The adegenet Package

September 10, 2007

Version 1.0-2

Date 2007/09/08

Title Genetic data handling for multivariate analysis using ade4

Author Thibaut Jombart <jombart@biomserv.univ-lyon1.fr>

Maintainer Thibaut Jombart <jombart@biomserv.univ-lyon1.fr>

Suggests ade4, genetics, hierfstat, spdep, tripack

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

License GPL version 2 or newer

R topics documented:

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

Description

The function HWE.test.genind performs Hardy-Weinberg Equilibrium test on multilocus data (object of class genind). The test itself is performed using the function HWE.test of the genetics package. 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

```
HWE.test.genind(x,pop=NULL,permut=FALSE,nsim=1999,hide.NA=TRUE,res.type=c("full","m
```

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 ⟨jombart@biomserv.univ-lyon1.fr⟩

See Also

HWE.test,chisq.test

Examples

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

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

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

---

adegenet-package *Genetic data handling for multivariate analysis*

Description

This package is devoted to manipulate data obtained from molecular markers. The newly defined classes of object facilitate their analysis within the multivariate framework of the ade4 package. However, this package also provides interfaces with other packages, making Hardy-Weinberg equilibrium test, F statistics, Goudet's G test for population structure, or linkage disequilibrium measure available directly or using simple conversion functions.
Since the second version (1.0-1), adegenet also includes spatial analysis function like Monmonier algorithm (see monmonier) to define genetic boundaries among individuals or populations.

The basic class of object is genind, and contains genotypes (genind stands for genotypes-individuals). It can be obtained by converting files from GENETIX, Fstat and Genepop using import2genind.
The second class is genpop: such object contains alleles counts per populations and loci. It can be obtained from any genind object using genind2genpop.
It is also possible to obtain a table of allelic frequencies using makefreq on a genpop object. In all cases, missing data can be treated using different options.

The package proposes useful functions for genind objects:
- HWE.test.genind to test for Hardy-Weinberg equilibrium on every locus x population combinaisons (based on HWE.test, package genetics).
- gstat.randtest is Monte Carlo test (class randtest) of Goudet's G statistic measuring population structure (based on g.stat.glob, package hierfstat).
- genind2genotype assures conversion into genotype objects used in genetics and LDheatmap

packages.
- genind2hierfstat assures conversion into the format (particular data.frame) used in hierfstat package.

Moreover, several genetic distances between populations can be computed using dist.genpop. Genetic distances between individuals are not yet implemented.

Lastly, the Monmonier algorithm is implemented to seek genetic boundaries. The main function is monmonier, but better (optimized) boundaries can be obtained using optimize.monmonier. Object of the class monmonier can be plotted and printed using the corresponding methods.

Details

Package:	adegenet
Type:	Package
Version:	1.0-2
Date:	2007-09-08
License:	GPL version 2 or newer

These are the essential functions provided by the package:

genind: adegenet class for individual genotypes

genpop: adegenet class for allele counts in populations

import2genind: Conversion function for adegenet (from GENETIX, Fstat, Genepop)

makefreq: Function to generate allelic frequencies

HWE.test.genind: Hardy-Weinberg Equilibrium test for multilocus data

gstat.randtest: Monte Carlo test of Goudet's G statistic for multilocus data

monmonier: Monmonier algorithm to find genetic boundaries among genetic entities on a connection network

Author(s)

Thibaut Jombart <jombart@biomserv.univ-lyon1.fr>

References

See adegenet website: http://pbil.univ-lyon1.fr/software/adegenet/

See Also

ade4 package for multivariate analysis

`Auxiliary functions`
Utilities functions

Description

Most of theses functions are to be used with genind and genpop objects.
truenames returns elements of the object with using true names (as opposed to generic labels) for individuals, markers, alleles, and population.
The function seploc splits the table (x$tab) by marker, allowing separate analysis of markers.
An exception is adegenetWeb, which simply opens the adegenet website in the default navigator.

