scatter.eig(pcabet1$eig,2,xax=1,yax=2)
}
```

---

genind constructor *genind constructor*

Description

Constructor for genind objects.
The function `genind` creates a genind object from a matrix of allelic frequency where genotypes are in rows and alleles in columns. This table must have correct names for rows and columns.

The function `as.genind` is an alias for `genind` function.

`is.genind` tests if an object is a valid genind object.

Note: to get the manpage about genind, please type 'class ? genind'.

Usage

```
genind(tab,pop=NULL,prevcall=NULL, ploidy=2, type=c("codom","PA"))
as.genind(tab,pop=NULL,prevcall=NULL, ploidy=2, type=c("codom","PA"))
is.genind(x)
```

Arguments

`tab`	A table corresponding to the @tab slot of a genind object, with individuals in rows and alleles in columns. Its content depends on `type` (type of marker). - 'codom': table contains allele frequencies (numeric values summing to 1).

- 'PA': table contains binary values, which indicate presence(1)/absence(0) of alleles.

pop	a factor giving the population of each genotype in 'x'
prevcall	call of an object
ploidy	an integer indicating the degree of ploidy of the genotypes. Beware: 2 is not an integer, but as.integer(2) is.
type	a character string indicating the type of marker: 'codom' stands for 'codominant' (e.g. microstallites, allozymes); 'PA' stands for 'presence/absence' (e.g. AFLP).
x	an object

Value

For genind and as.genind, a genind object. For is.genind, a logical.

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

See Also

genind class, and import2genind for importing from various types of file.

Related classes:
- genpop for storing data per populations

- genlight for an efficient storage of binary SNPs genotypes

Examples

```
data(nancycats)
nancycats@loc.names

# isolate one marker, fca23
obj <- seploc(nancycats)$"fca23"
obj
```

---

genind2genpop *Conversion from a genind to a genpop object*

Description

The function genind2genpop converts genotypes data (genind) into alleles counts per population (genpop).

Usage

```
genind2genpop(x,pop=NULL,missing=c("NA","0","chi2"),quiet=FALSE,
              process.other=FALSE, other.action=mean)
```

Arguments

x	an object of class genind.
pop	a factor giving the population of each genotype in 'x'. If note provided, seeked in x@pop, but if given, the argument prevails on x@pop.
missing	can be "NA", "0", or "chi2". See details for more information.
quiet	logical stating whether a conversion message must be printed (TRUE,default) or not (FALSE).
process.other	a logical indicating whether the @other slot should be processed (see details).
other.action	a function to be used when processing the @other slot. By default, 'mean' is used.

Details

=== 'missing' argument ===
The values of the 'missing' argument in genind2genpop have the following effects:
- "NA": if all genotypes of a population for a given allele are missing, count value will be NA

- "0": if all genotypes of a population for a given allele are missing, count value will be 0

- "chi2": if all genotypes of a population for a given allele are missing, count value will be that of a theoretical count in of a Chi-squared test. This is obtained by the product of the margins sums divided by the total number of alleles.

=== processing the @other slot ===
Essentially, genind2genpop is about aggregating data per population. The function can do the same for all numeric items in the @other slot provided they have the same length (for vectors) or the same number of rows (matrix-like objects) as the number of genotypes. When the case is encountered and if process.other is TRUE, then these objects are processed using the function defined in other.action per population. For instance, spatial coordinates of genotypes would be averaged to obtain population coordinates.

Value

A genpop object. The component @other in 'x' is passed to the created genpop object.

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

See Also

genind, genpop, na.replace

Examples

```
## simple conversion
  data(nancycats)
  nancycats
  catpop <- genind2genpop(nancycats)
  catpop
```

```
  summary(catpop)

## processing the @other slot
data(sim2pop)
sim2pop$other$foo <- letters
sim2pop
dim(sim2pop$other$xy) # matches the number of genotypes
sim2pop$other$foo # does not match the number of genotypes

obj <- genind2genpop(sim2pop, process.other=TRUE)
obj$other # the new xy is the populations' centre

pch <- as.numeric(pop(sim2pop))
col <- pop(sim2pop)
levels(col) <- c("blue","red")
col <- as.character(col)
plot(sim2pop$other$xy, pch=pch, col=col)
text(obj$other$xy, lab=row.names(obj$other$xy), col=c("blue","red"), cex=2, font=2)
```

`genlight auxiliary functions`

Auxiliary functions for genlight objects

Description

These functions provide facilities for usual computations using genlight objects. When ploidy varies across individuals, the outputs of these functions depend on whether the information units are individuals, or alleles within individuals (see details).

These functions are:

- `glSum`: computes the sum of the number of second allele in each SNP.

- `glNA`: computes the number of missing values in each SNP.

- `glMean`: computes the mean number of second allele in each SNP.

- `glVar`: computes the variance of the number of second allele in each SNP.

- `glDotProd`: computes dot products between (possibly centred/scaled) vectors of individuals - uses compiled C code - used by glPca.

Usage

```
glSum(x, alleleAsUnit = TRUE, useC = FALSE)
glNA(x, alleleAsUnit = TRUE)
glMean(x, alleleAsUnit = TRUE)
glVar(x, alleleAsUnit = TRUE)
glDotProd(x, center = FALSE, scale = FALSE, alleleAsUnit = FALSE,
                multicore = require("multicore"), n.cores = NULL)
```

Arguments

`x`	a genlight object

`alleleAsUnit` a logical indicating whether alleles are considered as units (i.e., a diploid genotype equals two samples, a triploid, three, etc.) or whether individuals are considered as units of information.

`center` a logical indicating whether SNPs should be centred to mean zero.

`scale` a logical indicating whether SNPs should be scaled to unit variance.

`useC` a logical indicating whether compiled C code should be used (TRUE) or not (FALSE, default).

`multicore` a logical indicating whether multiple cores -if available- should be used for the computations (TRUE, default), or not (FALSE); requires the package `multicore` to be installed (see details); this option cannot be used alongside useCoption.

`n.cores` if `multicore` is TRUE, the number of cores to be used in the computations; if NULL, then the maximum number of cores available on the computer is used.

Details

=== On the unit of information ===

In the cases where individuals can have different ploidy, computation of sums, means, etc. of allelic data depends on what we consider as a unit of information.

To estimate e.g. allele frequencies, unit of information can be considered as the allele, so that a diploid genotype contains two samples, a triploid individual, three samples, etc. In such a case, all computations are done directly on the number of alleles. This corresponds to `alleleAsUnit = TRUE`.

However, when the focus is put on studying differences/similarities between individuals, the unit of information is the individual, and all genotypes possess the same information no matter what their ploidy is. In this case, computations are made after standardizing individual genotypes to relative allele frequencies. This corresponds to `alleleAsUnit = FALSE`.

Note that when all individuals have the same ploidy, this distinction does not hold any more.

Value

A numeric vector containing the requested information.

Author(s)

Thibaut Jombart `<t.jombart@imperial.ac.uk>`

See Also

- `genlight`: class of object for storing massive binary SNP data.

- `dapc`: Discriminant Analysis of Principal Components.

- `glPca`: PCA for genlight objects.

- `glSim`: a simple simulator for genlight objects.

- `glPlot`: plotting genlight objects.

Examples

```
x <- new("genlight", list(c(0,0,1,1,0), c(1,1,1,0,0,1), c(2,1,1,1,1,NA)))
x
as.matrix(x)
ploidy(x)

## compute statistics - allele as unit ##
glNA(x)
glSum(x)
glMean(x)

## compute statistics - individual as unit ##
glNA(x, FALSE)
glSum(x, FALSE)
glMean(x, FALSE)

## explanation: data are taken as relative frequencies
temp <- as.matrix(x)/ploidy(x)
apply(temp,2, function(e) sum(is.na(e))) # NAs
apply(temp,2,sum, na.rm=TRUE) # sum
apply(temp,2,mean, na.rm=TRUE) # mean
```

---

genlight-class *Formal class "genlight"*

Description

The class genlight is a formal (S4) class for storing a genotypes of binary SNPs in a compact way, using a bit-level coding scheme. This storage is most efficient with haploid data, where the memory taken to represent data can reduced more than 50 times. However, genlight can be used for any level of ploidy, and still remain an efficient storage mode.

A genlight object can be constructed from vectors of integers giving the number of the second allele for each locus and each individual (see 'Objects of the class genlight' below).

genlight stores a multiple genotypes. Each genotype is stored as a SNPbin object.

Details

=== On the subsetting using [===

The function [accepts the following extra arguments:

treatOther a logical stating whether elements of the @other slot should be treated as well (TRUE), or not (FALSE). If treated, elements of the list are examined for a possible match of length (vectors, lists) or number of rows (matrices, data frames) with the number of individuals. Those who match are subsetted accordingly. Others are left as is, issuing a warning unless the argument quiet is set to TRUE.

quiet a logical indicating whether warnings should be issued when trying to subset components of the @other slot which do not match the number of individuals (TRUE), or not (FALSE, default).

Objects from the class genlight

`genlight` objects can be created by calls to `new("genlight", ...)`, where '...' can be the following arguments:

`gen` input genotypes, where each genotype is coded as a vector of numbers of the second allele. If a list, each slot of the list correspond to an individual; if a matrix or a data.frame, rows correspond to individuals and columns to SNPs. If individuals or loci are named in the input, these names will we stored in the produced object. All individuals are expected to have the same number of SNPs. Shorter genotypes are completed with NAs, issuing a warning.

`ploidy` an optional vector of integers indicating the ploidy of the genotypes. Genotypes can therefore have different ploidy. If not provided, ploidy will be guessed from the data (as the maximum number of second alleles in each individual).

`ind.names` an optional vector of characters giving the labels of the genotypes.

`loc.names` an optional vector of characters giving the labels of the SNPs.

`loc.all` an optional vector of characters indicating the alleles of each SNP; for each SNP, alleles must be coded by two letters separated by '/', e.g. 'a/t' is valid, but 'a t' or 'a|t' are not.

`chromosome` an optional factor indicating the chromosome to which each SNP belongs.

`position` an optional vector of integers indicating the position of the SNPs.

`other` an optional list storing miscellaneous information.

Slots

The following slots are the content of instances of the class `genlight`; note that in most cases, it is better to retrieve information via accessors (see below), rather than by accessing the slots manually.

`gen`**:** a list of genotypes stored as SNPbin objects.

`n.loc`**:** an integer indicating the number of SNPs of the genotype.

`ind.names`**:** a vector of characters indicating the names of genotypes.

`loc.names`**:** a vector of characters indicating the names of SNPs.

`loc.all`**:** a vector of characters indicating the alleles of each SNP.

`chromosome`**:** an optional factor indicating the chromosome to which each SNP belongs.

`position`**:** an optional vector of integers indicating the position of the SNPs.

`ploidy`**:** a vector of integers indicating the ploidy of each individual.

`pop`**:** a factor indicating the population of each individual.

`other`**:** a list containing other miscellaneous information.

Methods

Here is a list of methods available for `genlight` objects. Most of these methods are accessors, that is, functions which are used to retrieve the content of the object. Specific manpages can exist for accessors with more than one argument. These are indicated by a '*' symbol next to the method's name. This list also contains methods for conversion from `genlight` to other classes.

[`signature(x = "genlight")`: usual method to subset objects in R. Is to be applied as if the object was a matrix where genotypes were rows and SNPs were columns. Indexing can be done via vectors of signed integers or of logicals. See details for extra supported arguments.

show `signature(x = "genlight")`: printing of the object.

$ `signature(x = "genlight")`: similar to the @ operator; used to access the content of slots of the object.

$<- `signature(x = "genlight")`: similar to the @ operator; used to replace the content of slots of the object.

nInd `signature(x = "genlight")`: returns the number of individuals in the object.

nLoc `signature(x = "genlight")`: returns the number of SNPs in the object.

names `signature(x = "genlight")`: returns the names of the slots of the object.

indNames `signature(x = "genlight")`: returns the names of the individuals, if provided when the object was contructed.

indNames<- `signature(x = "genlight")`: sets the names of the individuals using a character vector of length `nInd(x)`.

locNames `signature(x = "genlight")`: returns the names of the loci, if provided when the object was contructed.

locNames<- `signature(x = "genlight")`: sets the names of the SNPs using a character vector of length `nLoc(x)`.

ploidy `signature(x = "genlight")`: returns the ploidy of the genotypes.

ploidy<- `signature(x = "genlight")`: sets the ploidy of the individuals using a vector of integers of size `nInd(x)`; if a single value is provided, the same ploidy is assumed for all individuals.

NA.posi `signature(x = "genlight")`: returns the indices of missing values (NAs) as a list with one vector of integer for each individual.

alleles `signature(x = "genlight")`: returns the names of the alleles of each SNPs, if provided when the object was contructed.

alleles<- `signature(x = "genlight")`: sets the names of the alleles of each SNPs using a character vector of length `nLoc(x)`; for each SNP, two alleles must be provided, separated by a "/", e.g. 'a/t', 'c/a', etc.

chromosome `signature(x = "genlight")`: returns a factor indicating the chromosome of each SNPs, or NULL if the information is missing.

chromosome<- `signature(x = "genlight")`: sets the chromosome to which SNPs belong using a factor of length `nLoc(x)`.

chr `signature(x = "genlight")`: shortcut for `chromosome`.

chr<- `signature(x = "genlight")`: shortcut for `chromosome<-`.

position `signature(x = "genlight")`: returns an integer vector indicating the position of each SNPs, or NULL if the information is missing.

position<- `signature(x = "genlight")`: sets the positions of the SNPs using an integer vector of length `nLoc(x)`.

pop `signature(x = "genlight")`: returns a factor indicating the population of each individual, if provided when the object was contructed.

pop<- `signature(x = "genlight")`: sets the population of each individual using a factor of length `nInd(x)`.

other `signature(x = "genlight")`: returns the content of the slot `@other`.

other<- `signature(x = "genlight")`: sets the content of the slot `@other`.

as.matrix `signature(x = "genlight")`: converts a `genlight` object into a matrix of integers, with individuals in rows and SNPs in columns. The S4 method 'as' can be used as well (e.g. as(x, "matrix")).

as.data.frame `signature(x = "genlight")`: same as `as.matrix`.

as.list `signature(x = "genlight")`: converts a `genlight` object into a list of genotypes coded as vector of integers (numbers of second allele). The S4 method 'as' can be used as well (e.g. as(x, "list")).

cbind `signature(x = "genlight")`: merges several genlight objects by column, i.e. regroups data of identical individuals genotyped for different SNPs.

rbind `signature(x = "genlight")`: merges several genlight objects by row, i.e. regroups data of different individuals genotyped for the same SNPs.

Author(s)

Thibaut Jombart (<t.jombart@imperial.ac.uk>)

See Also

Related class:
- SNPbin, for storing individual genotypes of binary SNPs

- genind, for storing other types of genetic markers.

Examples

```
## TOY EXAMPLE ##
## create and convert data
dat <- list(toto=c(1,1,0,0), titi=c(NA,1,1,0), tata=c(NA,0,3, NA))
x <- new("genlight", dat)
x

## examine the content of the object
names(x)
x@gen
x@gen[[1]]@snp # bit-level coding for first individual

## conversions
as.list(x)
as.matrix(x)

## round trips - must return TRUE
identical(x, new("genlight", as.list(x))) # list
identical(x, new("genlight", as.matrix(x))) # matrix
identical(x, new("genlight", as.data.frame(x))) # data.frame

## test subsetting
x[c(1,3)] # keep individuals 1 and 3
as.list(x[c(1,3)])
x[c(1,3), 1:2] # keep individuals 1 and 3, loci 1 and 2
as.list(x[c(1,3), 1:2])
x[c(TRUE,FALSE), c(TRUE,TRUE,FALSE,FALSE)] # same, using logicals
as.list(x[c(TRUE,FALSE), c(TRUE,TRUE,FALSE,FALSE)])


## REAL-SIZE EXAMPLE ##
## 50 genotypes of 1,000,000 SNPs
dat <- lapply(1:50, function(i) sample(c(0,1,NA), 1e6, prob=c(.5, .49, .01), replace=TRUE
```

```
names(dat) <- paste("indiv", 1:length(dat))
print(object.size(dat), unit="aut") # size of the original data

x <- new("genlight", dat) # conversion
x
print(object.size(x), unit="au") # size of the genlight object
object.size(dat)/object.size(x) # conversion efficiency


#### cbind, rbind ####
a <- new("genlight", list(toto=rep(1,10), tata=rep(c(0,1), each=5), titi=c(NA, rep(1,9))

ara <- rbind(a,a)
ara
as.matrix(ara)

aca <- cbind(a,a)
aca
as.matrix(aca)


#### subsetting @other ####
x <- new("genlight", list(a=1,b=0,c=1), other=list(1:3, letters,data.frame(2:4)))
x
other(x)
x[2:3]
other(x[2:3])
other(x[2:3, treatOther=FALSE])


