The adegenet Package

February 13, 2008

Version 1.1-0

Date 2008/02/12

Title The adegenet package for R: classes and methods for the multivariate analysis of genetic markers.

Author Thibaut Jombart <jombart@biomserv.univ-lyon1.fr>, with contributions from Peter Solymos

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

Suggests ade4, genetics, hierfstat, spdep, tripack

Depends methods

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

License GPL (>=2)

LazyLoad yes

Collate classes.R auxil.R makefreq.R chooseCN.R dist.genpop.R export.R gstat.randtest.R HWE.R import.R monmonier.R coords.monmonier.R spca.R spca.rtests.R zzz.R hybridize.R

R topics documented:

`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","matrix"))
```

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(obj)

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

#more precise view to...
```

```
obj.test$fca90$P10
}
```

Accessors *Accessors for adegenet objects*

Description

Several accessors 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.

Usage

Value

A genind or genpop object.

Author(s)

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

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
```

---

adegenet-package *The adegenet package*

Description

This package is devoted to the multivariate analysis of molecular markers data. It 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`.

=== 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.

=== 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.stat.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.
- `hybridize`: implements hybridization between two populations.
- `spca`: implements Jombart et al. (in revision) spatial Principal Component Analysis
- `global.rtest`: implements Jombart et al. (in revision) test for global spatial structures
- `local.rtest`: implements Jombart et al. (in revision) test for local spatial structures

Adegenet also proposes several real or simulated datasets.

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

To cite adegenet, please use `citation("adegenet")`.

Details

Package:	adegenet
Type:	Package
Version:	1.1-0
Date:	2008-02-12
License:	GPL (>=2)

Author(s)

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

References

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

See Also

The `ade4` package for multivariate analysis

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)
as.genind(tab,pop=NULL,prevcall=NULL)
is.genind(x)
```

Arguments

`tab`	a individuals x alleles matrix of genotypes coded as allelic frequencies, i.e. like in a genind object
`pop`	a factor giving the population of each genotype in 'x'
`prevcall`	call of an object
`x`	an object

Value

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

Author(s)

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

See Also

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

Examples

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

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

---

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)
as.genpop(tab,prevcall=NULL)
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
`x`	an object

Value

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

Author(s)

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

See Also

`"genpop"`, 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
```

`Auxiliary functions` *Utilities functions for adegenet*

Description

These functions are mostly used internally in adegenet. The notable exception is `adegenetWeb`, which opens the adegenet website in the default navigator.

The other functions are:
- `.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()
.genlab(base, n)
```

Arguments

`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 ⟨jombart@biomserv.univ-lyon1.fr⟩

Examples

```
## Not run:
## this opens the adegenet website
adegenetWeb()
## 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. 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. The xy coordinates are passed as attribute to the created object.

Author(s)

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

References

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))
}
```

---

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 neighbouring pairs) which are divided by the path (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)
```

---

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 (2, 4 or 6) characters coding both alleles without separation. Missing values are coded by NA or zeros. Uncomplete strings are filled with "0" at the begining. For instance, "912" in a 4-characters coding scheme is interpreted as "0912", thus as an heterozygote with alleles "09" and "12". Note that this format is the one of GENETIX for a 6-characters coding scheme

The function `genind2df` converts a genind back to such a data.frame.

Usage

```
df2genind(X, ncode=NULL, ind.names=NULL, loc.names=NULL, pop=NULL,
 missing=NA)
genind2df(x,pop=NULL, sep="")
```

Arguments

`X`	a matrix or a data.frame (see decription)
`ncode`	an optional integer the number of characters used for coding one genotype at one locus (can be 2, 4 or 6). 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.
`x`	a genind object
`sep`	a character used to separate two alleles

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 for `df2genind`; a matrix of biallelic genotypes for `genind2df`

Author(s)

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

See Also

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

Examples

```
df <- data.frame(loc1=c("11","11","12","32"),
loc2=c(NA,"34","55","15"),loc3=c("22","22","21","22"))
row.names(df) <- .genlab("geontype",4)
df

obj <- df2genind(df)
obj
truenames(obj)
genind2df(obj)
```

---

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

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 ⟨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 <- 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=""),nlc
}
if(require(genetics)){
genind2genotype(obj)
}
```

`genind` *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 an empty valid genind object. 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-. 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.

`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`.

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 ⟨jombart@biomserv.univ-lyon1.fr⟩

References

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)
}
```

---

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)
```

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).

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

- "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.

Value

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

Author(s)

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

See Also

genind, genpop, `na.replace`

Examples

```
data(nancycats)
nancycats
catpop <- genind2genpop(nancycats)
catpop
summary(catpop)
```

---

genpop — *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

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

Extends

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

Methods

dist `signature(x = "genpop", method = "ANY", diag = "ANY", upper = "ANY", p = "missing")`: ...

names `signature(x = "genpop")`: ...

show `signature(object = "genpop")`: ...

summary `signature(object = "genpop")`: ...