Usage

```
truenames(x)
seploc(x,truenames=FALSE)
adegenetWeb()
```

Arguments

x	a genind or genpop object.
truenames	a logical indicating whether generic labels (FALSE,default) or true names should be used (TRUE).

Value

The function truenames returns a matrix similar to x$tab but with true labels. If x$pop exists, it returns a list with this matrix ($tab) and a population vector with true names ($pop).

The function seploc applied to genind or genpop objects returns a list of matrices, one per marker.

Author(s)

Thibaut Jombart ⟨jombart@biomserv.univ-lyon1.fr⟩

Examples

```
data(microbov)
# restore true names
truenames(microbov)$tab[1:5,1:5]

# isolate each marker
obj <- seploc(microbov,truenames=TRUE)
```

```
names(obj)

# make a new object with INRA5
head(obj$INRA5)
inra5.gind <- as.genind(obj$INRA5)
inra5.gind

# perform tests only on this marker
if(require(genetics)){
hw.test <- HWE.test.genind(inra5.gind,pop=microbov$pop,res.type="matrix",permut=TRUE)
hw.test
}

if(require(hierfstat)){
g.test <- gstat.randtest(inra5.gind,pop=microbov$pop,nsim=99)
g.test
}
```

`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 kind of graphes. `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 = 1, result.type = "nb", d1 = NULL,
    d2 = NULL, k = 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).
`type`	an integer giving the type of graph (see details). Used if `ask=FALSE`
`result.type`	a character giving the class of the returned object. Either "nb" (default) or "listw", both from `spdep` package.
`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`.
`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 6 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)

Value

Returns a connection network having the class `nb` or `listw`.

Author(s)

Thibaut Jombart ⟨jombart@biomserv.univ-lyon1.fr⟩

References

See Also

`monmonier`

Examples

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

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

`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 `quasieuclid` 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):
$$D_1(a,b) = -\ln\left(\frac{\sum_{k=1}^{\nu}\sum_{j=1}^{m(k)} p_{aj}^k p_{bj}^k}{\sqrt{\sum_{k=1}^{\nu}\sum_{j=1}^{m(k)} \left(p_{aj}^k\right)^2}\sqrt{\sum_{k=1}^{\nu}\sum_{j=1}^{m(k)} \left(p_{bj}^k\right)^2}}\right)$$

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

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

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

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

Value

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

Author(s)

Thibaut Jombart ⟨jombart@biomserv.univ-lyon1.fr⟩
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., Ocaña 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

quasieuclid,dudi.pco

Examples

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

listDist <- lapply(1:5, function(i) quasieuclid(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))}
}
```

---

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 ⟨jombart@biomserv.univ-lyon1.fr⟩

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 <- fstat2genind(system.file("data/diploid.dat",package="hierfstat"))

X <- genind2hierfstat(obj)
X

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

genind *adegenet class for individual genotypes*

Description

The objects of class genind contain individual genotypes.
It consists in a list with several components (see value section).
The function genind2genpop converts individuals genotypes of known population into a genpop object.
The summary of a genind object invisibly returns a list of components (see value section). The function as.genind is called by import functions (see import2genind).

Usage