#### seppop ####
pop(x) # no population info
pop(x) <- c("pop1","pop1", "pop2") # set population memberships
pop(x)
seppop(x)
```

`genpop class` *adegenet formal class (S4) for allele counts in populations*

Description

An object of class `genpop` contain alleles counts for several loci.
It contains several components (see 'slots' section).
Such object is obtained using `genind2genpop` which converts individuals genotypes of known population into a `genpop` object. Note that the function `summary` of a `genpop` object returns a list of components. Note that as in other S4 classes, slots are accessed using @ instead of \$.

Slots

`tab`**:** matrix of alleles counts for each combinaison of population -in rows- and alleles -in columns-. Rows and columns are given generic names.

`loc.names`**:** character vector containing the real names of the loci

`loc.fac`: locus factor for the columns of `tab`

`loc.nall`: integer vector giving the number of alleles per locus

`all.names`: list having one component per locus, each containing a character vector of alleles names

`call`: the matched call

`pop.names`: character vector containing the real names of the populations

`ploidy`: an integer indicating the degree of ploidy of the genotypes. Beware: 2 is not an integer, but as.integer(2) is.

`type`: a character string indicating the type of marker: 'codom' stands for 'codominant' (e.g. microstallites, allozymes); 'PA' stands for 'presence/absence' (e.g. AFLP).

`other`: (optional) a list containing other information

Extends

Class "`gen`", directly. Class "`popInfo`", directly.

Methods

names `signature(x = "genpop")`: give the names of the components of a genpop object

print `signature(x = "genpop")`: prints a genpop object

show `signature(object = "genpop")`: shows a genpop object (same as print)

summary `signature(object = "genpop")`: summarizes a genpop object, invisibly returning its content

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

See Also

`as.genpop`, `is.genpop`,`makefreq`, `genind`, `import2genind`, `read.genetix`, `read.genepop`, `read.fstat`, `na.replace`

Examples

```
obj1 <- import2genind(system.file("files/nancycats.gen",
package="adegenet"))
obj1


obj2 <- genind2genpop(obj1)
obj2

if(require(ade4)){
data(microsatt)
# use as.genpop to convert convenient count tab to genpop
obj3 <- as.genpop(microsatt$tab)
obj3

all(obj3@tab==microsatt$tab)
all(obj3@pop.names==rownames(microsatt$tab))
# it worked
```

```
# perform a correspondance analysis
obj4 <- genind2genpop(obj1,missing="chi2")
ca1 <- dudi.coa(as.data.frame(obj4@tab),scannf=FALSE)
s.label(ca1$li,sub="Correspondance Analysis",csub=2)
add.scatter.eig(ca1$eig,2,xax=1,yax=2,posi="top")
}
```

`genpop constructor` *genpop constructor*

Description

Constructor for genpop objects.
The function `genpop` creates a genpop object from a matrix of alleles counts where genotypes are in rows and alleles in columns. This table must have correct names for rows and columns.

The function `as.genpop` is an alias for `genpop` function.

`is.genpop` tests if an object is a valid genpop object.

Note: to get the manpage about genpop, please type 'class ? genpop'.

Usage

```
genpop(tab,prevcall=NULL, ploidy=as.integer(2), type=c("codom","PA"))
as.genpop(tab, prevcall=NULL, ploidy=as.integer(2), type=c("codom","PA"))
is.genpop(x)
```

Arguments

`tab`	a pop x alleles matrix which terms are numbers of alleles, i.e. like in a genpop object
`prevcall`	call of an object
`ploidy`	an integer indicating the degree of ploidy of the genotypes. Beware: 2 is not an integer, but as.integer(2) is.
`type`	a character string indicating the type of marker: 'codom' stands for 'codominant' (e.g. microstallites, allozymes); 'PA' stands for 'presence/absence' (e.g. AFLP, RAPD).
`x`	an object

Value

For `genpop` and `as.genpop`, a genpop object. For `is.genpop`, a logical.

Author(s)

Thibaut Jombart `<t.jombart@imperial.ac.uk>`

See Also

genpop class, and genind2genpop for conversion from a genind to a genpop object.

Examples

```
data(nancycats)
obj <- genind2genpop(nancycats)

# isolate one locus, fca77
obj <- seploc(obj)$"fca77"
obj
```

---

global.rtest *Global and local tests*

Description

These two Monte Carlo tests are used to assess the existence of global and local spatial structures. They can be used as an aid to interprete global and local components of spatial Principal Component Analysis (sPCA).

They rely on the decomposition of a data matrix X into global and local components using multiple regression on Moran's Eigenvector Maps (MEMs). They require a data matrix (X) and a list of weights derived from a connection network. X is regressed onto global MEMs (U+) in the global test and on local ones (U-) in the local test. One mean R^2 is obtained for each MEM, the k highest being summed to form the test statistic.

The reference distribution of these statistics are obtained by randomly permuting the rows of X.

Usage

```
global.rtest(X, listw, k = 1, nperm = 499)
local.rtest(X, listw, k = 1, nperm = 499)
```

Arguments

X	a data matrix, with variables in columns
listw	a list of weights of class listw. Can be obtained easily using the function chooseCN.
k	integer: the number of highest R^2 summed to form the test statistics
nperm	integer: the number of randomisations to be performed.

Details

This test is purely R code. A C or C++ version will be developped soon.

Value

An object of class randtest.

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

References

Jombart, T., Devillard, S., Dufour, A.-B. and Pontier, D. Revealing cryptic spatial patterns in genetic variability by a new multivariate method. *Heredity*, **101**, 92–103.

See Also

chooseCN, spca, monmonier

Examples

```
## Not run:
 data(sim2pop)
if(require(spdep)){
cn <- chooseCN(sim2pop@other$xy,ask=FALSE,type=1,plot=FALSE,res="listw")

# global test
Gtest <- global.rtest(sim2pop@tab,cn)
Gtest

# local test
Ltest <- local.rtest(sim2pop@tab,cn)
Ltest
}

## End(Not run)
```

---

glPca *Principal Component Analysis for genlight objects*

Description

These functions implement Principal Component Analysis (PCA) for massive SNP datasets stored as genlight object. This implementation has the advantage of never representing to complete data matrix, therefore making huge economies in terms of rapid access memory (RAM). When the `multicore` package is available, `glPca` uses multiple-core ressources for more efficient computations. `glPca` returns lists with the class `glPca` (see 'value').

Other functions are defined for objects of this class:

- `print`: prints the content of a `glPca` object.

- `scatter`: produces scatterplots of principal components, with a screeplot of eigenvalues as inset.

- `loadingplot`: plots the loadings of the analysis for one given axis, using an adapted version of the generic function `loadingplot`.

Usage

```
glPca(x, center = TRUE, scale = FALSE, nf = NULL, loadings = TRUE,
    alleleAsUnit = FALSE, useC = TRUE, multicore = require("multicore"),
  n.cores = NULL, returnDotProd=FALSE, matDotProd=NULL)

## S3 method for class 'glPca'
print(x, ...)

## S3 method for class 'glPca'
scatter(x, xax = 1, yax = 2, posi = "bottomleft", bg = "white",
    ratio = 0.3, label = rownames(x$scores), clabel = 1, xlim = NULL,
    ylim = NULL, grid = TRUE, addaxes = TRUE, origin = c(0, 0),
    include.origin = TRUE, sub = "", csub = 1, possub = "bottomleft",
    cgrid = 1, pixmap = NULL, contour = NULL, area = NULL, ...)

## S3 method for class 'glPca'
loadingplot(x, at=NULL, threshold=NULL, axis=1,
    fac=NULL, byfac=FALSE, lab=rownames(x$loadings), cex.lab=0.7, cex.fac=1,
    lab.jitter=0, main="Loading plot", xlab="SNP positions",
    ylab="Contributions", srt = 90, adj = c(0, 0.5), ...)
```

Arguments

x	for `glPca`, a genlight object; for `print`, `scatter`, and `loadingplot`, a `glPca` object.
center	a logical indicating whether the numbers of alleles should be centered; defaults to TRUE
scale	a logical indicating whether the numbers of alleles should be scaled; defaults to FALSE
nf	an integer indicating the number of principal components to be retained; if NULL, a screeplot of eigenvalues will be displayed and the user will be asked for a number of retained axes.
loadings	a logical indicating whether loadings of the alleles should be computed (TRUE, default), or not (FALSE). Vectors of loadings are not always useful, and can take a large amount of RAM when millions of SNPs are considered.
alleleAsUnit	a logical indicating whether alleles are considered as units (i.e., a diploid genotype equals two samples, a triploid, three, etc.) or whether individuals are considered as units of information.
useC	a logical indicating whether compiled C code should be used for faster computations; this option cannot be used alongside multicore option.
multicore	a logical indicating whether multiple cores -if available- should be used for the computations (TRUE, default), or not (FALSE); requires the package `multicore` to be installed (see details); this option cannot be used alongside useCoption.
n.cores	if `multicore` is TRUE, the number of cores to be used in the computations; if NULL, then the maximum number of cores available on the computer is used.
returnDotProd	a logical indicating whether the matrix of dot products between individuals should be returned (TRUE) or not (FALSE, default).

matDotProd	an optional matrix of dot products between individuals, NULL by default. This option is used internally to speed up computation time when re-running the same PCA several times. Leave this argument as NULL unless you really know what you are doing.
...	further arguments to be passed to other functions.
xax,yax	integers specifying which principal components should be shown in x and y axes.
posi,bg,ratio	arguments used to customize the inset in scatterplots of glPca results. See add.scatter documentation in the ade4 package for more details.
label,clabel,xlim,ylim,grid,addaxes,origin,include.origin,sub,csub,possub,cgrid,	arguments passed to s.class; see ?s.label for more information
at	an optional numeric vector giving the abscissa at which loadings are plotted. Useful when variates are SNPs with a known position in an alignement.
threshold	a threshold value above which values of x are identified. By default, this is the third quartile of x.
axis	an integer indicating the column of x to be plotted; used only if x is a matrix-like object.
fac	a factor defining groups of SNPs.
byfac	a logical stating whether loadings should be averaged by groups of SNPs, as defined by fac.
lab	a character vector giving the labels used to annotate values above the threshold.
cex.lab	a numeric value indicating the size of annotations.
cex.fac	a numeric value indicating the size of annotations for groups of observations.
lab.jitter	a numeric value indicating the factor of randomisation for the position of annotations. Set to 0 (by default) implies no randomisation.
main	the main title of the figure.
xlab	the title of the x axis.
ylab	the title of the y axis.
srt	rotation of the labels; see ?text.
adj	adjustment of the labels; see ?text.

Details

=== Using multiple cores ===

Most recent machines have one or several processors with multiple cores. R processes usually use one single core. The package multicore allows for parallelizing some computations on multiple cores, which decreases drastically computational time.

To use this functionality, you need to have the last version of the multicore package installed. To install it, type: install.packages("multicore",,"http://rforge.net/",type="source")

DO NOT use the version on CRAN, which is slightly outdated.

Lastly, note that using compiled C code (useC=TRUE)is an alternative for speeding up computations, but cannot be used together with the multicore option.

Value

=== glPca objects ===

The class `glPca` is a list with the following components:

`call`	the matched call.
`eig`	a numeric vector of eigenvalues.
`scores`	a matrix of principal components, containing the coordinates of each individual (in row) on each principal axis (in column).
`loadings`	(optional) a matrix of loadings, containing the loadings of each SNP (in row) for each principal axis (in column).

-

=== other outputs ===

Other functions have different outputs:
- `scatter` return the matched call.
- `loadingplot` returns information about the most contributing SNPs (see `loadingplot.default`)

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

See Also

- `genlight`: class of object for storing massive binary SNP data.
- `glSim`: a simple simulator for genlight objects.
- `glPlot`: plotting genlight objects.
- `dapc`: Discriminant Analysis of Principal Components.

Examples

```
## simulate a toy dataset
x <- glSim(50,4e3, 50, ploidy=2)
x
plot(x)

## perform PCA
pca1 <- glPca(x, nf=2)

## plot eigenvalues
barplot(pca1$eig, main="eigenvalues", col=heat.colors(length(pca1$eig)))

## basic plot
scatter(pca1, ratio=.2)

## plot showing groups
s.class(pca1$scores, pop(x), col=colors()[c(131,134)])
add.scatter.eig(pca1$eig,2,1,2)
```

glPlot *Plotting genlight objects*

Description

genlight object can be plotted using the function glPlot, which is also used as the dedicated plot method. These functions relie on image to represent SNPs data. More specifically, colors are used to represent the number of second allele for each locus and individual.

Usage

```
glPlot(x, col=NULL, legend=TRUE, posi="bottomleft", bg=rgb(1,1,1,.5),...)

## S4 method for signature 'genlight'
plot(x, y=NULL, col=NULL, legend=TRUE, posi="bottomleft", bg=rgb(1,1,1,.5),...)
```

Arguments

x	a genlight object.
col	an optional color vector; the first value corresponds to 0 alleles, the last value corresponds to the ploidy level of the data. Therefore, the vector should have a length of (ploidy(x)+1).
legend	a logical indicating whether a legend should be added to the plot.
posi	a character string indicating where the legend should be positioned. Can be any concatenation of "bottom"/"top" and "left"/"right".
bg	a color used as a background for the legend; by default, transparent white is used; this may not be supported on some devices, and therefore background should be specified (e.g. bg="white").
...	further arguments to be passed to image.
y	ununsed argument, present for compatibility with the plot generic.

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

See Also

- genlight: class of object for storing massive binary SNP data.
- glSim: a simple simulator for genlight objects.
- glPca: PCA for genlight objects.

Examples

```
## simulate data
x <- glSim(100, 1e3, n.snp.struc=100, ploid=2)

## default plot
glPlot(x)
plot(x) # identical plot
```

```
## disable legend
plot(x, leg=FALSE)

## use other colors
plot(x, col=heat.colors(3), bg="white")
```

---

glSim *Simulation of simple genlight objects*

Description

The function glSim simulates simple SNP data with the possibility of contrasted structures between two groups. Returned objects are instances of the class genlight.

Usage

```
glSim(n.ind, n.snp.nonstruc, n.snp.struc = 0, grp.size = round(n.ind/2),
    ploidy = 1, alpha = 0, block.size = NULL, LD = FALSE)
```

Arguments

n.ind	an integer indicating the number of individuals to be simulated.
n.snp.nonstruc	an integer indicating the number of non-structured SNPs to be simulated; for these SNPs, all individuals are drawn from the same binomial distribution.
n.snp.struc	an integer indicating the number of structured SNPs to be simulated; for these SNPs, different binomial distributions are used for the two simulated groups; frequencies of the derived alleles in groups A and B are built to differ (see details).
grp.size	an integer indicating the size of the first group of individuals (noted 'A'); by default, both groups have the same size.
ploidy	an integer indicating the ploidy of the simulated genotypes.
alpha	asymmetry parameter: a numeric value between 0 and 0.5, used to enforce allelic differences between the groups (see details); ignored if LD=TRUE.
block.size	an optional integer indicating the number of SNPs to be handled at a time during the simulations. By default, all SNPs are simulated at the same time, but RAM can limit this operation. Using blocks of a few hundred or thousand SNPs decreases RAM requirement at a cost of more computational time. When LD=TRUE, large blocks will come at a large costs in terms of computational time and RAM, since the underlying matrices of correlation will be large.
LD	a logical indicating whether loci should be displaying linkage disequilibrium (TRUE) or be generated independently (FALSE, default). When set to TRUE, data are generated by blocks of correlated SNPs (see details).

Details

=== Allele frequencies in contrasted groups ===

When `n.snp.struc` is greater than 0, some SNPs are simulated in order to differ between groups (noted 'A' and 'B'). Such differences can be achieved differently depending on whether loci are independent (`LD=FALSE`), or not (`LD=TRUE`). In the first case, different patterns between groups are achieved by using different frequencies of the second allele for A and B, denoted p_A and p_B. For a given SNP, p_A is drawn from a uniform distribution between 0 and (0.5 - alpha). p_B is then computed as 1 - p_A. Therefore, differences between groups are mild for alpha=0, and total for alpha = 0.5.

Whenever loci are linked (`LD=TRUE`), this option is no longer available. Differences between groups merely occur by drawing alleles from randomly generated, group-specific allele frequencies.

=== Linked or independent loci ===

Independent loci (`LD=FALSE`) are simulated using the standard binomial distribution, with randomly generated allele frequencies. Linked loci (`LD=FALSE`) are trickier towe need to simulate discrete variables with pre-defined correlation structure.

Here, we first generate deviates from multivariate normal distributions with randomly generated correlation structures. These variables are then discretized using the quantiles of the distribution. Further improvement of the procedure will aim at i) specifying the strength of the correlations between blocks of alleles and ii) enforce contrasted structures between groups.

Value

A genlight object.

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

See Also

- `genlight`: class of object for storing massive binary SNP data.
- `glPlot`: plotting genlight objects.
- `glPca`: PCA for genlight objects.

Examples

```
## no structure
x <- glSim(100, 1e3, ploid=2)
plot(x)

