Package: poppr (via r-universe)

October 28, 2026

Type Package

Title Genetic Analysis of Populations with Mixed Reproduction

Version 2.9.8

Maintainer Zhian N. Kamvar <zkamvar@gmail.com>

Encoding UTF-8

URL https://grunwaldlab.github.io/poppr/,
https://github.com/grunwaldlab/poppr/,
https://grunwaldlab.github.io/Population_Genetics_in_R/

Description Population genetic analyses for hierarchical analysis of partially clonal populations built upon the architecture of the 'adegenet' package. Originally described in Kamvar, Tabima, and Grünwald (2014) <doi:10.7717/peerj.281> with version 2.0 described in Kamvar, Brooks, and Grünwald (2015) <doi:10.3389/fgene.2015.00208>.

MailingList https://groups.google.com/d/forum/poppr

BugReports https://github.com/grunwaldlab/poppr/issues/

Depends R (>= 2.15.1), adegenet (>= 2.0.0)

Imports stats, graphics, grDevices, utils, vegan, ggplot2, ape (>= 3.1-1), igraph (>= 1.0.0), methods, ade4, pegas, polysat, dplyr (>= 0.4), rlang, boot, shiny, magrittr, progressr

Suggests testthat, knitr, rmarkdown, poweRlaw, cowplot

Config/Needs/check dbailleul/RClone

License GPL-2 | GPL-3

VignetteBuilder knitr

RoxygenNote 7.3.2.9000

NeedsCompilation yes

Author Zhian N. Kamvar [cre, aut] (ORCID: <https://orcid.org/0000-0003-1458-7108>), Javier F. Tabima [aut] (ORCID: <https://orcid.org/0000-0002-3603-2691>), Sydney

E. Everhart [ctb, dtc] (ORCID:
<https://orcid.org/0000-0002-5773-1280>), Jonah C. Brooks
[aut], Stacy A. Krueger-Hadfield [ctb] (ORCID:
<https://orcid.org/0000-0002-7324-7448>), Erik Sotka [ctb],
Brian J. Knaus [ctb] (ORCID:
<https://orcid.org/0000-0003-1665-4343>), Patrick G. Meirmans
[ctb] (ORCID: <https://orcid.org/0000-0002-6395-8107>),
Frédéric D. Chevalier [ctb] (ORCID:
<https://orcid.org/0000-0003-2611-8106>), David Folarin [aut],
Niklaus J. Grünwald [ths] (ORCID:
<https://orcid.org/0000-0003-1656-7602>)

Repository https://cran.r-universe.dev

Date/Publication 2025-08-24 02:25:51 UTC

RemoteUrl https://github.com/cran/poppr

RemoteRef HEAD

RemoteSha 5a9b40a7140c33dc7b1bc0275e7409018a944125

Contents

poppr-package . . . 3
aboot . . . 9
Aeut . . . 12
as.snpclone . . . 13
bitwise.dist . . . 14
boot.ia . . . 16
bootgen2genind . . . 17
bruvo.boot . . . 19
bruvo.dist . . . 21
bruvo.msn . . . 24
clonecorrect . . . 28
cutoff_predictor . . . 30
diss.dist . . . 32
diversity_boot . . . 33
diversity_ci . . . 35
diversity_stats . . . 38
filter_stats . . . 40
fix_replen . . . 42
genclone-class . . . 43
genind2genalex . . . 45
genotype_curve . . . 47
getfile . . . 49
greycurve . . . 50
ia . . . 51
imsn . . . 56
incomp . . . 58
info_table . . . 59

informloci . . . 61
is.snpclone . . . 62
locus_table . . . 63
make_haplotypes . . . 65
missingno . . . 66
mlg . . . 68
mlg.filter . . . 72
mll . . . 76
mll.custom . . . 78
mll.reset . . . 79
monpop . . . 80
nei.dist . . . 81
old2new_genclone . . . 83
partial_clone . . . 84
pgen . . . 84
Pinf . . . 86
plot_poppr_msn . . . 87
poppr . . . 92
poppr.all . . . 97
poppr.amova . . . 98
poppr.msn . . . 102
poppr_has_parallel . . . 105
popsub . . . 106
Pram . . . 107
private_alleles . . . 108
psex . . . 110
rare_allele_correction . . . 114
read.genalex . . . 116
recode_polyploids . . . 118
rraf . . . 120
rrmlg . . . 122
samp.ia . . . 124
shufflepop . . . 125
test_replen . . . 127
upgma . . . 128
win.ia . . . 129

Index **132**

`poppr-package` *The* **poppr** *R package*

Description

Poppr provides tools for population genetic analysis that include genotypic diversity measures, genetic distances with bootstrap support, native organization and handling of population hierarchies, and clone correction.