```
is.genind(x)
as.genind(tab=NULL,pop=NULL,prevcall=NULL)
## S3 method for class 'genind':
print(x, ...)
## S3 method for class 'genind':
summary(object, ...)
genind2genpop(x,pop=NULL,missing=NA,quiet=FALSE)
```

Arguments

x	an object of class genind.
tab	a individuals x alleles matrix of genotypes coded as allelic frequencies.
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.
prevcall	call of an object, for internal use.
...	other -unused- arguments
object	an object of class genind.
missing	can be NA, 0, or "replace". See details for more information.
quiet	logical stating whether a conversion message must be printed (TRUE,default) or not (FALSE).

Details

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
- "replace": when an allele is not typed in a population, it is assigned an allele count so that the allelic frequency in this populations is the same as the frequency in the whole dataset.
If allele 'j' of locus 'k' in pop 'i' is missing, the count value is number 'x' so that the frequency 'x/s' ('s' being the number of observations in 'k') equals the frequency 'f' computed on the whole data (i.e. considering all pop as one)

Then x verifies:

$$x/s = f(1-f) => x = f(1-f)s$$

Note on 'pop': the factor is sorted in the order of appearance of each modality.

Value

`tab`	matrix of genotypes -in rows- for all alleles -in columns-. Values are frequency: '0' if the genotype does not have the corresponding allele, '1' for an homozygote and 0.5 for an heterozygte.Rows and columns are given generic names.
`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`.
`loc.names`	character vector containing the real names of the loci
`loc.nall`	integer vector giving the number of alleles per locus
`loc.fac`	locus factor for the columns of `tab`
`all.names`	list having one component per locus, each containing a character vector of alleles names
`call`	the matched call
`pop`	(optional) factor giving the population of each individual
`pop.names`	(optional) vector giving the real names of the populations
`N`	(summary) total number of genotypes.
`pop.eff`	(summary) populations sample size.
`loc.nall`	(summary) number of alleles per locus.
`pop.nall`	(summary) number of alleles per population.
`NA.perc`	(summary) percentage of - appearing - missing data.
`Hobs`	(summary) observed heterozygosity.
`Hexp`	(summary) expected heterozygosity.

Author(s)

Thibaut Jombart ⟨jombart@biomserv.univ-lyon1.fr⟩

References

See Also

`genpop`, `import2genind`, `genetix2genind`, `genepop2genind`, `fstat2genind`

Examples

```
obj <- genetix2genind(system.file("files/nancycats.gtx",package="adegenet"),missing="mean")
is.genind(obj)
summary(obj)
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)
}
```

---

genpop *adegenet class for allele counts in populations*

Description

The objects of class genpop contain alleles counts for several loci.
It consists in a list with several components (see value 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.

Usage

```
is.genpop(x)
as.genpop(tab = NULL, prevcall = NULL)
## S3 method for class 'genpop':
print(x, ...)
## S3 method for class 'genpop':
summary(object, ...)
```

Arguments

x	an object of class genpop.
tab	a populations x alleles matrix of allele counts.
prevcall	call of an object, for internal use.

...	other -unused- arguments
object	an object of class genpop.

Value

tab	matrix of alleles counts for each combinaison of population -in rows- and alleles -in columns-. Rows and columns are given generic names.
pop.names	character vector containing the real names of the populations
loc.names	character vector containing the real names of the loci
loc.nall	integer vector giving the number of alleles per locus
loc.fac	locus factor for the columns of tab
all.names	list having one component per locus, each containing a character vector of alleles names
call	the matched call
npop	(summary) number of populations.
loc.nall	(summary) number of alleles per locus.
pop.nall	(summary) number of alleles per population.
NA.perc	(summary) percentage of - appearing - missing data.

Author(s)

Thibaut Jombart ⟨jombart@biomserv.univ-lyon1.fr⟩

References

See Also

makefreq, genind, import2genind, genetix2genind, genepop2genind, fstat2genind

Examples

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

obj2 <- genind2genpop(obj1)
is.genpop(obj2)
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="replace")
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")
}
```

`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.
```

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 ⟨jombart@biomserv.univ-lyon1.fr⟩

See Also

`g.stats.glob`, `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 <- genetix2genind(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$Locality,method="between",sub.pop=gtrunchier$Patch)
}
```

`import` *Conversion function for adegenet*

Description

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 functions are :
- `genetix2genind` for GENETIX files (.gtx). Note that this function is called by the others.
- `genepop2genind` for Genepop files (.gen)
- `fstat2genind` for Fstat files `.dat`

Usage

```
import2genind(file,missing=NA,quiet=FALSE)
genetix2genind(file=NULL,X=NULL,pop=NULL,missing=NA,quiet=FALSE)
genepop2genind(file,missing=NA,quiet=FALSE)
fstat2genind(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).
`X`	if file is not provided, `genetix2genind` can be used on a data frame with genotypes in GENETIX format (e.g. "080082" for an heterozygote with alleles 80 and 82); individuals are in rows, loci are in columns. Missing values are coded as "000000".
`pop`	an optional factor giving the population of each genotype in 'x'.

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.

Beware: same data in different formats are not expected to produce the 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 ⟨jombart@biomserv.univ-lyon1.fr⟩

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).

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

`read.fstat.data`

Examples