## 1,000 non structured SNPs, 100 structured SNPs
x <- glSim(100, 1e3, n.snp.struc=100, ploid=2)
plot(x)

## 1,000 non structured SNPs, 100 structured SNPs, ploidy=4
x <- glSim(100, 1e3, n.snp.struc=100, ploid=4)
plot(x)

## same thing, stronger differences between groups
x <- glSim(100, 1e3, n.snp.struc=100, ploid=2, alpha=0.4)
plot(x)
```

```
##  same thing, loci with LD structures
x <- glSim(100, 1, n.snp.struc=100, ploid=2, alpha=0.4, LD=TRUE, block.size=100)
plot(x)
```

`gstat.randtest` *Goudet's G-statistic Monte Carlo test for genind object*

Description

The function `gstat.randtest` implements Goudet's G-statistic Monte Carlo test (`g.stats.glob`, package `hierfstat`) for `genind` object.
The output is an object of the class `randtest` (package `ade4`) from a `genind` object.

This procedure tests for genetic structuring of individuals using 3 different schemes (see details).

Usage

```
gstat.randtest(x,pop=NULL, method=c("global","within","between"),
sup.pop=NULL, sub.pop=NULL, nsim=499)
```

Arguments

`x`	an object of class `genind`.
`pop`	a factor giving the 'population' of each individual. If NULL, pop is seeked from x\@pop. Note that the term population refers in fact to any grouping of individuals'.
`method`	a character (if a vector, only first argument is kept) giving the method to be applied: 'global', 'within' or 'between' (see details).
`sup.pop`	a factor indicating any grouping of individuals at a larger scale than 'pop'. Used in 'within' method.
`sub.pop`	a factor indicating any grouping of individuals at a finer scale than 'pop'. Used in 'between' method.
`nsim`	number of simulations to be used for the randtest.

Details

This G-statistic Monte Carlo procedure tests for population structuring at different levels. This is determined by the argument 'method':

- "global": tests for genetic structuring given 'pop'.
- "within": tests for genetic structuring within 'pop' inside each 'sup.pop' group (i.e., keeping sup.pop effect constant).
- "between": tests for genetic structuring between 'pop' keeping individuals in their 'sub.pop' groups (i.e., keeping sub.pop effect constant).

Value

Returns an object of the class `randtest` (package `ade4`).

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

See Also

fstat,

genind2hierfstat

Examples

```
if(require(hierfstat)){
# here the example of g.stats.glob is taken using gstat.randtest
data(gtrunchier)
x <- df2genind(X=gtrunchier[,-c(1,2)],pop=gtrunchier$Patch)

# test in hierfstat
gtr.test<- g.stats.glob(gtrunchier[,-1])
gtr.test

# randtest version
x.gtest <- gstat.randtest(x,nsim=99)
x.gtest
plot(x.gtest)

# pop within sup.pop test
gstat.randtest(x,nsim=99,method="within",sup.pop=gtrunchier$Locality)

# pop test with sub.pop kept constant
gstat.randtest(x,nsim=99,pop=gtrunchier$Locality,method="between",sub.pop=gtrunchier$Patc
}
```

H3N2 *Seasonal influenza (H3N2) HA segment data*

Description

The dataset H3N2 consists of 1903 strains of seasonal influenza (H3N2) distributed worldwide, and typed at 125 SNPs located in the hemagglutinin (HA) segment. It is stored as an R object with class genind and can be accessed as usual using data(H3N2) (see example). These data were gathered from DNA sequences available from Genbank (http://www.ncbi.nlm.nih.gov/Genbank/).

The data file usflu.fasta is a toy dataset also gathered from Genbank, consisting of the aligned sequences of 80 seasonal influenza isolates (HA segment) sampled in the US, in fasta format. This file is installed alongside the package; the path to this file is automatically determined by R using system.file (see example in this manpage and in ?fasta2genlight) as well.

Usage

```
data(H3N2)
```

Format

`H3N2` is a genind object with several data frame as supplementary components (`H3N2@other`) `slort`, which contains the following items:

x a `data.frame` containing miscellanous annotations of the sequences.

xy a matrix with two columns indicating the geographic coordinates of the strains, as longitudes and latitudes.

epid a character vector indicating the epidemic of the strains.

Source

This dataset was prepared by Thibaut Jombart (t.jombart@imperia.ac.uk), from annotated sequences available on Genbank (http://www.ncbi.nlm.nih.gov/Genbank/).

References

Jombart, T., Devillard, S. and Balloux, F. Discriminant analysis of principal components: a new method for the analysis of genetically structured populations. Submitted to *BMC genetics*.

Examples

```
#### H3N2 ####
## LOAD DATA
data(H3N2)
H3N2

## set population to yearly epidemics
pop(H3N2) <- factor(H3N2$other$epid)



## PERFORM DAPC - USE POPULATIONS AS CLUSTERS
## to reproduce exactly analyses from the paper, use "n.pca=1000"
dapc1 <- dapc(H3N2, all.contrib=TRUE, scale=FALSE, n.pca=150, n.da=5)
dapc1

## (see ?dapc for details about the output)


## SCREEPLOT OF EIGENVALUES
barplot(dapc1$eig, main="H3N2 - DAPC eigenvalues")


## SCATTERPLOT (axes 1-2)
scatter(dapc1, posi.da="topleft", cstar=FALSE, cex=2, pch=17:22,
solid=.5, bg="white")




#### usflu.fasta ####
myPath <- system.file("files/usflu.fasta",package="adegenet")
myPath

## extract SNPs from alignments using fasta2genlight
```

```
## see ?fasta2genlight for more details
obj <- fasta2genlight(myPath, chunk=10) # process 10 sequences at a time
obj
```

`haploGen` *Simulation of genealogies of haplotypes*

Description

The function `haploGen` implements simulations of genealogies of haplotypes. This forward-time, individual-based simulation tool allow haplotypes to replicate and mutate according to specified parameters, and keeps track of entire genealogies.

Simulations can be spatially explicit or not (see `geo.sim` argument). In the first case, haplotypes are assigned to locations on a regular grip. New haplotypes disperse from their ancestor's location according to a random Poisson diffusion, or alternatively according to a pre-specified migration scheme. This tool does not allow for simulating selection or linkage disequilibrium.

Produced objects are lists with the class `haploGen`; see 'value' section for more information on this class. Other functions are available to print, plot, subset, sample or convert `haploGen` objects. A seqTrack method is also provided for analysing `haploGen` objects.

Usage

```
haploGen(seq.length=10000, mu=0.0001, t.max=20,
              gen.time=function(){round(rnorm(1,5,1))},
              repro=function(){round(rnorm(1,2,1))}, max.nb.haplo=1e3,
              geo.sim=TRUE, grid.size=5, lambda.xy=0.5,
              mat.connect=NULL, ini.n=1, ini.xy=NULL)
## S3 method for class 'haploGen'
print(x, ...)
## S3 method for class 'haploGen'
x[i, j, drop=FALSE]
## S3 method for class 'haploGen'
labels(object, ...)
## S3 method for class 'haploGen'
as.POSIXct(x, tz="", origin=as.POSIXct("2000/01/01"), ...)
## S3 method for class 'haploGen'
seqTrack(x, best=c("min","max"), prox.mat=NULL, ...)
as.seqTrack.haploGen(x)
plotHaploGen(x, annot=FALSE, date.range=NULL, col=NULL, bg="grey", add=FALSE, ..
sample.haploGen(x, n)
## S4 method for signature 'haploGen,graphNEL'
coerce(from, to, strict=TRUE)
```

Arguments

`seq.length`	an integer indicating the length of the simulated haplotypes, in number of nucleotides.
`mu`	the mutation rate, in number of mutation per site and per time unit. Can be a (fixed) number or a function returning a number (then called for each replication event).

`t.max` an integer indicating the maximum number of time units to run the simulation for.

`gen.time` an integer indicating the generation time, in number of time units. Can be a (fixed) number or a function returning a number (then called for each reproduction event).

`repro` an integer indicating the number of descendents per haplotype. Can be a (fixed) number or a function returning a number (then called for each reproduction event).

`max.nb.haplo` an integer indicating the maximum number of haplotypes handled at any time of the simulation, used to control the size of the produced object. Larger number will lead to slower simulations. If this number is exceeded, the genealogy is prunded to as to keep this number of haplotypes.

`geo.sim` a logical stating whether simulations should be spatially explicit (TRUE, default) or not (FALSE). Spatially-explicit simulations are slightly slower than their non-spatial counterpart.

`grid.size` the size of the square grid of possible locations for spatial simulations. The total number of locations will be this number squared.

`lambda.xy` the parameter of the Poisson distribution used to determine dispersion in x and y axes.

`mat.connect` a matrix of connectivity describing migration amongts all pairs of locations. `mat.connect[i,j]` indicates the probability, being in 'i', to migrate to 'j'. The rows of this matrix thus sum to 1. It has as many rows and columns as there are locations, with row 'i' / column 'j' corresponding to locations number 'i' and 'j'. Locations are numbered as in a matrix in which rows and columns are respectively x and y coordinates. For instance, in a 5x5 grid, locations are numbered as in `matrix(1:25,5,5)`.

`ini.n` an integer specifying the number of (identical) haplotypes to initiate the simulation

`ini.xy` a vector of two integers giving the x/y coordinates of the initial haplotype.

`x,object` `haploGen` objects.

`i,j, drop` `i` is a vector used for subsetting the object. For instance, `i=1:3` will retain only the first three haplotypes of the genealogy. `j` and `drop` are only provided for compatibility, but not used.

`best, prox.mat` arguments to be passed to the `seqTrack` function. See documentation of `seqTrack` for more information.

`annot,date.range,col,bg,add` arguments to be passed to `plotSeqTrack`.

`n` an integer indicating the number of haplotypes to be retained in the sample

`from, to` arguments of the conversion function, for converting a `haploGen` object into a `graphNEL-class`.

`tz, origin` aguments to be passed to `as.POSIXct` (see ?as.POSIXct)

`...` further arguments to be passed to other methods

`strict` a logical used for compatibility with `as` generic function, but not used in the conversion. See `setAs` for more information.

Details

=== Dependencies with other packages ===
- ape package is required as it implements efficient handling of DNA sequences used in `haploGen` objects. To install this package, simply type:
`install.packages("ape")`

- for various purposes including plotting, converting genealogies to graphs (`graphNEL-class` class) can be useful. This requires the packages graph, and possibly Rgraphviz for plotting. These packages are not on CRAN, but on Bioconductor. To install them, use:
source("http://bioconductor.org/biocLite.R")
biocLite("graph")
biocLite("Rgraphviz")

See the respective vignettes for more information on using these packages.

=== Converting haploGen objects to graphs ===
`haploGen` objects can be converted to `graphNEL-class` objects, which can in turn be plotted and manipulated using classical graph tools. Simply use 'as(x, "graphNEL")' where 'x' is a `haploGen` object. This functionality requires the `graph` package (see 'details').

Value

=== haploGen class ===
`haploGen` objects are lists containing the following slots:
- seq: DNA sequences in the DNAbin matrix format
- dates: dates of appearance of the haplotypes
- ances: a vector of integers giving the index of each haplotype's ancestor
- id: a vector of integers giving the index of each haplotype
- xy: (optional) a matrix of spatial coordinates of haplotypes
- call: the matched call

=== misc functions ===
- as.POSIXct: returns a vector of dates with POSIXct format
- labels: returns the labels of the haplotypes
- as.seqTrack: returns a seqTrack object. Note that this object is not a proper seqTrack analysis, but just a format conversion convenient for plotting `haploGen` objects.

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

References

Jombart T, Eggo R, Dodd P, Balloux F (2010) Reconstructing disease outbreaks from genetic data: a graph approach. Heredity. doi: 10.1038/hdy.2010.78.

Examples

```
if(require(ape)){
## PERFORM SIMULATIONS
x <- haploGen(repro=2)
x

## PLOT SPATIAL SPREAD
plotHaploGen(x, bg="white")
title("Spatial dispersion of the haplotypes")
```

```
## PLOT GENEALOGY
if(require(graph) & require(Rgraphviz)){
g=as(x, "graphNEL")
g
renderGraph(layoutGraph(g))
}


## USE SEQTRACK RECONSTRUCTION
x.recons <- seqTrack(x)
mean(x.recons$ances==x$ances, na.rm=TRUE) # proportion of correct reconstructions

}
```

`haploPop` *Simulation of populations of haplotypes*

Description

Important: these functions are parts of a publication currently under review. They will be documented once accepted for publication. Please email the author if you are interested in using it.

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

`Hs` *Expected heterozygosity*

Description

This function computes the expected heterozygosity (Hs) within populations of a genpop object. This function is available for codominant markers (`@type="codom"`) only. Hs is commonly used for measuring within population genetic diversity (and as such, it still has sense when computed from haploid data).

Usage

```
Hs(x, truenames=TRUE)
```

Arguments

`x`	an object of class genpop.
`truenames`	a logical indicating whether true labels (as opposed to generic labels) should be used to name the output.

Details

Let *m(k)* be the number of alleles of locus *k*, with a total of *K* loci. We note f_i the allele frequency of allele *i* in a given population. Then, Hs is given for a given population by:

$\frac{1}{K}\sum_{k=1}^{K}(1-\sum_{i=1}^{m(k)}f_i^2)$

Value

A vector of Hs values (one value per population).

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

Examples

```
data(nancycats)
Hs(genind2genpop(nancycats))
```

`HWE.test.genind` *Hardy-Weinberg Equilibrium test for multilocus data*

Description

The function `HWE.test` is a generic function to perform Hardy-Weinberg Equilibrium tests defined by the `genetics` package. adegenet proposes a method for `genind` objects.

The output can be of two forms:
- a list of tests (class `htest`) for each locus-population combinaison
- a population x locus matrix containing p-values of the tests

Usage

```
## S3 method for class 'genind'
HWE.test(x,pop=NULL,permut=FALSE,nsim=1999,hide.NA=TRUE,res.type=c("full","matri
```

Arguments

`x`	an object of class `genind`.
`pop`	a factor giving the population of each individual. If NULL, pop is seeked from x\$pop.
`permut`	a logical passed to `HWE.test` stating whether Monte Carlo version (TRUE) should be used or not (FALSE, default).
`nsim`	number of simulations if Monte Carlo is used (passed to `HWE.test`).
`hide.NA`	a logical stating whether non-tested loci (e.g., when an allele is fixed) should be hidden in the results (TRUE, default) or not (FALSE).
`res.type`	a character or a character vector whose only first argument is considered giving the type of result to display. If "full", then a list of complete tests is returned. If "matrix", then a matrix of p-values is returned.

Details

Monte Carlo procedure is quiet computer-intensive when large datasets are involved. For more precision on the performed test, read `HWE.test` documentation (`genetics` package).

Value

Returns either a list of tests or a matrix of p-values. In the first case, each test is designated by locus first and then by population. For instance if `res` is the "full" output of the function, then the test for population "PopA" at locus "Myloc" is given by res$Myloc$PopA. If `res` is a matrix of p-values, populations are in rows and loci in columns. P-values are given for the upper-tail: they correspond to the probability that an oberved chi-square statistic as high as or higher than the one observed occured under H0 (HWE).

In all cases, NA values are likely to appear in fixed loci, or entirely non-typed loci.

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

See Also

`HWE.test`,`chisq.test`

Examples

```
data(nancycats)
obj <- nancycats
if(require(genetics)){
obj.test <- HWE.test(obj)

# pvalues matrix to have a preview
HWE.test(obj,res.type="matrix")

#more precise view to...
obj.test$fca90$P10
}
```

---

hybridize *Simulated hybridization between two samples of populations*

Description

The function `hybridize` performs hybridization between two set of genotypes stored in genind objects (referred as the "2 populations"). Allelic frequencies are derived for each population, and then gametes are sampled following a multinomial distribution.

The result consists in a set of 'n' genotypes, with different possible outputs (see 'res.type' argument).

Usage

```
hybridize(x1, x2, n, pop=NULL, res.type=c("genind","df","STRUCTURE"), file=NULL,
  quiet=FALSE, sep="/", hyb.label="h")
```

Arguments

x1	a genind object
x2	a genind object
n	an integer giving the number of hybrids requested
pop	a character string giving naming the population of the created hybrids. If NULL, will have the form "x1-x2"
res.type	a character giving the type of output requested. Must be "genind" (default), "df" (i.e. data.frame like in genind2df), or "STRUCTURE" to generate a .str file readable by STRUCTURE (in which case the 'file' must be supplied). See 'details' for STRUCTURE output.
file	a character giving the name of the file to be written when 'res.type' is "STRUCTURE"; if NULL, a the created file is of the form "hybrids_[the current date].str".
quiet	a logical specifying whether the writing to a file (when 'res.type' is "STRUCTURE") should be announced (FALSE, default) or not (TRUE).
sep	a character used to separate two alleles
hyb.label	a character string used to construct the hybrids labels; by default, "h", which gives labels: "h01", "h02", "h03",...

Details

If the output is a STRUCTURE file, this file will have the following caracteristics:
- file contains the genotypes of the parents, and then the genotypes of hybrids
- the first column identifies genotypes
- the second column identifies the population (1 and 2 for parents x1 and x2; 3 for hybrids)
- the first line contains the names of the markers
- one row = one genotype (onerowperind will be true)
- missing values coded by "-9" (the software's default)

Value

A genind object (by default), or a data.frame of alleles (res.type="df"). No R output if res.type="STRUCTURE" (results written to the specified file).

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

Examples