To cite **poppr**, please use `citation("poppr")`. When referring to **poppr** in your manuscript, please use lower case unless it occurs at the beginning of a sentence.

Details

This package relies on the **adegenet** package. It is built around the adegenet::genind and adegenet::genlight object. Genind objects store genetic information in a table of allele frequencies while genlight objects store SNP data efficiently by packing binary allele calls into single bits. **Poppr** has extended these object into new objects called genclone and snpclone, respectively. These objects are designed for analysis of clonal organisms as they add the **@mlg** slot for keeping track of multilocus genotypes and multilocus lineages.

Documentation: Documentation is available for any function by typing `?function_name` in the R console. Detailed topic explanations live in the package vignettes:

Vignette	command
Data import and manipulation	`vignette("poppr_manual", "poppr")`
Algorithms and Equations	`vignette("algo", "poppr")`
Multilocus Genotype Analysis	`vignette("mlg", "poppr")`

Essential functions for importing and manipulating data are detailed within the *Data import and manipulation* vignette, details on algorithms used in **poppr** are within the *Algorithms and equations* vignette, and details for working with multilocus genotypes are in *Multilocus Genotype Analysis*.

Examples of analyses are available in a primer written by Niklaus J. Grünwald, Zhian N. Kamvar, and Sydney E. Everhart at https://grunwaldlab.github.io/Population_Genetics_in_R/.

Getting help: If you have a specific question or issue with **poppr**, feel free to contribute to the google group at https://groups.google.com/d/forum/poppr. If you find a bug and are a github user, you can submit bug reports at https://github.com/grunwaldlab/poppr/issues. Otherwise, leave a message on the groups. Personal emails are highly discouraged as they do not allow others to learn.

Functions in poppr

Below are descriptions and links to functions found in **poppr**. Be aware that all functions in **adegenet** are also available. The functions are documented as:

- `function_name()` (data type) - Description

Where 'data type' refers to the type of data that can be used:

m	a genclone or genind object
s	a snpclone or genlight object
x	a different data type (e.g. a matrix from `mlg.table()`)

Data import/export

- `getfile()` (x) - Provides a quick GUI to grab files for import
- `read.genalex()` (x) - Reads GenAlEx formatted csv files to a genind object
- `genind2genalex()` (m) - Converts genind objects to GenAlEx formatted csv files
- `genclone2genind()` (m) - Removes the @mlg slot from genclone objects
- `as.genambig()` (m) - Converts genind data to **polysat**'s genambig data structure.
- `bootgen2genind()` (x) - see `aboot()` for details)

Data Structures

Data structures "genclone" (based off of adegenet's genind) and "snpclone" (based off of adegenet's genlight for large SNP data sets). Both of these data structures are defined by the presence of an extra MLG slot representing multilocus genotype assignments, which can be a numeric vector or a MLG class object.

- genclone - Handles microsatellite, presence/absence, and small SNP data sets
- snpclone - Designed to handle larger binary SNP data sets.
- MLG - An internal class holding a data frame of multilocus genotype assignments that acts like a vector, allowing the user to easily switch between different MLG definitions.
- bootgen - An internal class used explicitly for `aboot()` that inherits the gen-class virtual object. It is designed to allow for sampling loci with replacement.
- bruvomat - An internal class designed to handle bootstrapping for Bruvo's distance where blocks of integer loci can be shuffled.

Data manipulation

- `as.genclone()` (m) - Converts genind objects to genclone objects
- `missingno()` (m) - Handles missing data
- `clonecorrect()` (m | s) - Clone-censors at a specified population hierarchy
- `informloci()` (m) - Detects and removes phylogenetically uninformative loci
- `popsub()` (m | s) - Subsets genind objects by population
- `shufflepop()` (m) - Shuffles genotypes at each locus using four different shuffling algorithms
- `recode_polyploids()` (m | x) - Recodes polyploid data sets with missing alleles imported as "0"
- `make_haplotypes()` (m | s) - Splits data into pseudo-haplotypes. This is mainly used in AMOVA.
- `test_replen()` (m) - Tests for inconsistent repeat lengths in microsatellite data. For use in `bruvo.dist()` functions.
- `fix_replen()` (m) - Fixes inconsistent repeat lengths. For use in `bruvo.dist()` functions.