Author(s)

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

References

See Also

`as.genpop`, `is.genpop`,codemakefreq, `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")
}
```

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 xpop. 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 <- 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$Locality,method="between",sub.pop=gtrunchier$Patch)
}
```

---

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 ⟨jombart@biomserv.univ-lyon1.fr⟩

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 two ways to import genotyping data to a genind object: from a data.frame with a given format (see `df2genind`, or from a file with a recognized extension.

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 ⟨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).

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"), n.ind=237, n.loc=9, col.lab=1, col.pop=2, ask=FALSE)
```

---

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="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 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

spca,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$x1),cn1,threshold=2)
plot(mon1,mondata1$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
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
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

# 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,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,ntry=6)

# 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 ⟨jombart@biomserv.univ-lyon1.fr⟩

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 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
}
```

`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 ⟨jombart@biomserv.univ-lyon1.fr⟩

`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 ⟨jombart@biomserv.univ-lyon1.fr⟩

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 ⟨jombart@biomserv.univ-lyon1.fr⟩

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 ⟨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 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=FALSE, col.lab=NULL, col
```

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.
`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 ⟨jombart@biomserv.univ-lyon1.fr⟩

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"),
  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 ⟨jombart@biomserv.univ-lyon1.fr⟩

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
```

---

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 ⟨jombart@biomserv.univ-lyon1.fr⟩

See Also

seppop, repool

Examples

```
data(microbov)

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

obj$INRA5
```

`seploc` *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"))
```

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.

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 ⟨jombart@biomserv.univ-lyon1.fr⟩

See Also

seploc, repool

Examples

```
data(microbov)

obj <- seppop(microbov)
names(obj)

obj$Salers
```

---

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@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

Usage

```
spca(obj, xy=NULL, cn=NULL, scale=FALSE, scannf=TRUE, nfposi=1, nfnega=1, type=1, a
plot.nb=TRUE, edit.nb=FALSE ,truenames=TRUE, d1=NULL, d2=NULL, k=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, ...)

## S3 method for class 'spca':
screeplot(x, ..., main=NULL)
```

Arguments

`obj`	a `genind` or `genpop` object.
`xy`	an matrix or data.frame with two columns for x and y coordinates. Can be NULL if cn is provided.
`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.
`scale`	a logical indicating whether alleles should be scaled to unit variance (TRUE) or not (FALSE, default).
`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). Used if `ask=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`.
`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.

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.

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.
`cn`	a connection network of class `nb`.

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 ⟨jombart@biomserv.univ-lyon1.fr⟩

References

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

Smouse, P. E. and Peakall, R. (1999) Spatial autocorrelation analysis of individual multiallele and multilocus genetic structure. *Heredity*, **82**, 561–573.

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`

Examples

```
## data(spcaIllus) illustrates the sPCA
## see ?spcaIllus
##

example(spcaIllus)
```

`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 ⟨jombart@biomserv.univ-lyon1.fr⟩

References

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

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)
```

`spcaIllus` *Simulated data illustrating the sPCA*

Description

Datasets illustrating the spatial Principal Component Analysis (Jombart et al. submitted). 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 ⟨jombart@biomserv.univ-lyon1.fr⟩

Source

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

References

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

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)
## 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,nfn

spca2B <- spca(dat2B,xy=dat2B$other$xy,ask=FALSE,type=1,plot=FALSE,scannf=FALSE,nfposi=1,nfn

spca2C <- spca(dat2C,xy=dat2C$other$xy,ask=FALSE,type=1,plot=FALSE,scannf=FALSE,nfposi=0,nfn

spca3 <- spca(dat3,xy=dat3$other$xy,ask=FALSE,type=1,plot=FALSE,scannf=FALSE,nfposi=1,nfnega

spca4 <- spca(dat4,xy=dat4$other$xy,ask=FALSE,type=1,plot=FALSE,scannf=FALSE,nfposi=1,nfnega

# an auxiliary function for graphics
plotaux <- function(x,analysis,axis=1,lab=NULL,...){
neig <- NULL
if(inherits(analysis,"spca")) neig <- nb2neig(analysis$cn)
xrange <- range(x$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,nei
...)

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="bott
}

# plots
get(getOption("device"))(width=10,height=5)
par(mfrow=c(1,2))
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)

get(getOption("device"))()
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,axis
plotaux(dat4,spca4,lab=dat4$other$sup.pop,sub="dat4 - sPCA loc1",pos="bottomleft",csub=2,axi

# 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 ⟨jombart@biomserv.univ-lyon1.fr⟩

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 ⟨jombart@biomserv.univ-lyon1.fr⟩

Index