```
## Let's make some cattle hybrids
##
data(microbov)

## first, isolate each breed
temp <- seppop(microbov)
```

```
names(temp)

salers <- temp$Salers
zebu <- temp$Zebu
borgou <- temp$Borgou
somba <- temp$Somba

## let's make some... Zeblers
zebler <- hybridize(salers, zebu, n=40)

## and some Somgou
somgou <- hybridize(somba, borgou, n=40)

## now let's merge all data into a single genind
newDat <- repool(microbov, zebler, somgou)

## make a correspondance analysis
## and see where hybrids are placed
if(require(ade4)){
X <- genind2genpop(newDat,missing="chi2",quiet=TRUE)
coa1 <- dudi.coa(as.data.frame(X$tab),scannf=FALSE,nf=3)
s.label(coa1$li,label=X$pop.names)
add.scatter.eig(coa1$eig,2,1,2)
}
```

`import` *Importing data from several softwares to a genind object*

Description

Their are several ways to import genotype data to a genind object: i) from a data.frame with a given format (see `df2genind`), ii) from a file with a recognized extension, or iii) from an alignement of sequences (see `DNAbin2genind`).

The function `import2genind` detects the extension of the file given in argument and seeks for an appropriate import function to create a `genind` object.
Current recognized formats are :
- GENETIX files (.gtx)
- Genepop files (.gen)
- Fstat files (.dat)
- STRUCTURE files (.str or .stru)

Usage

```
import2genind(file,missing=NA,quiet=FALSE, ...)
```

Arguments

`file`	a character string giving the path to the file to convert, with the appropriate extension.
`missing`	can be NA, 0 or "mean". See details section.

quiet	logical stating whether a conversion message must be printed (TRUE,default) or not (FALSE).
...	other arguments passed to the appropriate 'read' function (currently passed to read.structure)

Details

There are 3 treatments for missing values:
- NA: kept as NA.

- 0: allelic frequencies are set to 0 on all alleles of the concerned locus. Recommended for a PCA on compositionnal data.

- "mean": missing values are replaced by the mean frequency of the corresponding allele, computed on the whole set of individuals. Recommended for a centred PCA.

Beware: same data in different formats are not expected to produce exactly the same genind objects.
For instance, conversions made by GENETIX to Fstat may change the the sorting of the genotypes; GENETIX stores individual names whereas Fstat does not; Genepop chooses a sample's name from the name of its last genotype; etc.

Value

an object of the class genind

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

References

Belkhir K., Borsa P., Chikhi L., Raufaste N. & Bonhomme F. (1996-2004) GENETIX 4.05, logiciel sous Windows TM pour la génétique des populations. Laboratoire Génome, Populations, Interactions, CNRS UMR 5000, Université de Montpellier II, Montpellier (France).

Pritchard, J.; Stephens, M. & Donnelly, P. (2000) Inference of population structure using multilocus genotype data. *Genetics*, **155**: 945-959

Raymond M. & Rousset F, (1995). GENEPOP (version 1.2): population genetics software for exact tests and ecumenicism. *J. Heredity*, **86**:248-249

Fstat (version 2.9.3). Software by Jerome Goudet. http://www2.unil.ch/popgen/softwares/fstat.htm

Excoffier L. & Heckel G.(2006) Computer programs for population genetics data analysis: a survival guide *Nature*, **7**: 745-758

See Also

import2genind, read.genetix, read.fstat, read.structure, read.genepop

Examples

```
import2genind(system.file("files/nancycats.gtx",
package="adegenet"))

import2genind(system.file("files/nancycats.dat",
package="adegenet"))

import2genind(system.file("files/nancycats.gen",
package="adegenet"))

import2genind(system.file("files/nancycats.str",
package="adegenet"), onerowperind=FALSE, n.ind=237, n.loc=9, col.lab=1, col.pop=2, ask=FA
```

Inbreeding estimation

Likelihood-based estimation of inbreeding

Description

The function `inbreeding` estimates the inbreeding coefficient of an individuals (F) by computing its likelihood function. It can return either the density of probability of F, or a sample of F values from this distribution. This operation is performed for all the individuals of a genind object. Any ploidy greater than 1 is acceptable.

Usage

```
inbreeding(x, pop = NULL, truenames = TRUE, res.type = c("sample", "function"),
```

Arguments

`x`	an object of class genind.
`pop`	a factor giving the 'population' of each individual. If NULL, pop is seeked from `pop(x)`. Note that the term population refers in fact to any grouping of individuals'.
`truenames`	a logical indicating whether true names should be used (TRUE, default) instead of generic labels (FALSE); used if res.type is "matrix".
`res.type`	a character string matching "sample" or "function", specifying whether the output should be a function giving the density of probability of F values ("function") or a sample of F values taken from this distribution ("sample", default).
`N`	an integer indicating the size of the sample to be taken from the distribution of F values.
`M`	an integer indicating the number of different F values to be used to generate the sample. Values larger than N are recommended to avoid poor sampling of the distribution.

Details

Let F denote the inbreeding coefficient, defined as the probability for an individual to inherit two identical alleles from a single ancestor.

Let p_i refer to the frequency of allele i in the population. Let h be an variable which equates 1 if the individual is homozygote, and 0 otherwise. For one locus, the probability of being homozygote is computed as:

$F + (1 - F) \sum_i p_i^2$

The probability of being heterozygote is: $1 - (F + (1 - F) \sum_i p_i^2)$

The likelihood of a genotype is defined as the probability of being the observed state (homozygote or heterozygote). In the case of multilocus genotypes, log-likelihood are summed over the loci.

Value

A named list with one component for each individual, each of which is a function or a vector of sampled F values (see `res.type` argument).

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

See Also

`Hs`: computation of expected heterozygosity.

Examples

```
## cattle breed microsatellite data
data(microbov)

## isolate Lagunaire breed
lagun <- seppop(microbov)$Lagunaire

## estimate inbreeding - return sample of F values
Fsamp <- inbreeding(lagun)

## plot the first 10 results
invisible(sapply(Fsamp[1:10], function(e) plot(density(e), xlab="F", xlim=c(0,1), main="D

## compute means for all individuals
Fmean=sapply(Fsamp, mean)
hist(Fmean, col="orange", xlab="mean value of F", main="Distribution of mean F across ind

## estimate inbreeding - return proba density functions
Fdens <- inbreeding(lagun, res.type="function")

## view function for the first individual
Fdens[[1]]

## plot the first 10 functions
invisible(sapply(Fdens[1:10], plot, ylab="Density", main="Density of probability of F val
```

isPoly-methods *Assess polymorphism in genind/genpop objects*

Description

The simple function `isPoly` can be used to check which loci are polymorphic, or alternatively to check which alleles give rise to polymorphism.

Usage

```
## S4 method for signature 'genind'
isPoly(x, by=c("locus","allele"), thres=1/100)
## S4 method for signature 'genpop'
isPoly(x, by=c("locus","allele"), thres=1/100)
```

Arguments

`x`	a genind and genpop object
`by`	a character being "locus" or "allele", indicating whether results should indicate polymorphic loci ("locus"), or alleles giving rise to polymorphism ("allele").
`thres`	a numeric value giving the minimum frequency of an allele giving rise to polymorphism (defaults to 0.01).

Value

A vector of logicals.

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

Examples

```
data(nancycats)
isPoly(nancycats,by="loc", thres=0.1)
isPoly(nancycats[1:3],by="loc", thres=0.1)
genind2df(nancycats[1:3])
```

loadingplot *Represents a cloud of points with colors*

Description

The `loadingplot` function represents positive values of a vector and identifies the values above a given threshold. It can also indicate groups of observations provided as a factor.

Such graphics can be used, for instance, to assess the weight of each variable (loadings) in a given analysis.

Usage

```
loadingplot(x, ...)

## Default S3 method:
loadingplot(x, at=NULL, threshold=quantile(x,0.75), axis=1, fac=NULL, byfac=FALS
            lab=NULL, cex.lab=0.7, cex.fac=1, lab.jitter=0,
            main="Loading plot", xlab="Variables", ylab="Loadings", srt = 0, adj
```

Arguments

x	either a vector with numeric values to be plotted, or a matrix-like object containing numeric values. In such case, the x[,axis] is used as vector of values to be plotted.
at	an optional numeric vector giving the abscissa at which loadings are plotted. Useful when variates are SNPs with a known position in an alignement.
threshold	a threshold value above which values of x are identified. By default, this is the third quartile of x.
axis	an integer indicating the column of x to be plotted; used only if x is a matrix-like object.
fac	a factor defining groups of observations.
byfac	a logical stating whether loadings should be averaged by groups of observations, as defined by fac.
lab	a character vector giving the labels used to annotate values above the threshold; if NULL, names are taken from the object.
cex.lab	a numeric value indicating the size of annotations.
cex.fac	a numeric value indicating the size of annotations for groups of observations.
lab.jitter	a numeric value indicating the factor of randomisation for the position of annotations. Set to 0 (by default) implies no randomisation.
main	the main title of the figure.
xlab	the title of the x axis.
ylab	the title of the y axis.
srt	rotation of the labels; see ?text.
adj	adjustment of the labels; see ?text.
...	further arguments to be passed to the plot function.

Value

Invisibly returns a list with the following components:
- threshold: the threshold used
- var.names: the names of observations above the threshold
- var.idx: the indices of observations above the threshold
- var.values: the values above the threshold

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

Examples

```
x <- runif(20)
names(x) <- letters[1:20]
grp <- factor(paste("group", rep(1:4,each=5)))

## basic plot
loadingplot(x)

## adding groups
loadingplot(x,fac=grp,main="My title",cex.lab=1)
```

---

makefreq *Function to generate allelic frequencies*

Description

The function `makefreq` generates a table of allelic frequencies from an object of class `genpop`.

Usage

```
makefreq(x,quiet=FALSE,missing=NA,truenames=TRUE)
```

Arguments

`x`	an object of class `genpop`.
`quiet`	logical stating whether a conversion message must be printed (TRUE,default) or not (FALSE).
`missing`	treatment for missing values. Can be NA, 0 or "mean" (see details)
`truenames`	a logical indicating whether true labels (as opposed to generic labels) should be used to name the output.

Details

There are 3 treatments for missing values:
- NA: kept as NA.
- 0: missing values are considered as zero. Recommended for a PCA on compositionnal data.
- "mean": missing values are given the mean frequency of the corresponding allele. Recommended for a centred PCA.

Value

Returns a list with the following components:

`tab`	matrix of allelic frequencies (rows: populations; columns: alleles).
`nobs`	number of observations (i.e. alleles) for each population x locus combinaison.
`call`	the matched call

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

See Also

genpop

Examples

```
data(microbov)
obj1 <- microbov

obj2 <- genind2genpop(obj1)

Xfreq <- makefreq(obj2,missing="mean")

if(require(ade4)){

# perform a correspondance analysis on counts data

Xcount <- genind2genpop(obj1,missing="chi2")
ca1 <- dudi.coa(as.data.frame(Xcount@tab),scannf=FALSE)
s.label(ca1$li,sub="Correspondance Analysis",csub=1.2)
add.scatter.eig(ca1$eig,nf=2,xax=1,yax=2,posi="topleft")

# perform a principal component analysis on frequency data
pca1 <- dudi.pca(Xfreq$tab,scale=FALSE,scannf=FALSE)
s.label(pca1$li,sub="Principal Component Analysis",csub=1.2)
add.scatter.eig(pca1$eig,nf=2,xax=1,yax=2,posi="top")
}
```

---

microbov *Microsatellites genotypes of 15 cattle breeds*

Description

This data set gives the genotypes of 704 cattle individuals for 30 microsatellites recommended by the FAO. The individuals are divided into two countries (Afric, France), two species (Bos taurus, Bos indicus) and 15 breeds. Individuals were chosen in order to avoid pseudoreplication according to their exact genealogy.

Usage

```
data(microbov)
```

Format

`microbov` is a genind object with 3 supplementary components:

coun a factor giving the country of each individual (AF: Afric; FR: France).

breed a factor giving the breed of each individual.

spe is a factor giving the species of each individual (BT: Bos taurus; BI: Bos indicus).

Source

Data prepared by Katayoun Moazami-Goudarzi and Denis Lalo\"e (INRA, Jouy-en-Josas, France)

References

Lalo\"e D., Jombart T., Dufour A.-B. and Moazami-Goudarzi K. (2007) Consensus genetic structuring and typological value of markers using Multiple Co-Inertia Analysis. *Genetics Selection Evolution*. **39**: 545–567.

Examples

```
data(microbov)
microbov
summary(microbov)

# make Y, a genpop object
Y <- genind2genpop(microbov)

# make allelic frequency table
temp <- makefreq(Y,missing="mean")
X <- temp$tab
nsamp <- temp$nobs

# perform 1 PCA per marker

if(require(ade4)){
kX <- ktab.data.frame(data.frame(X),Y@loc.nall)

kpca <- list()
for(i in 1:30) {kpca[[i]] <- dudi.pca(kX[[i]],scannf=FALSE,nf=2,center=TRUE,scale=FALSE)}
}

sel <- sample(1:30,4)
col = rep('red',15)
col[c(2,10)] = 'darkred'
col[c(4,12,14)] = 'deepskyblue4'
col[c(8,15)] = 'darkblue'

# display %PCA
par(mfrow=c(2,2))
for(i in sel) {
s.multinom(kpca[[i]]$c1,kX[[i]],n.sample=nsamp[,i],coulrow=col,sub=Y@loc.names[i])
add.scatter.eig(kpca[[i]]$eig,3,xax=1,yax=2,posi="top")
}

# perform a Multiple Coinertia Analysis
kXcent <- kX
for(i in 1:30) kXcent[[i]] <- as.data.frame(scalewt(kX[[i]],center=TRUE,scale=FALSE))
mcoa1 <- mcoa(kXcent,scannf=FALSE,nf=3, option="uniform")

# coordinated %PCA
mcoa.axes <- split(mcoa1$axis,Y@loc.fac)
mcoa.coord <- split(mcoa1$Tli,mcoa1$TL[,1])
var.coord <- lapply(mcoa.coord,function(e) apply(e,2,var))

par(mfrow=c(2,2))
for(i in sel) {
s.multinom(mcoa.axes[[i]][,1:2],kX[[i]],n.sample=nsamp[,i],coulrow=col,sub=Y@loc.names[i]
add.scatter.eig(var.coord[[i]],2,xax=1,yax=2,posi="top")
}
```

```
# reference typology
par(mfrow=c(1,1))
s.label(mcoa1$SynVar,lab=microbov@pop.names,sub="Reference typology",csub=1.5)
add.scatter.eig(mcoa1$pseudoeig,nf=3,xax=1,yax=2,posi="top")

# typologial values
tv <- mcoa1$cov2
tv <- apply(tv,2,function(c) c/sum(c))*100
rownames(tv) <- Y@loc.names
tv <- tv[order(Y@loc.names),]

par(mfrow=c(3,1),mar=c(5,3,3,4),las=3)
for(i in 1:3){
barplot(round(tv[,i],3),ylim=c(0,12),yaxt="n",main=paste("Typological value -
structure",i))
axis(side=2,at=seq(0,12,by=2),labels=paste(seq(0,12,by=2),"%"),cex=3)
abline(h=seq(0,12,by=2),col="grey",lty=2)
}
```

`monmonier` *Boundary detection using Monmonier algorithm*

Description

The Monmonier's algorithm detects boundaries among vertices of a valuated graph. This is achieved by finding the path exhibiting the largest distances between connected vertices.

The highest distance between two connected vertices (i.e. neighbours) is found, giving the starting point of the path. Then, the algorithm seeks the highest distance between immediate neighbours, and so on until a threshold value is attained. This threshold can be chosen from the plot of sorted distances between connected vertices: a boundary will likely result in an abrupt decrease of these values.

When several paths are looked for, the previous paths are taken into account, and cannot be either crossed or redrawn. Monmonier's algorithm can be used to assess the boundaries between patches of homogeneous observations.

Although Monmonier algorithm was initially designed for Voronoi tesselation, this implementation generalizes this algorithm to different connection networks. The `optimize.monmonier` function produces a `monmonier` object by trying several starting points, and returning the best boundary (i.e. largest sum of local distances). This is designed to avoid the algorithm to be trapped by a single strong local difference inside an homogeneous patch.

Usage

```
monmonier(xy, dist, cn, threshold=NULL, bd.length=NULL, nrun=1,
skip.local.diff=rep(0,nrun),scanthres=is.null(threshold), allowLoop=TRUE)

optimize.monmonier(xy, dist, cn, ntry=10, bd.length=NULL, return.best=TRUE,
display.graph=TRUE, threshold=NULL, scanthres=is.null(threshold), allowLoop=TRUE
```

```
## S3 method for class 'monmonier'
plot(x, variable=NULL,
displayed.runs=1:x$nrun, add.arrows=TRUE,
col='blue', lty=1, bwd=4, clegend=1, csize=0.7,
method=c('squaresize','greylevel'), sub='', csub=1, possub='topleft',
cneig=1, pixmap=NULL, contour=NULL, area=NULL, add.plot=FALSE, ...)