Genetic distances

- `bruvo.dist()` (m) - Bruvo's distance (see also: `fix_replen()`)
- `diss.dist()` (m) - Absolute genetic distance (see `prevosti.dist()`)
- `nei.dist()` (m | x) - Nei's 1978 genetic distance
- `rogers.dist()` (m | x) - Rogers' euclidean distance
- `reynolds.dist()` (m | x) - Reynolds' coancestry distance
- `edwards.dist()` (m | x) - Edwards' angular distance
- `prevosti.dist()` (m | x) - Prevosti's absolute genetic distance
- `bitwise.dist()` (s) - Calculates fast pairwise distances for genlight objects.

Bootstrapping

- `aboot()` (m | s | x) - Creates a bootstrapped dendrogram for any distance measure
- `bruvo.boot()` (m) - Produces dendrograms with bootstrap support based on Bruvo's distance
- `diversity_boot()` (x) - Generates boostrap distributions of diversity statistics for multilocus genotypes
- `diversity_ci()` (m | s | x) - Generates confidence intervals for multilocus genotype diversity.
- `resample.ia()` (m) - Calculates the index of association over subsets of data.

Multilocus Genotypes

- `mlg()` (m | s) - Calculates the number of multilocus genotypes
- `mll()` (m | s) - Displays the current multilocus lineages (genotypes) defined.
- `nmll()` (m | s) - Same as `mlg()`.
- `mlg.crosspop()` (m | s) - Finds all multilocus genotypes that cross populations
- `mlg.table()` (m | s) - Returns a table of populations by multilocus genotypes
- `mlg.vector()` (m | s) - Returns a vector of a numeric multilocus genotype assignment for each individual
- `mlg.id()` (m | s) - Finds all individuals associated with a single multilocus genotype
- `mlg.filter()` (m | s) - Collapses MLGs by genetic distance
- `filter_stats()` (m | s) - Calculates mlg.filter for all algorithms and plots
- `cutoff_predictor()` (x) - Predicts cutoff threshold from mlg.filter.
- `mll.custom()` (m | s) - Allows for the custom definition of multilocus lineages
- `mll.levels()` (m | s) - Allows the user to change levels of custom MLLs.
- `mll.reset()` (m | s) - Reset multilocus lineages.
- `diversity_stats()` (x) - Creates a table of diversity indices for multilocus genotypes.

Index of Association Analysis

Analysis of multilocus linkage disequilibrium.

- `ia()` (m) - Calculates the index of association
- `pair.ia()` (m) - Calculates the index of association for all loci pairs.
- `win.ia()` (s) - Index of association windows for genlight objects.
- `samp.ia()` (s) - Index of association on random subsets of loci for genlight objects.

Population Genetic Analysis

- `poppr.amova()` (m | s) - Analysis of Molecular Variance (as implemented in ade4)
- `poppr()` (m | x) - Returns a diversity table by population
- `poppr.all()` (m | x) - Returns a diversity table by population for all compatible files specified
- `private_alleles()` (m) - Tabulates the occurrences of alleles that only occur in one population.
- `locus_table()` (m) - Creates a table of summary statistics per locus.
- `rrmlg()` (m | x) - Round-robin multilocus genotype estimates.
- `rraf()` (m) - Round-robin allele frequency estimates.
- `pgen()` (m) - Probability of genotypes.
- `psex()` (m) - Probability of observing a genotype more than once.
- rare_allele_correction (m) - rules for correcting rare alleles for round-robin estimates.
- `incomp()` (m) - Check data for incomparable samples.

Visualization

- `imsn()` (m | s) - Interactive construction and visualization of minimum spanning networks
- `plot_poppr_msn()` (m | s | x) - Plots minimum spanning networks produced in poppr with scale bar and legend
- `greycurve()` (x) - Helper to determine the appropriate parameters for adjusting the grey level for msn functions
- `bruvo.msn()` (m) - Produces minimum spanning networks based off Bruvo's distance colored by population
- `poppr.msn()` (m | s | x) - Produces a minimum spanning network for any pairwise distance matrix related to the data
- `info_table()` (m) - Creates a heatmap representing missing data or observed ploidy
- `genotype_curve()` (m | x) - Creates a series of boxplots to demonstrate how many markers are needed to represent the diversity of your data.

