Package 'adegenet'

April 15, 2010

Version 1.2-4

Date 2009/04/15

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

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

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

Suggests ade4, genetics, hierfstat, spdep, tripack, ape, pegas, graph, RBGL

Depends methods, MASS

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

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 two 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)

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

Both types of objects store information from molecular markers in a matrix (\$tab slot), that can be directly analyzed using multivariate methods such as Principal Component Analysis, Correspondance Analysis, etc. See the "dudi.[...]" methods in the ade4 package. Moreover, this package offers methods for manipulating and analyzing information coming from genetic markers (see below).

=== IMPORTING DATA ===
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. 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.

It is possible to extract SNPs from DNA alignments stored in the ape package using DNAbin2genind.

=== 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
- 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 genetic 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

- gstat.randtest: performs a Monte Carlo test of Goudet's G statistic, measuring population structure (based on g.stats.glob package hierfstat).
- 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: implements the Discriminant Analysis of Principal Component (DAPC, Jombart et al., submitted A).
- seqTrack: implements the SeqTrack algorithm for recontructing transmission trees of pathogens (Jombart et al., submitted B) .

=== 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.

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

=== 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().

Tutorials are available on the adegenet website, or by typing adegenetTutorial().

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

Details

Package:	adegenet
Type:	Package
Version:	1.2-4
Date:	2009-04-15
License:	GPL (>=2)

Author(s)

Thibaut Jombart <t.jombart@imperial.ac.uk>
with contributions of: Peter Solymos, Francois Balloux
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

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

`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

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

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` component is not returned.

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.

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

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.

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

Author(s)

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

Examples

```
data(nancycats)
nancycats
nancycats$pop

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

truenames(obj)$pop

# let's isolate two markers, fca23 and fca90
nancycats$loc.names

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

obj$loc.fac
obj$loc.names

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

# 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)
}
```

`Auxiliary functions`
Utilities functions for adegenet

Description

These functions are mostly used internally in adegenet. The notable exceptions are `adegenetWeb` which opens the adegenet website in the default navigator, and `adegenetTutorial` which opens online tutorials for adegenet.

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.

Usage

```
adegenetWeb()
adegenetTutorial(which=c("general","spca"))
.genlab(base, n)
```

Arguments

`which`	a character string being "general" or "spca", indicating which tutorial should be opened.
`base`	a character string forming the base of the labels
`n`	the number of labels to generate

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()

## this opens the general tutorial for adegenet
adegenetTutorial()

## this opens the sPCA tutorial for adegenet
adegenetTutorial("spca")
```

```
## End(Not run)

.genlab("Locus-",11)
```

---

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.

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.

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 the matched call.

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$xy,ask=FALSE,type=1,plot=FALSE,scannf=FALSE,nfposi=1,nfr
colorplot(spca3, cex=4, main="colorplot: a sPCA example")
text(spca3$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

Important: this method is currently under review. Please email the author before using it.

These functions implement the Discriminant Analysis of Principal Components (DAPC). See 'details' section for a succint description of the method. DAPC implementation calls upon `dudi.pca` from the `ade4` package and `lda` from the `MASS` package.

`dapc` performs the DAPC on a `data.frame`, a `matrix`, or a `genind` object, and returns an object with class `dapc`. If data are stored in a `data.frame` or a `matrix`, these have to be quantitative data (i.e., `numeric` or `integers`), as opposed to `characters` or `factors`.

Other functions are:

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

- `summary.dapc`: extracts useful information from a `dapc` object.

- `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.

Usage

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

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

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

## S3 method for class 'dapc':
scatter(x, xax=1, yax=2,
        col=rainbow(length(levels(x$grp))), posi="bottomleft", bg="grey",
        ratio=0.3, csub=1.2, ...)

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

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

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

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.

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).

xax,yax — integers specifying which principal components of DAPC should be shown in x and y axes.

col — a suitable color to be used for groups. The specified vector should match the number of groups, not the number of individuals.

posi,bg,ratio,csub — arguments used to customize the inset in scatterplots of DAPC results. See add.scatter documentation in the ade4 package for more details.

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

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

pch — a numeric indicating the type of point to be used to indicate the prior group of individuals (see points documentation for more details).

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).

- `scatter.dapc, assignplot` return the matched call.

Author(s)

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

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

- `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)


## Not run:
example(eHGDP)
example(H3N2)

## End(Not run)
```

---

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 *PLoS 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 (on 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):
$D_4(a,b) = \frac{1}{\nu}\sum_{k=1}^{\nu}\sqrt{\frac{1}{2}\sum_{j=1}^{m(k)}(p_{aj}^k - p_{bj}^k)^2}$

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

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 *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

Examples

```
if(require(hierfstat)){

obj <- read.fstat(system.file("data/diploid.dat",package="hierfstat"))

X <- genind2hierfstat(obj)
X

read.fstat.data(paste(.path.package("hierfstat"),"/data/diploid.dat",sep="",collapse=""),
}
if(require(genetics)){
genind2genotype(obj)
}
```

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

Description

Important: this method is currently under review. Please email the author before using it.