## S3 method for class 'monmonier'
print(x, ...)
```

Arguments

xy	a matrix yielding the spatial coordinates of the objects, with two columns respectively giving X and Y
dist	an object of class dist, giving the distances between the objects
cn	a connection network of class nb (package spdep)
threshold	a number giving the minimal distance between two neighbours crossed by the path; by default, this is the third quartile of all the distances between neighbours
bd.length	an optional integer giving the requested length of the boundaries (in number of local differences)
nrun	is a integer giving the number of runs of the algorithm, that is, the number of paths to search, being one by default
skip.local.diff	is a vector of integers, whose length is the number of paths (nrun); each integer gives the number of starting point to skip, to avoid being stuck in a local difference between two neighbours into an homogeneous patch; none are skipped by default
scanthres	a logical stating whether the threshold sould be chosen from the barplot of sorted distances between neighbours
allowLoop	a logical specifying whether the boundary can loop (TRUE, default) or not (FALSE)
ntry	an integer giving the number of different starting points tried.
return.best	a logical stating whether the best monmonier object should be returned (TRUE, default) or not (FALSE)
display.graph	a logical whether the scores of each try should be plotted (TRUE, default) or not
x	a monmonier object
variable	a variable to be plotted using s.value (package ade4)
displayed.runs	an integer vector giving the rank of the paths to represent
add.arrows	a logical, stating whether arrows should indicate the direction of the path (TRUE) or not (FALSE, used by default)
col	a characters vector giving the colors to be used for each boundary; recycled is needed; 'blue' is used by default
lty	a characters vector giving the type of line to be used for each boundary; 1 is used by default

bwd	a number giving the boundary width factor, applying to every segments of the paths; 4 is used by default
clegend	like in s.value, the size factor of the legend if a variable is represented
csize	like in s.value, the size factor of the squares used to represent a variable
method	like in s.value, a character giving the method to be used to represent the variable, either 'squaresize' (by default) or 'greylevel'
sub	a string of characters giving the subtitle of the plot
csub	the size factor of the subtitle
possub	the position of the subtitle; available choices are 'topleft' (by default), 'topright', 'bottomleft', and 'bottomright'
cneig	the size factor of the connection network
pixmap	an object of the class pixmap displayed in the map background
contour	a data frame with 4 columns to plot the contour of the map: each row gives a segment (x1,y1,x2,y2)
area	a data frame of class 'area' to plot a set of surface units in contour
add.plot	a logical stating whether the plot should be added to the current one (TRUE), or displayed in a new window (FALSE, by default)
...	further arguments passed to other methods

Details

The function monmonier returns a list of the class monmonier, which contains the general informations about the algorithm, and about each run. When displayed, the width of the boundaries reflects their 'strength'. Let a segment MN be part of the path, M being the middle of AB, N of CD. Then the boundary width for MN is proportionnal to (d(AB)+d(CD))/2.

As there is no perfect method to display graphically a quantitative variable (see for instance the differences between the two methods of s.value), the boundaries provided by this algorithm seem sometimes more reliable than the boundaries our eyes perceive (or miss).

Value

Returns an object of class monmonier, which contains the following elements :

run1 (run2, ...)	for each run, a list containing a dataframe giving the path coordinates, and a vector of the distances between neighbours of the path
nrun	the number of runs performed, i.e. the number of boundaries in the monmonier object
threshold	the threshold value, minimal distance between neighbours accounted for by the algorithm
xy	the matrix of spatial coordinates
cn	the connection network of class nb
call	the call of the function

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

References

Monmonier, M. (1973) Maximum-difference barriers: an alternative numerical regionalization method. *Geographic Analysis*, **3**, 245–261.

Manni, F., Guerard, E. and Heyer, E. (2004) Geographic patterns of (genetic, morphologic, linguistic) variation: how barriers can be detected by "Monmonier's algorithm". *Human Biology*, **76**, 173–190

See Also

spca,edit.nb

Examples

```
if(require(spdep) & require(ade4)){

### non-interactive example

# est-west separation
load(system.file("files/mondata1.rda",package="adegenet"))
cn1 <- chooseCN(mondata1$xy,type=2,ask=FALSE)
mon1 <- monmonier(mondata1$xy,dist(mondata1$x1),cn1,threshold=2)
plot(mon1,mondata1$x1)
plot(mon1,mondata1$x1,met="greylevel",add.arr=FALSE,col="red",bwd=6,lty=2)

# square in the middle
load(system.file("files/mondata2.rda",package="adegenet"))
cn2 <- chooseCN(mondata2$xy,type=1,ask=FALSE)
mon2 <- monmonier(mondata2$xy,dist(mondata2$x2),cn2,threshold=2)
plot(mon2,mondata2$x2,method="greylevel",add.arr=FALSE,bwd=6,col="red",csize=.5)

### genetic data example
## Not run:
data(sim2pop)

if(require(hierfstat)){
## try and find the Fst
fstat(sim2pop,fst=TRUE)
# Fst = 0.038
}

## run monmonier algorithm

# build connection network
gab <- chooseCN(sim2pop@other$xy,ask=FALSE,type=2)

# filter random noise
pca1 <- dudi.pca(sim2pop@tab,scale=FALSE, scannf=FALSE, nf=1)

# run the algorithm
mon1 <- monmonier(sim2pop@other$xy,dist(pca1$l1[,1]),gab,scanthres=FALSE)

# graphical display
plot(mon1,var=pca1$l1[,1])
temp <- sim2pop@pop
levels(temp) <- c(17,19)
temp <- as.numeric(as.character(temp))
```

```
plot(mon1)
points(sim2pop@other$xy,pch=temp,cex=2)
legend("topright",leg=c("Pop A", "Pop B"),pch=c(17,19))


### interactive example

# north-south separation
xy <- matrix(runif(120,0,10), ncol=2)
x1 <- rnorm(60)
x1[xy[,2] > 5] <- x1[xy[,2] > 5]+3
cn1 <- chooseCN(xy,type=1,ask=FALSE)
mon1 <- optimize.monmonier(xy,dist(x1)^2,cn1,ntry=10)

# graphics
plot(mon1,x1,met="greylevel",csize=.6)

# island in the middle
x2 <- rnorm(60)
sel <- (xy[,1]>3.5 & xy[,2]>3.5 & xy[,1]<6.5 & xy[,2]<6.5)
x2[sel] <- x2[sel]+4
cn2 <- chooseCN(xy,type=1,ask=FALSE)
mon2 <- optimize.monmonier(xy,dist(x2)^2,cn2,ntry=10)

# graphics
plot(mon2,x2,method="greylevel",add.arr=FALSE,bwd=6,col="red",csize=.5)

## End(Not run)
}
```

`na.replace-methods` *Replace missing values (NA) from an object*

Description

The generic function `na.replace` replaces NA in an object by appropriate values as defined by the argument `method`.

Methods are defined for genind and genpop objects.

Usage

```
## S4 method for signature 'genind'
na.replace(x,method, quiet=FALSE)
## S4 method for signature 'genpop'
na.replace(x,method, quiet=FALSE)
```

Arguments

`x`	a genind and genpop object
`method`	a character string: can be "0" or "mean" for genind objects, and "0" or "chi2" for genpop objects.
`quiet`	logical stating whether a message should be printed (TRUE,default) or not (FALSE).

Details

The argument "method" have the following effects:
- "0": missing values are set to "0". An entity (individual or population) that is not type on a locus has zeros for all alleles of that locus.

- "mean": missing values are set to the mean of the concerned allele, computed on all available observations (without distinction of population).

- "chi2": if a population is not typed for a marker, the corresponding count is set to that of a theoretical count in of a Chi-squared test. This is obtained by the product of the sums of both margins divided by the total number of alleles.

Value

A genind and genpop object without missing values.

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

Examples

```
data(nancycats)

obj1 <- genind2genpop(nancycats)
# note missing data in this summary
summary(obj1)

# NA are all in pop 17 and marker fca45
which(is.na(obj1$tab),TRUE)
truenames(obj1)[17,]

# replace missing values
obj2 <- na.replace(obj1,"chi2")
obj2$loc.names

# missing values where replaced
truenames(obj2)[,obj2$loc.fac=="L4"]
```

---

nancycats *Microsatellites genotypes of 237 cats from 17 colonies of Nancy (France)*

Description

This data set gives the genotypes of 237 cats (Felis catus L.) for 9 microsatellites markers. The individuals are divided into 17 colonies whose spatial coordinates are also provided.

Usage

```
data(nancycats)
```

Format

nancycats is a genind object with spatial coordinates of the colonies as a supplementary components (@xy). Beware: these coordinates are given for the true names (stored in @pop.names) and not for the generic names (used in @pop).

Source

Dominique Pontier (UMR CNRS 5558, University Lyon1, France)

References

Devillard, S.; Jombart, T. & Pontier, D. Disentangling spatial and genetic structure of stray cat (Felis catus L.) colonies in urban habitat using: not all colonies are equal. submitted to *Molecular Ecology*

Examples

```
data(nancycats)
nancycats

# summary's results are stored in x
x <- summary(nancycats)

# some useful graphics
barplot(x$loc.nall,ylab="Alleles numbers",main="Alleles numbers
per locus")

plot(x$pop.eff,x$pop.nall,type="n",xlab="Sample size",ylab="Number of alleles")
text(x$pop.eff,y=x$pop.nall,lab=names(x$pop.nall))

par(las=3)
barplot(table(nancycats@pop),ylab="Number of genotypes",main="Number of genotypes per col

# are cats structured among colonies ?
if(require(hierfstat)){

if(require(ade4)){
gtest <- gstat.randtest(nancycats,nsim=99)
gtest
plot(gtest)
}

dat <- genind2hierfstat(nancycats)

Fstat <- varcomp.glob(dat$pop,dat[,-1])
Fstat
}
```

---

old2new *Convert objects with obsolete classe into new objects*

Description

Adegenet classes changed from S3 to S4 types starting from version 1.1-0. `old2new` has two methods for genind and genpop objects, so that old adegenet objects can be retrieved and used in recent versions.

Usage

```
## S4 method for signature 'genind'
old2new(object)
## S4 method for signature 'genpop'
old2new(object)
```

Arguments

`object`	a genind or genpop object in S3 version, i.e. prior adegenet_1.1-0

Details

Optional content but `$pop` and `$pop.names` will not be converted. These are to be coerced into a list and set in the `@other` slot of the new object.

Author(s)

Thibaut Jombart `<t.jombart@imperial.ac.uk>`

`propShared` *Compute proportion of shared alleles*

Description

The function `propShared` computes the proportion of shared alleles in a set of genotypes (i.e. from a genind object). Current implementation works for haploid and diploid genotypes.

Usage

```
propShared(obj)
```

Arguments

`obj`	a genind object.

Details

Computations of the proportion of shared alleles are computed in C for diploid individuals, and in efficient R code for haploid genotypes. Proportions are computed from all available data, i.e. proportion can be computed as far as there is at least one typed locus in common between two genotypes.

Value

Returns a matrix of proportions

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

See Also

dist.genpop

Examples

```
## make a small object
data(microbov)
obj <- microbov[1:5,microbov@loc.fac %in% c("L01","L02")]

## verify results
propShared(obj)
genind2df(obj,sep="|")

## Use this similarity measure inside a PCoA
##  ! This is for illustration only !
## the distance should be rendered Euclidean before
## (e.g. using cailliez from package ade4).
if(require(ade4)){
matSimil <- propShared(microbov)
matDist <- exp(-matSimil)
D <- cailliez(as.dist(matDist))
pcoa1 <- dudi.pco(D,scannf=FALSE,nf=3)
s.class(pcoa1$li,microbov$pop,lab=microbov$pop.names)
}
```

---

propTyped-methods *Compute the proportion of typed elements*

Description

The generic function propTyped is devoted to investigating the structure of missing data in adegenet objects.

Methods are defined for genind and genpop objects. They can return the proportion of available (i.e. non-missing) data per individual/population, locus, or the combination of both in with case the matrix indicates which entity (individual or population) was typed on which locus.

Usage

```
## S4 method for signature 'genind'
propTyped(x,  by=c("ind","loc","both"))
## S4 method for signature 'genpop'
propTyped(x,  by=c("pop","loc","both"))
```

Arguments

x	a genind and genpop object
by	a character being "ind","loc", or "both" for genind object and "pop","loc", or "both" for genpop object. It specifies whether proportion of typed data are provided by entity ("ind"/"pop"), by locus ("loc") or both ("both"). See details.

Details

When by is set to "both", the result is a matrix of binary data with entities in rows (individuals or populations) and markers in columns. The values of the matrix are 1 for typed data, and 0 for NA.

Value

A vector of proportion (when by equals "ind", "pop", or "loc"), or a matrix of binary data (when by equals "both")

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

Examples

```
data(nancycats)
propTyped(nancycats,by="loc")
propTyped(genind2genpop(nancycats),by="both")
```

---

read.fstat *Reading data from Fstat*

Description

The function read.fstat reads Fstat data files (.dat) and convert them into a genind object.

Usage

```
read.fstat(file,missing=NA,quiet=FALSE)
```

Arguments

file	a character string giving the path to the file to convert, with the appropriate extension.
missing	can be NA, 0 or "mean". See details section.
quiet	logical stating whether a conversion message must be printed (TRUE,default) or not (FALSE).

Details

There are 3 treatments for missing values:
- NA: kept as NA.

- 0: allelic frequencies are set to 0 on all alleles of the concerned locus. Recommended for a PCA on compositionnal data.

- "mean": missing values are replaced by the mean frequency of the corresponding allele, computed on the whole set of individuals. Recommended for a centred PCA.

Value

an object of the class `genind`

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

References

Fstat (version 2.9.3). Software by Jerome Goudet. http://www2.unil.ch/popgen/softwares/fstat.htm

See Also

`import2genind`, `df2genind`, `read.genetix`, `read.structure`, `read.genepop`

Examples

```
obj <- read.fstat(system.file("files/nancycats.dat",package="adegenet"))
obj
```

`read.genepop` *Reading data from Genepop*

Description

The function `read.genepop` reads Genepop data files (.gen) and convert them into a genind object.

Usage

```
read.genepop(file,missing=NA,quiet=FALSE)
```

Arguments

`file`	a character string giving the path to the file to convert, with the appropriate extension.
`missing`	can be NA, 0 or "mean". See details section.
`quiet`	logical stating whether a conversion message must be printed (TRUE,default) or not (FALSE).

Details

There are 3 treatments for missing values:
- NA: kept as NA.

- 0: allelic frequencies are set to 0 on all alleles of the concerned locus. Recommended for a PCA on compositionnal data.

- "mean": missing values are replaced by the mean frequency of the corresponding allele, computed on the whole set of individuals. Recommended for a centred PCA.

Value

an object of the class `genind`

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

References

Raymond M. & Rousset F, (1995). GENEPOP (version 1.2): population genetics software for exact tests and ecumenicism. *J. Heredity*, **86**:248-249

See Also

`import2genind`, `df2genind`, `read.fstat`, `read.structure`, `read.genetix`

Examples

```
obj <- read.genepop(system.file("files/nancycats.gen",package="adegenet"))
obj
```

`read.genetix` *Reading data from GENETIX*

Description

The function `read.genetix` reads GENETIX data files (.gtx) and convert them into a genind object.

Usage

```
read.genetix(file=NULL,missing=NA,quiet=FALSE)
```

Arguments

file	a character string giving the path to the file to convert, with the appropriate extension.
missing	can be NA, 0 or "mean". See details section.
quiet	logical stating whether a conversion message must be printed (TRUE,default) or not (FALSE).

Details

There are 3 treatments for missing values:
- NA: kept as NA.

- 0: allelic frequencies are set to 0 on all alleles of the concerned locus. Recommended for a PCA on compositionnal data.

- "mean": missing values are replaced by the mean frequency of the corresponding allele, computed on the whole set of individuals. Recommended for a centred PCA.

Value

an object of the class genind

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

References

Belkhir K., Borsa P., Chikhi L., Raufaste N. & Bonhomme F. (1996-2004) GENETIX 4.05, logiciel sous Windows TM pour la genetique des populations. Laboratoire Genome, Populations, Interactions, CNRS UMR 5000, Université de Montpellier II, Montpellier (France).

See Also

import2genind, df2genind, read.fstat, read.structure, read.genepop

Examples

```
obj <- read.genetix(system.file("files/nancycats.gtx",package="adegenet"))
obj
```

`read.PLINK` *Reading PLINK Single Nucleotide Polymorphism data*

Description

The function `read.PLINK` reads a data file exported by the PLINK software with extension '.raw' and converts it into a genlight object. Optionally, information about SNPs can be read from a ".map" file, either by specifying the argument `map.file` in `read.PLINK`, or using `extract.PLINKmap` to add information to an existing genlight object.

The function reads data by chunks of several genomes (minimum 1, no maximum) at a time, which allows one to read massive datasets with negligible RAM requirements (albeit at a cost of computational time). The argument `chunkSize` indicates the number of genomes read at a time. Increasing this value decreases the computational time required to read data in, while increasing memory requirements.

See details for the documentation about how to export data using PLINK to the '.raw' format.

Usage