Datasets

- `Aeut()` - (AFLP) Oomycete root rot pathogen *Aphanomyces euteiches* (Grünwald and Hoheisel, 2006)
- `monpop()` - (SSR) Peach brown rot pathogen *Monilinia fructicola* (Everhart and Scherm, 2015)
- `partial_clone()` - (SSR) partially-clonal data simulated via simuPOP (Peng and Amos, 2008)
- `Pinf()` - (SSR) Potato late blight pathogen *Phytophthora infestans* (Goss et. al., 2014)
- `Pram()` - (SSR) Sudden Oak Death pathogen *Phytophthora ramorum* (Kamvar et. al., 2015; Goss et. al., 2009)

Author(s)

Zhian N. Kamvar, Jonah C. Brooks, Sydney E. Everhart, Javier F. Tabima, Stacy Krueger-Hadfield, Erik Sotka, Niklaus J. Grünwald

Maintainer: Zhian N. Kamvar

References

——— Papers announcing poppr ———

Kamvar ZN, Tabima JF, Grünwald NJ. (2014) Poppr: an R package for genetic analysis of populations with clonal, partially clonal, and/or sexual reproduction. PeerJ 2:e281 doi:10.7717/peerj.281

Kamvar ZN, Brooks JC and Grünwald NJ (2015) Novel R tools for analysis of genome-wide population genetic data with emphasis on clonality. Front. Genet. 6:208. doi:10.3389/fgene.2015.00208

——— Papers referencing data sets ———

Grünwald, NJ and Hoheisel, G.A. 2006. Hierarchical Analysis of Diversity, Selfing, and Genetic Differentiation in Populations of the Oomycete *Aphanomyces euteiches*. Phytopathology 96:1134-1141 doi: doi:10.1094/PHYTO961134

SE Everhart, H Scherm, (2015) Fine-scale genetic structure of *Monilinia fructicola* during brown rot epidemics within individual peach tree canopies. Phytopathology 105:542-549 doi: doi:10.1094/PHYTO03140088R

Bo Peng and Christopher Amos (2008) Forward-time simulations of nonrandom mating populations using simuPOP. *bioinformatics*, 24 (11): 1408-1409.

Goss, Erica M., Javier F. Tabima, David EL Cooke, Silvia Restrepo, William E. Fry, Gregory A. Forbes, Valerie J. Fieland, Martha Cardenas, and Niklaus J. Grünwald. (2014) "The Irish potato famine pathogen *Phytophthora infestans* originated in central Mexico rather than the Andes." Proceedings of the National Academy of Sciences 111:8791-8796. doi: doi:10.1073/pnas.1401884111

Kamvar, Z. N., Larsen, M. M., Kanaskie, A. M., Hansen, E. M., & Grünwald, N. J. (2015). Spatial and temporal analysis of populations of the sudden oak death pathogen in Oregon forests. Phytopathology 105:982-989. doi: doi:10.1094/PHYTO12140350FI

Goss, E. M., Larsen, M., Chastagner, G. A., Givens, D. R., and Grünwald, N. J. 2009. Population genetic analysis infers migration pathways of *Phytophthora ramorum* in US nurseries. PLoS Pathog. 5:e1000583. doi: doi:10.1371/journal.ppat.1000583

See Also

Useful links:

- https://grunwaldlab.github.io/poppr/
- https://github.com/grunwaldlab/poppr/
- https://grunwaldlab.github.io/Population_Genetics_in_R/
- Report bugs at https://github.com/grunwaldlab/poppr/issues/

---

aboot *Calculate a dendrogram with bootstrap support using any distance applicable to genind or genclone objects.*

Description

Calculate a dendrogram with bootstrap support using any distance applicable to genind or genclone objects.

Usage

```
aboot(
  x,
  strata = NULL,
  tree = "upgma",
  distance = "nei.dist",
  sample = 100,
  cutoff = 0,
  showtree = TRUE,
  missing = "mean",
  mcutoff = 0,
  quiet = FALSE,
  root = NULL,
  ...
)
```

Arguments

x	a genind-class, genpop-class, genclone-class, adegenet::genlight, snpclone or matrix object.
strata	a formula specifying the strata to be used to convert x to a genclone object if x is a genind object. Defaults to NULL. See details.
tree	a text string or function that can calculate a tree from a distance matrix. Defaults to "upgma". Note that you must load the package with the function for it to work.
distance	a character or function defining the distance to be applied to x. Defaults to nei.dist().
sample	An integer representing the number of bootstrap replicates Default is 100.

cutoff	An integer from 0 to 100 setting the cutoff value to return the bootstrap values on the nodes. Default is 0.
showtree	If TRUE (Default), a dendrogram will be plotted. If FALSE, nothing will be plotted.
missing	any method to be used by missingno(): "mean" (default), "zero", "loci", "genotype", or "ignore".
mcutoff	a value between 0 (default) and 1 defining the percentage of tolerable missing data if the missing parameter is set to "loci" or "genotype". This should only be set if the distance metric can handle missing data.
quiet	if FALSE (default), a progress bar will be printed to screen.
root	is the tree rooted? This is a parameter passed off to ape::boot.phylo(). If the tree parameter returns a rooted tree (like UPGMA), this should be TRUE, otherwise (like neighbor-joining), it should be false. When set to NULL (default), the tree is considered rooted if ape::is.ultrametric() is true.
...	any parameters to be passed off to the distance method.