These functions implement the clustering procedure used in Discriminant Analysis of Principal Components (DAPC, Jombart et al. submitted). 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`.

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.

`.find.sub.clusters` is a hidden function called in some instances of `find.clusters`, and should not be called directly by the user.

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.

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("min","diff", "conserv"),
              max.n.clust=round(nrow(x)/10), n.iter=1e3, n.start=10,
              center=TRUE, scale=TRUE, ...)

## 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("min","diff", "conserv"),
              max.n.clust=round(nrow(x@tab)/10), n.iter=1e3, n.start=10,
              scale=FALSE, scale.method=c("sigma", "binom"),
              truenames=TRUE, ...)
```

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, as automatic procedures have not been fully evaluated.
criterion	a character string matching "min", "diff", or "conserv", indicating the criterion for automatic selection of the optimal number of clusters. Honestly, you should go for interactive selection of the number of clusters. Do as you wish. No warranty. If you still want to give it a try, see details.
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).
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.

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, that we do not recommend for now, is automatic selection based on a fixed criterion. For this, set choose.n.clust to FALSE and specify the criterion you want to use, from the following values:

- "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.

- "diff": model selection based on successive improvement of the test statistic. This procedure attempts to increase K until the model improvement (difference in successive BIC, AIC, or WSS) is no longer important. May be more appropriate to models relating to stepping stones.

- "conserv": another criterion meant to be conservative, in that it seeks a good fit with a minimum number of clusters. Unlike "diff", it does not rely on differences between successive statistics, but rather 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 NULLif 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. Discriminant analysis of principal components: a new method for the analysis of genetically structured populations. Submitted to *Genetics*.

See Also

- dapc: implements the 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")
```

fstat *F statistics for genind objects*

Description

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

Usage

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

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'.
`fstonly`	a logical stating whether only the Fst value should be returned (TRUE) instead of all F statistics (FALSE, default).

Value

A matrix of F statistics.

Author(s)

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

See Also

`varcomp.glob`, `gstat.randtest`

Examples

```
if(require(hierfstat)){
data(nancycats)
fstat(nancycats)
}
```

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

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.

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)
```

---

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)
```

---

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

g.stats.glob, fstat, test.g, test.within, test.between, as.randtest, 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

This dataset consists of 1903 strains of seasonal influenza (H3N2) distributed worldwide, and typed at 125 SNPs located in the hemagglutinin (HA) segment. These data were gathered from DNA sequences available from Genbank (http://www.ncbi.nlm.nih.gov/Genbank/).

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 *Genetics*.

Examples

```
## 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, ratio=.2)
```

---

haploGen *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>

---

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 per population from genpop objects. This is possible for codominant markers (@type="codom"). For haploid data, Hs still provides a measure of within-population genetic diversity.

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
```

---

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, at=NULL, threshold=quantile(x,0.75), axis=1, fac=NULL, byfac=FALS
            lab=names(x), cex.lab=0.7, cex.fac=1, lab.jitter=0,
            main="Loading plot", xlab="Variables", ylab="Loadings",...)
```

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.
`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.
`...`	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.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 the same 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 or genpop) by marker, returning a list of objects whose components each correspond to a marker.

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"))
```

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.

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

```
data(microbov)

# separate all markers
obj <- seploc(microbov)
names(obj)

obj$INRA5
```

---

seppop *Separate genotypes per population*

Description

The function seppop splits a genind object by population, returning a list of objects whose components each correspond to a population.

By default, components of the list are genind objects. It can also be a matrix of genotypes corresponding to the x\$tab.

Usage

```
## S4 method for signature 'genind':
seppop(x,pop=NULL,truenames=TRUE,res.type=c("genind","matrix"),
  drop=FALSE)
```

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).

Value

According to 'rse.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
```

---

seqTrack *SeqTrack algorithm*

Description

Important: this method is currently under review. It 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>

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
- alignement (seqinr package): to come...

Usage

```
DNAbin2genind(x, pop=NULL, exp.char=c("a","t","g","c"), na.char=NULL, polyThres=
```

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.

Examples

```
if(require(ape)){
data(woodmouse)
x <- DNAbin2genind(woodmouse)
x
genind2df(x)
}
```

---

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)
```

---

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