```
read.PLINK(file, map.file=NULL, quiet=FALSE, chunkSize=1000,
           multicore=require("multicore"), n.cores=NULL, ...)

extract.PLINKmap(file, x=NULL)
```

Arguments

`file`	for `read.PLINK` a character string giving the path to the file to convert, with the extension ".raw"; for `extract.PLINKmap`, a character string giving the path to a file with extension ".map".
`map.file`	an optional character string indicating the path to a ".map" file, which contains information about the SNPs (chromosome, position). If provided, this information is processed by `extract.PLINKmap` and stored in the `@other` slot.
`quiet`	logical stating whether a conversion messages should be printed (TRUE,default) or not (FALSE).
`chunkSize`	an integer indicating the number of genomes to be read at a time; larger values require more RAM but decrease the time needed to read the data.
`multicore`	a logical indicating whether multiple cores -if available- should be used for the computations (TRUE, default), or not (FALSE); requires the package `multicore` to be installed (see details).
`n.cores`	if `multicore` is TRUE, the number of cores to be used in the computations; if NULL, then the maximum number of cores available on the computer is used.
`...`	other arguments to be passed to other functions - currently not used.
`x`	an optional object of the class genlight, in which the information read is stored; if provided, information is matched against the names of the loci in `x`, as returned by `locNames(x)`; if not provided, a list of two components is returned, containing chromosome and position information.

Details

=== Exporting data from PLINK ===

Data need to be exported from PLINK using the option "–recodeA" (and NOT "–recodeAD"). The PLINK command should therefore look like: `plink --file data --recodeA`. For more information on this topic, please look at this webpage: http://pngu.mgh.harvard.edu/~purcell/plink/dataman.shtml

=== Using multiple cores ===

Most recent machines have one or several processors with multiple cores. R processes usually use one single core. The package `multicore` allows for parallelizing some computations on multiple cores, which decreases drastically computational time.

To use this functionality, you need to have the last version of the `multicore` package installed. To install it, type: install.packages("multicore",,"http://rforge.net/",type="source")

DO NOT use the version on CRAN, which is slightly outdated.

Value

- read.PLINK: an object of the class genlight

- extract.PLINKmap: if a genlight is provided as argument `x`, this object incorporating the new information about SNPs in the `@other` slot (with new components 'chromosome' and 'position'); otherwise, a list with two components containing chromosome and position information.

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

See Also

- `?genlight` for a description of the class genlight.

- `read.snp`: read SNPs in adegenet's '.snp' format.

- `fasta2genlight`: extract SNPs from alignments with fasta format.

- other import function in adegenet: `import2genind`, `df2genind`, `read.genetix` `read.fstat`, `read.structure`, `read.genepop`.

- another function `read.plink` is available in the package `snpMatrix`.

---

read.snp *Reading Single Nucleotide Polymorphism data*

Description

The function `read.snp` reads a SNP data file with extension '.snp' and converts it into a genlight object. This format is devoted to handle biallelic SNP only, but can accomodate massive datasets such as complete genomes with considerably less memory than other formats.

The function reads data by chunks of a few genomes (minimum 1, no maximum) at a time, which allows one to read massive datasets with negligible RAM requirements (albeit at a cost of computational time). The argument `chunkSize` indicates the number of genomes read at a time. Increasing this value decreases the computational time required to read data in, while increasing memory requirements.

A description of the .snp format is provided in an example file distributed with adegenet (see example below).

Usage

```
read.snp(file, quiet=FALSE, chunkSize = 1000, multicore = require("multicore"),
    n.cores = NULL, ...)
```

Arguments

file	a character string giving the path to the file to convert, with the extension ".snp".
quiet	logical stating whether a conversion messages should be printed (TRUE,default) or not (FALSE).
chunkSize	an integer indicating the number of genomes to be read at a time; larger values require more RAM but decrease the time needed to read the data.
multicore	a logical indicating whether multiple cores -if available- should be used for the computations (TRUE, default), or not (FALSE); requires the package multicore to be installed (see details).
n.cores	if multicore is TRUE, the number of cores to be used in the computations; if NULL, then the maximum number of cores available on the computer is used.
...	other arguments to be passed to other functions - currently not used.

Details

=== The .snp format ===

Details of the .snp format can be found in the example file distributed with adegenet (see below), or on the adegenet website (type adegenetWeb() in R).

=== Using multiple cores ===

Most recent machines have one or several processors with multiple cores. R processes usually use one single core. The package multicore allows for parallelizing some computations on multiple cores, which decreases drastically computational time.

To use this functionality, you need to have the last version of the multicore package installed. To install it, type: install.packages("multicore",,"http://rforge.net/",type="source")

DO NOT use the version on CRAN, which is slightly outdated.

Value

an object of the class genlight

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

See Also

- ?genlight for a description of the class genlight.
- read.PLINK: read SNPs in PLINK's '.raw' format.
- fasta2genlight: extract SNPs from alignments with fasta format.
- df2genind: convert any multiallelic markers into adegenet genind.
- import2genind: read multiallelic markers from various software into adegenet.

Examples

```
## show the example file ##
## this is the path to the file:
system.file("files/exampleSnpDat.snp",package="adegenet")

## show its content:
file.show(system.file("files/exampleSnpDat.snp",package="adegenet"))

## read the file
obj <-
read.snp(system.file("files/exampleSnpDat.snp",package="adegenet"), chunk=2)
obj
as.matrix(obj)
ploidy(obj)
alleles(obj)
locNames(obj)
```

---

read.structure *Reading data from STRUCTURE*

Description

The function `read.structure` reads STRUCTURE data files (.str ou .stru) and convert them into a genind object. By default, this function is interactive and asks a few questions about data content. This can be disabled (for optional questions) by turning the 'ask' argument to FALSE. However, one has to know the number of genotypes, of markers and if genotypes are coded on a single or on two rows before importing data.

Usage

```
read.structure(file, n.ind=NULL, n.loc=NULL,  onerowperind=NULL, col.lab=NULL, c
```

Arguments

`file`	a character string giving the path to the file to convert, with the appropriate extension.
`n.ind`	an integer giving the number of genotypes (or 'individuals') in the dataset
`n.loc`	an integer giving the number of markers in the dataset
`onerowperind`	a STRUCTURE coding option: are genotypes coded on a single row (TRUE), or on two rows (FALSE, default)
`col.lab`	an integer giving the index of the column containing labels of genotypes. '0' if absent.
`col.pop`	an integer giving the index of the column containing population to which genotypes belong. '0' if absent.
`col.others`	an vector of integers giving the indexes of the columns containing other informations to be read. Will be available in @other of the created object.
`row.marknames`	an integer giving the index of the row containing the names of the markers. '0' if absent.

NA.char	the character string coding missing data. "-9" by default. Note that in any case, series of zero (like "000") are interpreted as NA too.
pop	an optional factor giving the population of each individual.
ask	a logical specifying if the function should ask for optional informations about the dataset (TRUE, default), or try to be as quiet as possible (FALSE).
missing	can be NA, 0 or "mean". See details section.
quiet	logical stating whether a conversion message must be printed (TRUE,default) or not (FALSE).

Details

There are 3 treatments for missing values:
- NA: kept as NA.

- 0: allelic frequencies are set to 0 on all alleles of the concerned locus. Recommended for a PCA on compositionnal data.

- "mean": missing values are replaced by the mean frequency of the corresponding allele, computed on the whole set of individuals. Recommended for a centred PCA.

Value

an object of the class genind

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

References

Pritchard, J.; Stephens, M. & Donnelly, P. (2000) Inference of population structure using multilocus genotype data. *Genetics*, **155**: 945-959

See Also

import2genind, df2genind, read.fstat, read.genetix, read.genepop

Examples

```
obj <- read.structure(system.file("files/nancycats.str",package="adegenet"),
  onerowperind=FALSE, n.ind=237, n.loc=9, col.lab=1, col.pop=2, ask=FALSE)

obj
```

repool *Pool several genotypes into a single dataset*

Description

The function repool allows to merge genotypes from different genind objects into a single 'pool' (i.e. a new genind). The markers have to be the same for all objects to be merged, but there is no constraint on alleles.

This function can be useful, for instance, when hybrids are created using hybridize, to merge hybrids with their parent population for further analyses. Note that repool can also reverse the action of seppop.

Usage

```
repool(...)
```

Arguments

...	can be i) a list whose components are valid genind objects or, ii) several valid genind objects separated by commas.

Value

A genind object.

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

See Also

seploc, seppop

Examples

```
## use the cattle breeds dataset
data(microbov)
temp <- seppop(microbov)
names(temp)

## hybridize salers and zebu -- nasty cattle
zebler <- hybridize(temp$Salers, temp$Zebu, n=40)
zebler

## now merge zebler with other cattle breeds
nastyCattle <- repool(microbov, zebler)
nastyCattle
```

`rupica` *Microsatellites genotypes of 335 chamois (Rupicapra rupicapra) from the Bauges mountains (France)*

Description

This data set contains the genotypes of 335 chamois (*Rupicapra rupicapra*) from the Bauges mountains, in France. No prior clustering about individuals is known. Each genotype is georeferenced. These data also contain a raster map of elevation of the sampling area.

Usage

```
data(rupica)
```

Format

`rupica` is a genind object with 3 supplementary components inside the `@other` slot:

xy a matrix containing the spatial coordinates of the genotypes.

mnt a raster map of elevation, with the `asc` format from the `adehabitat` package.

showBauges a function to display the map of elevation with an appropriate legend (use `showBauges()`).

Source

Daniel Maillard, 'Office National de la Chasse et de la Faune Sauvage' (ONCFS), France.

References

Cassar S (2008) Organisation spatiale de la variabilité génétique et phénotypique à l'échelle du paysage: le cas du chamois et du chevreuil, en milieu de montagne. PhD Thesis. University Claude Bernard - Lyon 1, France.

Cassar S, Jombart T, Loison A, Pontier D, Dufour A-B, Jullien J-M, Chevrier T, Maillard D. Spatial genetic structure of Alpine chamois (*Rupicapra rupicapra*): a consequence of landscape features and social factors? submitted to *Molecular Ecology*.

Examples

```
if(require(ade4) & require(adehabitat) & require(spdep)){

data(rupica)
rupica

## see the sampling area
showBauges <- rupica$other$showBauges
showBauges()
points(rupica$other$xy,col="red")

## perform a sPCA
spca1 <- spca(rupica,type=5,d1=0,d2=2300,plot=FALSE,scannf=FALSE,nfposi=2,nfnega=0)
barplot(spca1$eig,col=rep(c("black","grey"),c(2,100)),main="sPCA eigenvalues")
screeplot(spca1,main="sPCA eigenvalues: decomposition")
```

```
## data visualization
showBauges(,labcex=1)
s.value(spca1$xy,spca1$ls[,1],add.p=TRUE,csize=.5)
add.scatter.eig(spca1$eig,1,1,1,posi="topleft",sub="Eigenvalues")

showBauges(,labcex=1)
s.value(spca1$xy,spca1$ls[,2],add.p=TRUE,csize=.5)
add.scatter.eig(spca1$eig,2,2,2,posi="topleft",sub="Eigenvalues")

rupica$other$showBauges()
colorplot(spca1$xy,spca1$li,cex=1.5,add.plot=TRUE)

## Not run:
## global and local tests
Gtest <- global.rtest(rupica@tab,spca1$lw,nperm=999)
Gtest
plot(Gtest)
Ltest <- local.rtest(rupica@tab,spca1$lw,nperm=999)
Ltest
plot(Ltest)

## End(Not run)
}
```

`scaleGen-methods` *Compute scaled allele frequencies*

Description

The generic function `scaleGen` is an analogue to the `scale` function, but is designed with further arguments giving scaling options.

Methods are defined for genind and genpop objects. Both return data.frames of scaled allele frequencies.

Usage

```
## S4 method for signature 'genind'
scaleGen(x, center=TRUE, scale=TRUE, method=c("sigma", "binom"), missing=c("NA",
## S4 method for signature 'genpop'
scaleGen(x, center=TRUE, scale=TRUE, method=c("sigma", "binom"), missing=c("NA",
```

Arguments

`x`	a genind and genpop object
`center`	a logical stating whether alleles frequencies should be centred to mean zero (default to TRUE). Alternatively, a vector of numeric values, one per allele, can be supplied: these values will be substracted from the allele frequencies.
`scale`	a logical stating whether alleles frequencies should be scaled (default to TRUE). Alternatively, a vector of numeric values, one per allele, can be supplied: these values will be substracted from the allele frequencies.

method	a character indicating the method to be used. See details.
truenames	a logical indicating whether true labels (as opposed to generic labels) should be used to name the output.
missing	a character giving the treatment for missing values. Can be "NA", "0" or "mean"

Details

The argument method is used as follows:

- sigma: scaling is made using the usual standard deviation

- binom: scaling is made using the theoretical variance of the allele frequency. This can be used to avoid that frequencies close to 0.5 have a stronger variance that those close to 0 or 1.

Value

A matrix of scaled allele frequencies with genotypes (genind) or populations in (genpop) in rows and alleles in columns.

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

Examples

```
## load data
data(microbov)
obj <- genind2genpop(microbov)

## compare different scaling
X1 <- scaleGen(obj)
X2 <- scaleGen(obj,met="bin")

if(require(ade4)){
## compute PCAs
pcaObj <- dudi.pca(obj,scale=FALSE,scannf=FALSE) # pca with no scaling
pcaX1 <- dudi.pca(X1,scale=FALSE,scannf=FALSE,nf=100) # pca with usual scaling
pcaX2 <- dudi.pca(X2,scale=FALSE,scannf=FALSE,nf=100) # pca with scaling for binomial var

## get the loadings of alleles for the two scalings
U1 <- pcaX1$c1
U2 <- pcaX2$c1


## find an optimal plane to compare loadings
## use a procustean rotation of loadings tables
pro1 <- procuste(U1,U2,nf=2)

## graphics
par(mfrow=c(2,2))
# eigenvalues
barplot(pcaObj$eig,main="Eigenvalues\n no scaling")
barplot(pcaX1$eig,main="Eigenvalues\n usual scaling")
barplot(pcaX2$eig,main="Eigenvalues\n 'binomial' scaling")
# differences between loadings of alleles
```

```
s.match(pro1$scor1,pro1$scor2,clab=0,sub="usual -> binom (procustean rotation)")
}
```

`selPopSize` *Select genotypes of well-represented populations*

Description

The function `selPopSize` checks the sample size of each population in a genind object and keeps only genotypes of populations having a given minimum size.

Usage

```
## S4 method for signature 'genind'
selPopSize(x,pop=NULL,nMin=10)
```

Arguments

`x`	a genind object
`pop`	a vector of characters or a factor giving the population of each genotype in 'x'. If not provided, seeked from x\$pop.
`nMin`	the minimum sample size for a population to be retained. Samples sizes strictly less than `nMin` will be discarded, those equal to or greater than `nMin` are kept.

Value

A genind object.

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

See Also

`seploc`, `repool`

Examples

```
data(microbov)

table(pop(microbov))
obj <- selPopSize(microbov, n=50)

obj
table(pop(obj))
```

seploc *Separate data per locus*

Description

The function seploc splits an object (genind, genpop or genlight) by marker. For genind and genpop objects, the method returns a list of objects whose components each correspond to a marker. For genlight objects, the methods returns blocks of SNPs.

Usage

```
## S4 method for signature 'genind'
seploc(x,truenames=TRUE,res.type=c("genind","matrix"))
## S4 method for signature 'genpop'
seploc(x,truenames=TRUE,res.type=c("genpop","matrix"))
## S4 method for signature 'genlight'
seploc(x, n.block=NULL, block.size=NULL, random=FALSE,
       multicore=require(multicore), n.cores=NULL)
```

Arguments

x	a genind or a genpop object.
truenames	a logical indicating whether true names should be used (TRUE, default) instead of generic labels (FALSE).
res.type	a character indicating the type of returned results, a genind or genpop object (default) or a matrix of data corresponding to the 'tab' slot.
n.block	an integer indicating the number of blocks of SNPs to be returned.
block.size	an integer indicating the size (in number of SNPs) of the blocks to be returned.
random	should blocks be formed of contiguous SNPs, or should they be made or randomly chosen SNPs.
multicore	a logical indicating whether multiple cores -if available- should be used for the computations (TRUE, default), or not (FALSE); requires the package multicore to be installed.
n.cores	if multicore is TRUE, the number of cores to be used in the computations; if NULL, then the maximum number of cores available on the computer is used.

Value

The function seploc returns an list of objects of the same class as the initial object, or a list of matrices similar to x\$tab.

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

See Also

seppop, repool

Examples

```
## example on genind objects
data(microbov)