Details

This function automates the process of bootstrapping genetic data to create a dendrogram with bootstrap support on the nodes. It will randomly sample with replacement the loci of a gen (genind/genpop) object or the columns of a numeric matrix, **assuming that all loci/columns are independent**. The process of randomly sampling gen objects with replacement is carried out through the use of an internal class called bootgen. This is necessary due to the fact that columns in the genind matrix are defined as alleles and are thus interrelated. This function will specifically bootstrap loci so that results are biologically relevant. With this function, the user can also define a custom distance to be performed on the genind or genclone object. If you have a data frame-like object where all of the columns are independent or pairs of columns are independent, then it may be simpler to use ape::boot.phylo() to calculate your bootstrap support values.

the strata argument: There is an argument called strata. This argument is useful for when you want to bootstrap by populations from a adegenet::genind() object. When you specify strata, the genind object will be converted to adegenet::genpop() with the specified strata.

Value

an object of class ape::phylo().

Note

prevosti.dist() and diss.dist() are exactly the same, but diss.dist() scales better for large numbers of individuals (n > 125) at the cost of required memory.

missing data: Missing data is not allowed by many of the distances. Thus, one of the first steps of this function is to treat missing data by setting it to the average allele frequency in the data set. If you are using a distance that can handle missing data (Prevosti's distance), you can set missing = "ignore" to allow the distance function to handle any missing data. See missingno() for details on missing data.

Bruvo's Distance: While calculation of Bruvo's distance is possible with this function, it is optimized in the function bruvo.boot().

References

Kamvar ZN, Brooks JC and Grünwald NJ (2015) Novel R tools for analysis of genome-wide population genetic data with emphasis on clonality. Frontiers in Genetics 6:208. doi:10.3389/fgene.2015.00208

See Also

nei.dist() edwards.dist() rogers.dist() reynolds.dist() prevosti.dist() diss.dist() bruvo.boot() ape::boot.phylo() adegenet::dist.genpop() dist() bootgen2genind() bootgen

Examples

```
data(nancycats)
nan9 <- popsub(nancycats, 9)

set.seed(9999)
# Generate a tree using nei's distance
neinan <- aboot(nan9, dist = nei.dist)

set.seed(9999)
# Generate a tree using custom distance
bindist <- function(x) dist(tab(x), method = "binary")
binnan <- aboot(nan9, dist = bindist)

## Not run:
# Distances from other packages.
#
# Sometimes, distance functions from other packages will have the constraint
# that the incoming data MUST be genind. Internally, aboot uses the
# bootgen class ( class?bootgen ) to shuffle loci, and will throw an error
# The function bootgen2genind helps fix that. Here's an example of a function
# that expects a genind class from above
bindist <- function(x){
  stopifnot(is.genind(x))
  dist(tab(x), method = "binary")
}
#
# Fails:
# aboot(nan9, dist = bindist)
## Error: is.genind(x) is not TRUE
#
# Add bootgen2genind to get it working!
# Works:
aboot(nan9, dist = function(x) bootgen2genind(x) %>% bindist)

# AFLP data
data(Aeut)

# Nei's distance
anei <- aboot(Aeut, dist = nei.dist, sample = 1000, cutoff = 50)
```

```
# Rogers' distance
arog <- aboot(Aeut, dist = rogers.dist, sample = 1000, cutoff = 50)

# This can also be run on genpop objects
strata(Aeut) <- other(Aeut)$population_hierarchy[-1]
Aeut.gc <- as.genclone(Aeut)
setPop(Aeut.gc) <- ~Pop/Subpop
Aeut.pop <- genind2genpop(Aeut.gc)
set.seed(5000)
aboot(Aeut.pop, sample = 1000) # compare to Grunwald et al. 2006

# You can also use the strata argument to convert to genpop inside the function.
set.seed(5000)
aboot(Aeut.gc, strata = ~Pop/Subpop, sample = 1000)

# And genlight objects
# From glSim:
## 1,000 non structured SNPs, 100 structured SNPs
x <- glSim(100, 1e3, n.snp.struc=100, ploid=2)
aboot(x, distance = bitwise.dist)

# Utilizing other tree methods

library("ape")

aboot(Aeut.pop, tree = fastme.bal, sample = 1000)

# Utilizing options in other tree methods

myFastME <- function(x) fastme.bal(x, nni = TRUE, spr = FALSE, tbr = TRUE)
aboot(Aeut.pop, tree = myFastME, sample = 1000)


## End(Not run)
```