```
genetix2genind(system.file("files/nancycats.gtx",package="adegenet"))

fstat2genind(system.file("files/nancycats.dat",package="adegenet"))

genepop2genind(system.file("files/nancycats.gen",package="adegenet"))

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

if(require(hierfstat)){
obj <- fstat2genind(system.file("data/diploid.dat",package="hierfstat"))
obj
}
```

`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)
```

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)

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 ⟨jombart@biomserv.univ-lyon1.fr⟩

References

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="replace")
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 Laloë (INRA, Jouy-en-Josas, France)

References

Laloë D., Jombart T., Dufour A.-B. and Moazami-Goudarzi K. (2007) Consensus genetic structuring and typological value of markers using Multiple Co-Inertia Analysis. accepted in *Genetics Selection Evolution*.

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
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
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 by finding the path exhibiting the largest differences (provided in a distance matrix) between neighbouring objects.

The highest distance between two linked objects (i.e. neighbours) is found, giving the starting point of the path. Starting from this point, the algorithm seeks the highest distance between immediate neighbours, and so on until a threshold value is attained. It is recommended to choose this threshold from the barplot of sorted local differences: a boundary will likely be indicated by 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 function 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 (largest sum of local differences). 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, nrun=1,
skip.local.diff=rep(0,nrun),scanthres=is.null(threshold))

optimize.monmonier(xy, dist, cn, ntry=10, return.best=TRUE,
display.graph=TRUE, threshold=NULL, scanthres=is.null(threshold))

## 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
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
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 reflect 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 ⟨jombart@biomserv.univ-lyon1.fr⟩

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

nb

Examples

```
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$x1),cn1$cn,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$x2),cn2$cn,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
temp <- genind2hierfstat(sim2pop)
varcomp.glob(temp[,1],temp[,-1])
# Fst = 0.038
}
```

```
## run monmonier algorithm

# build connection network
gab <- chooseCN(sim2pop$xy,ask=FALSE,type=2)$cn

# filter random noise
pca1 <- dudi.pca(sim2pop$tab,scale=FALSE, scannf=FALSE, nf=1)

# run the algorithm
mon1 <- monmonier(sim2pop$xy,dist(pca1$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$xy,pch=temp,cex=2)
legend("topright",leg=c("Pop A", "Pop B"),pch=c(17,19))

### interactive example

# est-west separation
xy <- matrix(runif(120,0,10), ncol=2)
x1 <- rnorm(60)
x1[xy[,1] > 5] <- x1[xy[,1] > 5]+4
cn1 <- chooseCN(xy,type=2,ask=FALSE)
mon1 <- optimize.monmonier(xy,dist(x1),cn1$cn,ntry=6)

# graphics
plot(mon1,x1)
plot(mon1,x1,met="greylevel")

# square 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),cn2$cn,ntry=6)

# graphics
plot(mon2,x2,method="greylevel",add.arr=FALSE,bwd=6,col="red",csize=.5)
## End(Not run)
```

`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 colony

# 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
}
```

---

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 ⟨jombart@biomserv.univ-lyon1.fr⟩

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$xy,ask=FALSE,type=2)$cn

# filter random noise
pca1 <- dudi.pca(sim2pop$tab,scale=FALSE, scannf=FALSE, nf=1)

# run the algorithm
mon1 <- monmonier(sim2pop$xy,dist(pca1$l1[,1]),gab,scanthres=FALSE)

# graphical display
temp <- sim2pop$pop
levels(temp) <- c(17,19)
temp <- as.numeric(as.character(temp))
plot(mon1)
points(sim2pop$xy,pch=temp,cex=2)
legend("topright",leg=c("Pop A", "Pop B"),pch=c(17,19))
## End(Not run)
```

Index