# separate all markers
obj <- seploc(microbov)
names(obj)

obj$INRA5


## example on genlight objects
x <- glSim(100, 1000, 0, ploidy=2) # simulate data
x <- x[,order(glSum(x))] # reorder loci by frequency of 2nd allele
glPlot(x, main="All data") # plot data
foo <- seploc(x, n.block=3) # form 3 blocks
foo
glPlot(foo[[1]], main="1st block") # plot 1st block
glPlot(foo[[2]], main="2nd block") # plot 2nd block
glPlot(foo[[3]], main="3rd block") # plot 3rd block

foo <- seploc(x, block.size=600, random=TRUE) # split data, randomize loci
foo # note the different block sizes
glPlot(foo[[1]])
```

---

seppop *Separate genotypes per population*

Description

The function seppop splits a genind or a genlight object by population, returning a list of objects whose components each correspond to a population.

For genind objects, the output can either be a list of genind (default), or a list of matrices corresponding to the @tab slot.

Usage

```
## S4 method for signature 'genind'
seppop(x,pop=NULL,truenames=TRUE,res.type=c("genind","matrix"),
  drop=FALSE, treatOther=TRUE, quiet=TRUE)

## S4 method for signature 'genlight'
seppop(x,pop=NULL, treatOther=TRUE, quiet=TRUE)
```

Arguments

x	a genind object
pop	a factor giving the population of each genotype in 'x'. If not provided, seeked in x\$pop.
truenames	a logical indicating whether true names should be used (TRUE, default) instead of generic labels (FALSE); used if res.type is "matrix".

res.type	a character indicating the type of returned results, a list of genind object (default) or a matrix of data corresponding to the 'tab' slots.
drop	a logical stating whether alleles that are no longer present in a subset of data should be discarded (TRUE) or kept anyway (FALSE, default).
treatOther	a logical stating whether elements of the @other slot should be treated as well (TRUE), or not (FALSE). See details in accessor documentations (pop).
quiet	a logical indicating whether warnings should be issued when trying to subset components of the @other slot (TRUE), or not (FALSE, default).

Value

According to 'res.type': a list of genind object (default) or a matrix of data corresponding to the 'tab' slots.

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

See Also

seploc, repool

Examples

```
data(microbov)

obj <- seppop(microbov)
names(obj)

obj$Salers


#### example for genlight objects ####
x <- new("genlight", list(a=rep(1,1e3),b=rep(0,1e3),c=rep(1, 1e3)))
x

pop(x) # no population info
pop(x) <- c("pop1","pop2", "pop1") # set population memberships
pop(x)
seppop(x)
as.matrix(seppop(x)$pop1)[,1:20]
as.matrix(seppop(x)$pop2)[,1:20,drop=FALSE]
```

---

seqTrack *SeqTrack algorithm for reconstructing genealogies*

Description

The SeqTrack algorithm [1] aims at reconstructing genealogies of sampled haplotypes or genotypes for which a collection date is available. Contrary to phylogenetic methods which aims at reconstructing hypothetical ancestors for observed sequences, SeqTrack considers that ancestors and descendents are sampled together, and therefore infers ancestry relationships among the sampled sequences.

This approach proved more efficient than phylogenetic approaches for reconstructing transmission trees in densely sampled disease outbreaks [1]. This implementation defines a generic function `seqTrack` with methods for specific object classes.

Usage

```
seqTrack(...)

## S3 method for class 'matrix'
seqTrack(x, x.names, x.dates, best = c("min", "max"),
    prox.mat = NULL, mu = NULL, haplo.length = NULL, ...)

plotSeqTrack(x, xy, use.arrows=TRUE, annot=TRUE, labels=NULL, col=NULL,
                          bg="grey", add=FALSE, quiet=FALSE,
                          date.range=NULL, jitter.arrows=0, plot=TRUE, ...)

get.likelihood(...)

## S3 method for class 'seqTrack'
get.likelihood(x, mu, haplo.length, ...)
```

Arguments

`x`	for seqTrack, a matrix giving weights to pairs of ancestries such that x[i,j] is the weight of 'i ancestor of j'. For plotSeqTrack and get.likelihood. seqTrack, a `seqTrack` object.
`x.names`	a character vector giving the labels of the haplotypes/genotypes
`x.dates`	a vector of collection dates for the sampled haplotypes/genotypes. Dates must have the POSIXct format. See `details` or `?as.POSIXct` for more information.
`best`	a character string matching 'min' or 'max', indicating whether genealogies should minimize or maximize the sum of weights of ancestries.
`prox.mat`	an optional matrix of proximities between haplotypes/genotypes used to resolve ties in the choice of ancestors, by picking up the 'closest' ancestor amongst possible ancestors, in the sense of `prox.mat`. `prox.mat[i, j]` must indicate a proximity for the relationship 'i ancestor to j'. For instance, if `prox.mat` contains spatial proximities, then `prox.mat[i, j]` gives a measure of how easy it is to migrate from location 'i' to 'j'.
`mu`	(optional) a mutation rate, per site and per day. When 'x' contains numbers of mutations, used to resolve ties using a maximum likelihood approach (requires `haplo.length` to be provided).
`haplo.length`	(optional) the length of analysed sequences in number of nucleotides. When 'x' contains numbers of mutations, used to resolve ties using a maximum likelihood approach (requires `mu` to be provided).

xy	spatial coordinates of the sampled haplotypes/genotypes.
use.arrows	a logical indicating whether arrows should be used to represented ancestries (pointing from ancestor to descendent, TRUE), or whether segments shall be used (FALSE).
annot	a logical indicating whether arrows or segments representing ancestries should be annotated (TRUE) or not (FALSE).
labels	a character vector containing annotations of the ancestries. If left empty, ancestries are annotated by the descendent.
col	a vector of colors to be used for plotting ancestries.
bg	a color to be used as background.
add	a logical stating whether the plot should be added to current figure (TRUE), or drawn as a new plot (FALSE, default).
quiet	a logical stating whether messages other than errors should be displayed (FALSE, default), or hidden (TRUE).
date.range	a vector of length two with POSIXct format indicating the time window for which ancestries should be displayed.
jitter.arrows	a positive number indicating the amount of noise to be added to coordinates of arrows; useful when several arrows overlap. See jitter.
plot	a logical stating whether a plot should be drawn (TRUE, default), or not (FALSE). In all cases, the function invisibly returns plotting information.
...	further arguments to be passed to other methods

Details

=== Maximum parsimony genealogies ===
Maximum parsimony genealogies can be obtained easily using this implementation of seqTrack. One has to provide in x a matrix of genetic distances. The most straightforward distance is the number of differing nucleotides. See dist.dna in the ape package for a wide range of genetic distances between aligned sequences. The argument best should be set to "min" (its default value), so that the identified genealogy minimizes the total number of mutations. If x contains number of mutations, then mu and haplo.length should also be provided for resolving ties in equally parsimonious ancestors using maximum likelihood.

=== Likelihood of observed genetic differentiation ===
The probability of oberving a given number of mutations between a sequence and its ancestor can be computed using get.likelihood.seqTrack. Note that this is only possible if x contained number of mutations.

=== Converting seqTrack objects to graphs ===
seqTrack objects can be converted to graphNEL-class objects, which can in turn be plotted and manipulated using classical graph tools. Simply use 'as(x, "graphNEL")' where 'x' is a seqTrack object. This functionality requires the graph package. Note that this is to be installed from Bioconductor, likely using the following command lines:
source("http://bioconductor.org/biocLite.R")
biocLite("graph")

Also note that the R package Rgraphviz (also on Bioconductor) provides nice ways of plotting graphs (replace 'graph' with 'Rgraphviz' in the previous command lines to install this package).

Value

=== output of seqTrack ===
seqTrack function returns data.frame with the class `seqTrack`, in which each row is an inferred ancestry described by the following columns: - id: indices identifying haplotypes/genotypes
- ances: index of the inferred ancestor
- weight: weight of the inferred ancestries
- date: date of the haplotype/genotype
- ances.date: date of the ancestor

=== output of plotSeqTrack ===
This graphical function invisibly returns the coordinates of the arrows/segments drawn and their colors, as a data.frame.

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

References

Jombart T, Eggo R, Dodd P, Balloux F (2010) Reconstructing disease outbreaks from genetic data: a graph approach. Heredity. doi: 10.1038/hdy.2010.78.

See Also

`dist.dna` in the ape package to compute pairwise genetic distances in aligned sequences.

Examples

```
if(require(ape)){
## ANALYSIS OF SIMULATED DATA ##
## SIMULATE A GENEALOGY
dat <- haploGen(seq.l=1e4, repro=function(){sample(1:4,1)}, gen.time=1, t.max=3)


## SEQTRACK ANALYSIS
res <- seqTrack(dat, mu=0.0001, haplo.length=1e4)


## PROPORTION OF CORRECT RECONSTRUCTION
mean(dat$ances==res$ances,na.rm=TRUE)


## PLOT RESULTS
if(require(graph) && require(Rgraphviz)){
dat.g <- as(dat, "graphNEL")
res.g <- as(res, "graphNEL")


## ORIGINAL DATA
dat.annot <- as.character(unlist(edgeWeights(dat.g)))
names(dat.annot) <- edgeNames(dat.g)
renderGraph(layoutGraph(dat.g, edgeAttrs = list(label = dat.annot)))


## SEQTRACK RESULTS
res.annot <- as.character(unlist(edgeWeights(res.g)))
```

```
names(res.annot) <- edgeNames(res.g)
renderGraph(layoutGraph(res.g, edgeAttrs = list(label = res.annot)))
}



## ANALYSIS OF PANDEMIC A/H1N1 INFLUENZA DATA ##
dat <- read.csv(system.file("files/pdH1N1-data.csv",package="adegenet"))
ha <-  read.dna(system.file("files/pdH1N1-HA.fasta",package="adegenet"), format="fa")
na <- read.dna(system.file("files/pdH1N1-NA.fasta",package="adegenet"), format="fa")


## COMPUTE NUCLEOTIDIC DISTANCES
nbNucl <- ncol(as.matrix(ha)) + ncol(as.matrix(na))
D <- dist.dna(ha,model="raw")*ncol(as.matrix(ha)) + dist.dna(na,model="raw")*ncol(as.matr
D <- round(as.matrix(D))


## MATRIX OF SPATIAL CONNECTIVITY
## (to promote local transmissions)
xy <- cbind(dat$lon, dat$lat)
temp <- as.matrix(dist(xy))
M <- 1* (temp < 1e-10)


## SEQTRACK ANALYSIS
dat$date <- as.POSIXct(dat$date)
res <- seqTrack(D, rownames(dat), dat$date, prox.mat=M, mu=.00502/365, haplo.le=nbNucl)


## COMPUTE GENETIC LIKELIHOOD
p <- get.likelihood(res, mu=.00502/365, haplo.length=nbNucl)
# (these could be shown as colors when plotting results)
# (but mutations will be used instead)


## EXAMINE RESULTS
head(res)
tail(res)
range(res$weight, na.rm=TRUE)
barplot(table(res$weight)/sum(!is.na(res$weight)), ylab="Frequency",xlab="Mutations betwe


## DISPLAY SPATIO-TEMPORAL DYNAMICS
if(require(maps)){
myDates <- as.integer(difftime(dat$date, as.POSIXct("2009-01-21"), unit="day"))
myMonth <- as.POSIXct(c("2009-02-01", "2009-03-01","2009-04-01","2009-05-01","2009-06-01"
x.month <-  as.integer(difftime(myMonth, as.POSIXct("2009-01-21"), unit="day"))


## FIRST STAGE:
## SPREAD TO THE USA AND CANADA
curRange <- as.POSIXct(c("2009-03-29","2009-04-25"))
par(bg="deepskyblue")
map("world", fill=TRUE, col="grey")
opal <- palette()
```

```
palette(rev(heat.colors(10)))
plotSeqTrack(res, round(xy), add=TRUE,annot=FALSE,lwd=2, date.range=curRange, col=res$wei
title(paste(curRange, collapse=" to "))
legend("bottom", lty=1, leg=0:8, title="number of mutations", col=1:9, lwd=2, horiz=TRUE)


## SECOND STAGE:
## SPREAD WITHIN AMERICA, FIRST SEEDING OUTSIDE AMERICA
curRange <- as.POSIXct(c("2009-04-30","2009-05-07"))
par(bg="deepskyblue")
map("world", fill=TRUE, col="grey")
opal <- palette()
palette(rev(heat.colors(10)))
plotSeqTrack(res, round(xy), add=TRUE,annot=FALSE,lwd=2, date.range=curRange, col=res$wei
title(paste(curRange, collapse=" to "))
legend("bottom", lty=1, leg=0:8, title="number of mutations", col=1:9,lwd=2, horiz=TRUE)


## THIRD STAGE:
## PANDEMIC
curRange <- as.POSIXct(c("2009-05-15","2009-05-25"))
par(bg="deepskyblue")
map("world", fill=TRUE, col="grey")
opal <- palette()
palette(rev(heat.colors(10)))
plotSeqTrack(res, round(xy), add=TRUE,annot=FALSE,lwd=2, date.range=curRange, col=res$wei
title(paste(curRange, collapse=" to "))
legend("bottom", lty=1, leg=0:8, title="number of mutations", col=1:9,lwd=2, horiz=TRUE)

}
}
```

`SequencesToGenind` *Importing data from an alignement of sequences to a genind object*

Description

These functions take an alignement of sequences and translate SNPs into a genind object. Note that only polymorphic loci are retained.

Currently, accepted sequence formats are:
- DNAbin (ape package): function DNAbin2genind
- alignment (seqinr package): function alignment2genind

Usage

```
DNAbin2genind(x, pop=NULL, exp.char=c("a","t","g","c"), na.char=NULL,
                          polyThres=1/100)

alignment2genind(x, pop=NULL, exp.char=c("a","t","g","c"), na.char="-", polyThre
```

Arguments

x	an object containing aligned sequences.
pop	an optional factor giving the population to which each sequence belongs.
exp.char	a vector of single character providing expected values; all other characters will be turned to NA.
na.char	a vector of single characters providing values that should be considered as NA. If not NULL, this is used instead of exp.char.
polyThres	the minimum frequency of a minor allele for a locus to be considered as polymorphic (defaults to 0.01).

Value

an object of the class genind

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

See Also

import2genind, read.genetix, read.fstat, read.structure, read.genepop, DNAbin, as.alignment.

Examples

```
if(require(ape)){
data(woodmouse)
x <- DNAbin2genind(woodmouse)
x
genind2df(x)
}

if(require(seqinr)){
mase.res   <- read.alignment(file = system.file("sequences/test.mase",package = "seqinr")
mase.res
x <- alignment2genind(mase.res)
x
locNames(x) # list of polymorphic sites
genind2df(x)

## look at Euclidean distances
D <- dist(truenames(x))
D

if(require(ade4)){
## summarise with a PCoA
pco1 <- dudi.pco(D, scannf=FALSE,nf=2)
scatter(pco1, posi="bottomright")
title("Principal Coordinate Analysis\n-based on proteic distances-")
}
}
```

`sim2pop` *Simulated genotypes of two georeferenced populations*

Description

This simple data set was obtained by sampling two populations evolving in a island model, simulated using Easypop (2.0.1). See `source` for simulation details. Sample sizes were respectively 100 and 30 genotypes. The genotypes were given spatial coordinates so that both populations were spatially differentiated.

Usage

```
data(sim2pop)
```

Format

`sim2pop` is a genind object with a matrix of xy coordinates as supplementary component.

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

Source

Easypop version 2.0.1 was run with the following parameters:
- two diploid populations, one sex, random mating
- 1000 individuals per population
- proportion of migration: 0.002
- 20 loci
- mutation rate: 0.0001 (KAM model)
- maximum of 50 allelic states
- 1000 generations (last one taken)

References

Balloux F (2001) Easypop (version 1.7): a computer program for oppulation genetics simulations *Journal of Heredity*, **92**: 301-302

Examples

```
## Not run:
data(sim2pop)

if(require(hierfstat)){
## try and find the Fst
temp <- genind2hierfstat(sim2pop)
varcomp.glob(temp[,1],temp[,-1])
# Fst = 0.038
}

## run monmonier algorithm
```

```
# build connection network
gab <- chooseCN(sim2pop@other$xy,ask=FALSE,type=2)

# filter random noise
pca1 <- dudi.pca(sim2pop@tab,scale=FALSE, scannf=FALSE, nf=1)

# run the algorithm
mon1 <- monmonier(sim2pop@other$xy,dist(pca1$l1[,1]),gab,scanthres=FALSE)

# graphical display
temp <- sim2pop@pop
levels(temp) <- c(17,19)
temp <- as.numeric(as.character(temp))
plot(mon1)
points(sim2pop@other$xy,pch=temp,cex=2)
legend("topright",leg=c("Pop A", "Pop B"),pch=c(17,19))