`Aeut` *Oomycete root rot pathogen *Aphanomyces euteiches* AFLP data*

Description

The Aeut dataset consists of 187 isolates of the Oomycete root rot pathogen, *Aphanomyces euteiches* collected from two different fields in NW Oregon and W Washington, USA.

Usage

```
data(Aeut)
```

Format

a [genind()] object with two populations containing a data frame in the ‘other‘ slot called ‘population_hierarchy‘. This data frame gives indices of the populations and subpopulations for the data set.

References

Grunwald, NJ and Hoheisel, G.A. 2006. Hierarchical Analysis of Diversity, Selfing, and Genetic Differentiation in Populations of the Oomycete *Aphanomyces euteiches*. Phytopathology 96:1134-1141 doi: doi:10.1094/PHYTO961134

`as.snpclone` *Create a snpclone object from a genlight object.*

Description

Wrapper for snpclone initializer.

Usage

```
as.snpclone(x, ..., parallel = FALSE, n.cores = NULL, mlg, mlgclass = TRUE)
```

Arguments

`x`	a `genlight` or `snpclone` object
`...`	arguments to be passed on to the genlight constructor. These are not used if x is not missing.
`parallel`	should the parallel package be used to construct the object?
`n.cores`	how many cores should be utilized? See documentation for `genlight` for details.
`mlg`	a vector of multilocus genotypes or an object of class MLG for the new snpclone object.
`mlgclass`	if `TRUE` (default), the multilocus genotypes will be represented as an `MLG` object.

Author(s)

Zhian N. Kamvar

Examples

```
(x <- as.snpclone(glSim(100, 1e3, ploid=2)))
## Not run:
# Without parallel processing
system.time(x <- as.snpclone(glSim(1000, 1e5, ploid=2)))

# With parallel processing... doesn't really save you much time.
```

```
system.time(x <- as.snpclone(glSim(1000, 1e5, ploid=2, parallel = TRUE),
                             parallel = TRUE))

## End(Not run)
```

`bitwise.dist` *Calculate dissimilarity or Euclidean distance for genlight objects*

Description

This function calculates both dissimilarity and Euclidean distances for genlight or snpclone objects.

Usage

```
bitwise.dist(
  x,
  percent = TRUE,
  mat = FALSE,
  missing_match = TRUE,
  scale_missing = FALSE,
  euclidean = FALSE,
  differences_only = FALSE,
  threads = 0L
)
```

Arguments

`x`	a genlight or snpclone object.
`percent`	`logical`. Should the distance be represented from 0 to 1? Default set to `TRUE`. `FALSE` will return the distance represented as integers from 1 to n where n is the number of loci. This option has no effect if `euclidean = TRUE`
`mat`	`logical`. Return a matrix object. Default set to `FALSE`, returning a dist object. `TRUE` returns a matrix object.
`missing_match`	`logical`. Determines whether two samples differing by missing data in a location should be counted as matching at that location. Default set to `TRUE`, which forces missing data to match with anything. `FALSE` forces missing data to not match with any other information, **including other missing data**.
`scale_missing`	A logical. If `TRUE`, comparisons with missing data is scaled up proportionally to the number of columns used by multiplying the value by `m / (m - x)` where m is the number of loci and x is the number of missing sites. This option matches the behavior of base R's `dist()` function. Defaults to `FALSE`.
`euclidean`	`logical`. if `TRUE`, the Euclidean distance will be calculated.

differences_only
: logical. When differences_only = TRUE, the output will reflect the number of different loci. The default setting, differences_only = FALSE, reflects the number of different alleles. Note: this has no effect on haploid organisms since 1 locus = 1 allele. This option is NOT recommended.

threads
: The maximum number of parallel threads to be used within this function. A value of 0 (default) will attempt to use as many threads as there are available cores/CPUs. In most cases this is ideal. A value of 1 will force the function to run serially, which may increase stability on some systems. Other values may be specified, but should be used with caution.