## End(Not run)
```

SNPbin-class *Formal class "SNPbin"*

Description

The class SNPbin is a formal (S4) class for storing a genotype of binary SNPs in a compact way, using a bit-level coding scheme. This storage is most efficient with haploid data, where the memory taken to represent data can reduced more than 50 times. However, SNPbin can be used for any level of ploidy, and still remain an efficient storage mode.

A SNPbin object can be constructed from a vector of integers giving the number of the second allele for each locus.

SNPbin stores a single genotype. To store multiple genotypes, use the genlight class.

Objects from the class SNPbin

SNPbin objects can be created by calls to new("SNPbin", ...), where '...' can be the following arguments:

snp a vector of integers or numeric giving numbers of copies of the second alleles for each locus. If only one unnamed argument is provided to 'new', it is considered as this one.

ploidy an integer indicating the ploidy of the genotype; if not provided, will be guessed from the data (as the maximum from the 'snp' input vector).

label an optional character string serving as a label for the genotype.

Slots

The following slots are the content of instances of the class SNPbin; note that in most cases, it is better to retrieve information via accessors (see below), rather than by accessing the slots manually.

snp: a list of vectors with the class raw.

n.loc: an integer indicating the number of SNPs of the genotype.

NA.posi: a vector of integer giving the position of missing data.

label: an optional character string serving as a label for the genotype..

ploidy: an integer indicating the ploidy of the genotype.

Methods

Here is a list of methods available for `SNPbin` objects. Most of these methods are accessors, that is, functions which are used to retrieve the content of the object. Specific manpages can exist for accessors with more than one argument. These are indicated by a '*' symbol next to the method's name. This list also contains methods for conversion from `SNPbin` to other classes.

[`signature(x = "SNPbin")`: usual method to subset objects in R. The argument indicates how SNPs are to be subsetted. It can be a vector of signed integers or of logicals.

show `signature(x = "SNPbin")`: printing of the object.

$ `signature(x = "SNPbin")`: similar to the @ operator; used to access the content of slots of the object.

$<- `signature(x = "SNPbin")`: similar to the @ operator; used to replace the content of slots of the object.

nLoc `signature(x = "SNPbin")`: returns the number of SNPs in the object.

names `signature(x = "SNPbin")`: returns the names of the slots of the object.

ploidy `signature(x = "SNPbin")`: returns the ploidy of the genotype.

as.integer `signature(x = "SNPbin")`: converts a `SNPbin` object to a vector of integers. The S4 method 'as' can be used as well (e.g. as(x, "integer")).

cbind `signature(x = "SNPbin")`: merges genotyping of the same individual at different SNPs (all stored as SNPbin objects) into a single SNPbin.

c `signature(x = "SNPbin")`: same as cbind.SNPbin.

Author(s)

Thibaut Jombart (<t.jombart@imperial.ac.uk>)

See Also

Related class:
- `genlight`, for storing multiple binary SNP genotypes.
- `genind`, for storing other types of genetic markers.

Examples

```
#### HAPLOID EXAMPLE ####
## create a genotype of 1,000,000 SNPs
dat <- sample(c(0,1,NA), 1e6, prob=c(.495, .495, .01), replace=TRUE)
dat[1:10]
x <- new("SNPbin", dat)
x
x[1:10] # subsetting
as.integer(x[1:10])

## try a few accessors
ploidy(x)
nLoc(x)
head(x$snp[[1]]) # internal bit-level coding

## check that conversion is OK
identical(as(x, "integer"),as.integer(dat)) # SHOULD BE TRUE
```

```
## compare the size of the objects
print(object.size(dat), unit="auto")
print(object.size(x), unit="auto")
object.size(dat)/object.size(x) # EFFICIENCY OF CONVERSION


#### TETRAPLOID EXAMPLE ####
## create a genotype of 1,000,000 SNPs
dat <- sample(c(0:4,NA), 1e6, prob=c(rep(.995/5,5), 0.005), replace=TRUE)
x <- new("SNPbin", dat)
identical(as(x, "integer"),as.integer(dat)) # MUST BE TRUE

## compare the size of the objects
print(object.size(dat), unit="auto")
print(object.size(x), unit="auto")
object.size(dat)/object.size(x) # EFFICIENCY OF CONVERSION


#### c, cbind ####
a <- new("SNPbin", c(1,1,1,1,1))
b <- new("SNPbin", c(0,0,0,0,0))
a
b
ab <- c(a,b)
ab
identical(c(a,b),cbind(a,b))
as.integer(ab)
```

`spca` *Spatial principal component analysis*

Description

These functions are designed to perform a spatial principal component analysis and to display the results. They call upon `multispati` from the `ade4` package.

`spca` performs the spatial component analysis. Other functions are:

- `print.spca`: prints the spca content

- `summary.spca`: gives variance and autocorrelation statistics

- `plot.spca`: usefull graphics (connection network, 3 different representations of map of scores, eigenvalues barplot and decomposition)

- `screeplot.spca`: decomposes spca eigenvalues into variance and autocorrelation

- `colorplot.spca`: represents principal components of sPCA in space using the RGB system.

A tutorial describes how to perform a sPCA: see http://adegenet.r-forge.r-project.org/files/tutorial-spca.pdf or type `adegenetTutorial(which="spca")`.

Usage

```
spca(obj, xy=NULL, cn=NULL, matWeight=NULL,
     scale=FALSE, scale.method=c("sigma","binom"),
     scannf=TRUE, nfposi=1, nfnega=1,
     type=NULL, ask=TRUE, plot.nb=TRUE, edit.nb=FALSE,
     truenames=TRUE, d1=NULL, d2=NULL, k=NULL, a=NULL, dmin=NULL)

## S3 method for class 'spca'
print(x, ...)

## S3 method for class 'spca'
summary(object, ..., printres=TRUE)

## S3 method for class 'spca'
plot(x, axis = 1, useLag=FALSE, ...)

## S3 method for class 'spca'
screeplot(x, ..., main=NULL)

## S3 method for class 'spca'
colorplot(x, axes=1:ncol(x$li), useLag=FALSE, ...)
```

Arguments

obj	a genind or genpop object.
xy	a matrix or data.frame with two columns for x and y coordinates. Seeked from obj\$other\$xy if it exists when xy is not provided. Can be NULL if a nb object is provided in cn. Longitude/latitude coordinates should be converted first by a given projection (see 'See Also' section).
cn	a connection network of the class 'nb' (package spdep). Can be NULL if xy is provided. Can be easily obtained using the function chooseCN (see details).
matWeight	a square matrix of spatial weights, indicating the spatial proximities between entities. If provided, this argument prevails over cn (see details).
scale	a logical indicating whether alleles should be scaled to unit variance (TRUE) or not (FALSE, default).
scale.method	a character string indicating the method used for scaling allele frequencies. This argument is passed to scaleGen function (see ?scaleGen).
scannf	a logical stating whether eigenvalues should be chosen interactively (TRUE, default) or not (FALSE).
nfposi	an integer giving the number of positive eigenvalues retained ('global structures').
nfnega	an integer giving the number of negative eigenvalues retained ('local structures').
type	an integer giving the type of graph (see details in chooseCN help page). If provided, ask is set to FALSE.
ask	a logical stating whether graph should be chosen interactively (TRUE,default) or not (FALSE).

plot.nb	a logical stating whether the resulting graph should be plotted (TRUE, default) or not (FALSE).
edit.nb	a logical stating whether the resulting graph should be edited manually for corrections (TRUE) or not (FALSE, default).
truenames	a logical stating whether true names should be used for 'obj' (TRUE, default) instead of generic labels (FALSE)
d1	the minimum distance between any two neighbours. Used if type=5.
d2	the maximum distance between any two neighbours. Used if type=5.
k	the number of neighbours per point. Used if type=6.
a	the exponent of the inverse distance matrix. Used if type=7.
dmin	the minimum distance between any two distinct points. Used to avoid infinite spatial proximities (defined as the inversed spatial distances). Used if type=7.
x	a spca object.
object	a spca object.
printres	a logical stating whether results should be printed on the screen (TRUE, default) or not (FALSE).
axis	an integer between 1 and (nfposi+nfnega) indicating which axis should be plotted.
main	a title for the screeplot; if NULL, a default one is used.
...	further arguments passed to other methods.
axes	the index of the columns of X to be represented. Up to three axes can be chosen.
useLag	a logical stating whether the lagged components (x$ls) should be used instead of the components (x$li).

Details

The spatial principal component analysis (sPCA) is designed to investigate spatial patterns in the genetic variability. Given multilocus genotypes (individual level) or allelic frequency (population level) and spatial coordinates, it finds individuals (or population) scores maximizing the product of variance and spatial autocorrelation (Moran's I). Large positive and negative eigenvalues correspond to global and local structures.

Spatial weights can be obtained in several ways, depending how the arguments xy, cn, and matWeight are set.
When several acceptable ways are used at the same time, priority is as follows:
matWeight > cn > xy

Value

The class spca are given to lists with the following components:

eig	a numeric vector of eigenvalues.
nfposi	an integer giving the number of global structures retained.
nfnega	an integer giving the number of local structures retained.
c1	a data.frame of alleles loadings for each axis.

li	a data.frame of row (individuals or populations) coordinates onto the sPCA axes.
ls	a data.frame of lag vectors of the row coordinates; useful to clarify maps of global scores .
as	a data.frame giving the coordinates of the PCA axes onto the sPCA axes.
call	the matched call.
xy	a matrix of spatial coordinates.
lw	a list of spatial weights of class listw.

Other functions have different outputs:
- summary.spca returns a list with 3 components: Istat giving the null, minimum and maximum Moran's I values; pca gives variance and I statistics for the principal component analysis; spca gives variance and I statistics for the sPCA.

- plot.spca returns the matched call.

- screeplot.spca returns the matched call.

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

References

Jombart, T., Devillard, S., Dufour, A.-B. and Pontier, D. Revealing cryptic spatial patterns in genetic variability by a new multivariate method. *Heredity*, **101**, 92–103.

Wartenberg, D. E. (1985) Multivariate spatial correlation: a method for exploratory geographical analysis. *Geographical Analysis*, **17**, 263–283.

Moran, P.A.P. (1948) The interpretation of statistical maps. *Journal of the Royal Statistical Society, B* **10**, 243–251.

Moran, P.A.P. (1950) Notes on continuous stochastic phenomena. *Biometrika*, **37**, 17–23.

de Jong, P. and Sprenger, C. and van Veen, F. (1984) On extreme values of Moran's I and Geary's c. *Geographical Analysis*, **16**, 17–24.

See Also

spcaIllus, a set of simulated data illustrating the sPCA
global.rtest and local.rtest
chooseCN, multispati, multispati.randtest
convUL, from the package 'PBSmapping' to convert longitude/latitude to UTM coordinates.

Examples

```
## data(spcaIllus) illustrates the sPCA
## see ?spcaIllus
##

example(spcaIllus)
```

spcaIllus *Simulated data illustrating the sPCA*

Description

Datasets illustrating the spatial Principal Component Analysis (Jombart et al. 2009). These data were simulated using various models using Easypop (2.0.1). Spatial coordinates were defined so that different spatial patterns existed in the data. The spca-illus is a list containing the following genind or genpop objects:
- dat2A: 2 patches
- dat2B: cline between two pop
- dat2C: repulsion among individuals from the same gene pool
- dat3: cline and repulsion
- dat4: patches and local alternance

See "source" for a reference providing simulation details.

Usage

```
data(spcaIllus)
```

Format

spcaIllus is list of 5 components being either genind or genpop objects.

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

Source

Jombart, T., Devillard, S., Dufour, A.-B. and Pontier, D. Revealing cryptic spatial patterns in genetic variability by a new multivariate method. *Heredity*, **101**, 92–103.

References

Jombart, T., Devillard, S., Dufour, A.-B. and Pontier, D. Revealing cryptic spatial patterns in genetic variability by a new multivariate method. *Heredity*, **101**, 92–103.

Balloux F (2001) Easypop (version 1.7): a computer program for oppulation genetics simulations *Journal of Heredity*, **92**: 301-302

See Also

spca

Examples

```
if(require(spdep) & require(ade4)){

data(spcaIllus)
attach(spcaIllus)
opar <- par(no.readonly=TRUE)
## comparison PCA vs sPCA

# PCA
pca2A <- dudi.pca(dat2A$tab,center=TRUE,scale=FALSE,scannf=FALSE)
pca2B <- dudi.pca(dat2B$tab,center=TRUE,scale=FALSE,scannf=FALSE)
pca2C <- dudi.pca(dat2C$tab,center=TRUE,scale=FALSE,scannf=FALSE)
pca3 <- dudi.pca(dat3$tab,center=TRUE,scale=FALSE,scannf=FALSE,nf=2)
pca4 <- dudi.pca(dat4$tab,center=TRUE,scale=FALSE,scannf=FALSE,nf=2)

# sPCA
spca2A <- spca(dat2A,xy=dat2A$other$xy,ask=FALSE,type=1,plot=FALSE,scannf=FALSE,nfposi=1,

spca2B <- spca(dat2B,xy=dat2B$other$xy,ask=FALSE,type=1,plot=FALSE,scannf=FALSE,nfposi=1,

spca2C <- spca(dat2C,xy=dat2C$other$xy,ask=FALSE,type=1,plot=FALSE,scannf=FALSE,nfposi=0,

spca3 <- spca(dat3,xy=dat3$other$xy,ask=FALSE,type=1,plot=FALSE,scannf=FALSE,nfposi=1,nfn

spca4 <- spca(dat4,xy=dat4$other$xy,ask=FALSE,type=1,plot=FALSE,scannf=FALSE,nfposi=1,nfn

# an auxiliary function for graphics
plotaux <- function(x,analysis,axis=1,lab=NULL,...){
neig <- NULL
if(inherits(analysis,"spca")) neig <- nb2neig(analysis$lw$neighbours)
xrange <- range(x$other$xy[,1])
xlim <- xrange + c(-diff(xrange)*.1 , diff(xrange)*.45)
yrange <- range(x$other$xy[,2])
ylim <- yrange + c(-diff(yrange)*.45 , diff(yrange)*.1)

s.value(x$other$xy,analysis$li[,axis],include.ori=FALSE,addaxes=FALSE,cgrid=0,grid=FALSE,
...)

par(mar=rep(.1,4))
if(is.null(lab)) lab = gsub("[P]","",x$pop)
text(x$other$xy, lab=lab, col="blue", cex=1.2, font=2)
add.scatter({barplot(analysis$eig,col="grey");box();title("Eigenvalues",line=-1)},posi="b
}

# plots
plotaux(dat2A,pca2A,sub="dat2A - PCA",pos="bottomleft",csub=2)
plotaux(dat2A,spca2A,sub="dat2A - sPCA glob1",pos="bottomleft",csub=2)

plotaux(dat2B,pca2B,sub="dat2B - PCA",pos="bottomleft",csub=2)
plotaux(dat2B,spca2B,sub="dat2B - sPCA glob1",pos="bottomleft",csub=2)

plotaux(dat2C,pca2C,sub="dat2C - PCA",pos="bottomleft",csub=2)
plotaux(dat2C,spca2C,sub="dat2C - sPCA loc1",pos="bottomleft",csub=2,axis=2)

par(mfrow=c(2,2))
plotaux(dat3,pca3,sub="dat3 - PCA axis1",pos="bottomleft",csub=2)
```

```
plotaux(dat3,spca3,sub="dat3 - sPCA glob1",pos="bottomleft",csub=2)
plotaux(dat3,pca3,sub="dat3 - PCA axis2",pos="bottomleft",csub=2,axis=2)
plotaux(dat3,spca3,sub="dat3 - sPCA loc1",pos="bottomleft",csub=2,axis=2)

plotaux(dat4,pca4,lab=dat4$other$sup.pop,sub="dat4 - PCA axis1",pos="bottomleft",csub=2)
plotaux(dat4,spca4,lab=dat4$other$sup.pop,sub="dat4 - sPCA glob1",pos="bottomleft",csub=2
plotaux(dat4,pca4,lab=dat4$other$sup.pop,sub="dat4 - PCA axis2",pos="bottomleft",csub=2,a
plotaux(dat4,spca4,lab=dat4$other$sup.pop,sub="dat4 - sPCA loc1",pos="bottomleft",csub=2,

# color plot
par(opar)
colorplot(spca3, cex=4, main="colorplot sPCA dat3")
text(spca3$xy[,1], spca3$xy[,2], dat3$pop)

colorplot(spca4, cex=4, main="colorplot sPCA dat4")
text(spca4$xy[,1], spca4$xy[,2], dat4$other$sup.pop)

# detach data
detach(spcaIllus)
}
```

`truenames` *Restore true labels of an object*

Description

The function `truenames` returns some elements of an object (genind or genpop) using true names (as opposed to generic labels) for individuals, markers, alleles, and population.

Usage

```
## S4 method for signature 'genind'
truenames(x)
## S4 method for signature 'genpop'
truenames(x)
```

Arguments

`x`	a genind or a genpop object

Value

If x\$pop is empty (NULL), a matrix similar to the x\$tab slot but with true labels.

If x\$pop exists, a list with this matrix (\$tab) and a population vector with true names (\$pop).

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

Examples

```
data(microbov)
microbov

microbov$tab[1:5,1:5]
truenames(microbov)$tab[1:5,1:5]
```

`virtualClasses` *Virtual classes for adegenet*

Description

These virtual classes are only for internal use in adegenet

Objects from the Class

A virtual Class: No objects may be created from it.

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>

Index