Details

The default distance calculated here is quite simple and goes by many names depending on its application. The most familiar name might be the Hamming distance, or the number of differences between two strings.

As of poppr version 2.8.0, this function now also calculates Euclidean distance and is considerably faster and more memory-efficient than the standard dist() function.

Value

A dist object containing pairwise distances between samples.

Note

This function is optimized for genlight and snpclone objects. This does not mean that it is a catch-all optimization for SNP data. Three assumptions must be met for this function to work:

1. SNPs are bi-allelic
2. Samples are haploid or diploid
3. All samples have the same ploidy

If the user supplies a genind or genclone object, prevosti.dist() will be used for calculation.

Author(s)

Zhian N. Kamvar, Jonah C. Brooks

See Also

diss.dist(), snpclone, genlight, win.ia(), samp.ia()

Examples

```
set.seed(999)
x <- glSim(n.ind = 10, n.snp.nonstruc = 5e2, n.snp.struc = 5e2, ploidy = 2)
x
# Assess fraction of different alleles
system.time(xd <- bitwise.dist(x, threads = 1L))
xd
```

```
# Calculate Euclidean distance
system.time(xdt <- bitwise.dist(x, euclidean = TRUE, scale_missing = TRUE, threads = 1L))
xdt

## Not run:

# This function is more efficient in both memory and speed than [dist()] for
# calculating Euclidean distance on genlight objects. For example, we can
# observe a clear speed increase when we attempt a calculation on 100k SNPs
# with 10% missing data:

set.seed(999)
mat <- matrix(sample(c(0:2, NA),
                     100000 * 50,
                     replace = TRUE,
                     prob = c(0.3, 0.3, 0.3, 0.1)),
              nrow = 50)
glite <- new("genlight", mat, ploidy = 2)

# Default Euclidean distance
system.time(dist(glite))

# Bitwise dist
system.time(bitwise.dist(glite, euclidean = TRUE, scale_missing = TRUE))


## End(Not run)
```

---

boot.ia *Bootstrap the index of association*

Description

This function will perform the index of association on a bootstrapped data set multiple times to create a distribution, showing the variation of the index due to repeat observations.

Usage

```
boot.ia(gid, how = "partial", reps = 999, quiet = FALSE, ...)
```

Arguments

gid	a genind or genclone object
how	method of bootstrap. The default how = "partial" will include all the unique genotypes and sample with replacement from the unique genotypes until the total number of individuals has been reached. Using how = "full" will randomly sample with replacement from the data as it is. Using how = "psex" will sample

	from the full data set after first weighting the samples via the probability of encountering the nth occurence of a particular multilocus genotype. See psex() for details.
reps	an integer specifying the number of replicates to perform. Defaults to 999.
quiet	a logical. If FALSE, a progress bar will be displayed. If TRUE, the progress bar is suppressed.
...	options passed on to psex()

Value

a data frame with the index of association and standardized index of association in columns. Number of rows represents the number of reps.

Note

This function is experimental. Please do not use this unless you know what you are doing.

See Also

ia(), pair.ia(), psex()

Examples

```
data(Pinf)
boot.ia(Pinf, reps = 99)
```

---

bootgen2genind *Switch between genind and genclone objects.*

Description

as.genclone will create a genclone object from a genind object OR anything that can be passed to the genind initializer.

Usage

```
bootgen2genind(bg)

as.genclone(x, ..., mlg, mlgclass = TRUE)

genclone2genind(x)

as.genambig(x)
```

Arguments

bg	a bootgen object
x	a genind or genclone object
...	arguments passed on to the genind constructor
mlg	an optional vector of multilocus genotypes as integers
mlgclass	should the mlg slot be of class MLG?

Details

genclone2genind will remove the mlg slot from the genclone object, creating a genind object.

as.genambig will convert a genind or genclone object to a polysat genambig class.

Author(s)

Zhian N. Kamvar

See Also

splitStrata, genclone, read.genalex aboot

Examples

```
data(Aeut)
Aeut

# Conversion to genclone ----------------------------------------------
Aeut.gc <- as.genclone(Aeut)
Aeut.gc

# Conversion to genind ------------------------------------------------
Aeut.gi <- genclone2genind(Aeut.gc)
Aeut.gi

# Conversion to polysat's "genambig" class ----------------------------
if (require("polysat")) {
  data(Pinf)
  Pinf.gb <- as.genambig(Pinf)
  summary(Pinf.gb)
}

data(nancycats)

# Conversion to bootgen for random sampling of loci -------------------
nan.bg  <- new("bootgen", nancycats[pop = 9])
nan.bg

# Conversion back to genind -------------------------------------------
nan.gid <- bootgen2genind(nan.bg)
nan.gid
```

bruvo.boot *Create a tree using Bruvo's Distance with non-parametric bootstrapping.*

Description

Create a tree using Bruvo's Distance with non-parametric bootstrapping.

Usage

```
bruvo.boot(
  pop,
  replen = 1,
  add = TRUE,
  loss = TRUE,
  sample = 100,
  tree = "upgma",
  showtree = TRUE,
  cutoff = NULL,
  quiet = FALSE,
  root = NULL,
  ...
)
```

Arguments

pop	a genind or genclone object
replen	a vector of integers indicating the length of the nucleotide repeats for each microsatellite locus.
add	if TRUE, genotypes with zero values will be treated under the genome addition model presented in Bruvo et al. 2004.
loss	if TRUE, genotypes with zero values will be treated under the genome loss model presented in Bruvo et al. 2004.
sample	an integer indicated the number of bootstrap replicates desired.
tree	any function that can generate a tree from a distance matrix. Default is upgma.
showtree	logical if TRUE, a tree will be plotted with nodelabels.
cutoff	integer the cutoff value for bootstrap node label values (between 0 and 100).
quiet	logical defaults to FALSE. If TRUE, a progress bar and messages will be suppressed.
root	logical This is a parameter passed on to boot.phylo. If the tree argument produces a rooted tree (e.g. "upgma"), then this value should be TRUE. If it produces an unrooted tree (e.g. "nj"), then the value should be FALSE. By default, it is set to NULL, which will assume an unrooted phylogeny unless the function name contains "upgma".
...	any argument to be passed on to boot.phylo. eg. quiet = TRUE.

Details

This function will calculate a tree based off of Bruvo's distance and then utilize boot.phylo to randomly sample loci with replacement, recalculate the tree, and tally up the bootstrap support (measured in percent success). While this function can take any tree function, it has native support for two algorithms: nj and upgma. If you want to use any other functions, you must load the package before you use them (see examples).

Value

a tree of class phylo with nodelables

Note

Please refer to the documentation for bruvo.dist for details on the algorithm. If the user does not provide a vector of appropriate length for replen , it will be estimated by taking the minimum difference among represented alleles at each locus. IT IS NOT RECOMMENDED TO RELY ON THIS ESTIMATION.

Author(s)

Zhian N. Kamvar, Javier F. Tabima

References

Ruzica Bruvo, Nicolaas K. Michiels, Thomas G. D'Souza, and Hinrich Schulenburg. A simple method for the calculation of microsatellite genotype distances irrespective of ploidy level. Molecular Ecology, 13(7):2101-2106, 2004.

See Also

bruvo.dist, nancycats, upgma, nj, boot.phylo, nodelabels, tab, missingno.

Examples

```
# Please note that the data presented is assuming that the nancycat dataset
# contains all dinucleotide repeats, it most likely is not an accurate
# representation of the data.

# Load the nancycats dataset and construct the repeat vector.
data(nancycats)
ssr <- rep(2, 9)

# Analyze the 1st population in nancycats

bruvo.boot(popsub(nancycats, 1), replen = ssr)

## Not run:

# Always load the library before you specify the function.
library("ape")
```

```
# Estimate the tree based off of the BIONJ algorithm.

bruvo.boot(popsub(nancycats, 9), replen = ssr, tree = bionj)

# Utilizing  balanced FastME
bruvo.boot(popsub(nancycats, 9), replen = ssr, tree = fastme.bal)

# To change parameters for the tree, wrap it in a function.
# For example, let's build the tree without utilizing subtree-prune-regraft

myFastME <- function(x) fastme.bal(x, nni = TRUE, spr = FALSE, tbr = TRUE)
bruvo.boot(popsub(nancycats, 9), replen = ssr, tree = myFastME)


## End(Not run)
```

`bruvo.dist` *Bruvo's distance for microsatellites*

Description

Calculate the average Bruvo's distance over all loci in a population.

Usage

```
bruvo.dist(pop, replen = 1, add = TRUE, loss = TRUE, by_locus = FALSE)

bruvo.between(
  query,
  ref,
  replen = 1,
  add = TRUE,
  loss = TRUE,
  by_locus = FALSE
)
```

Arguments

pop	a genind or genclone object
replen	a vector of integers indicating the length of the nucleotide repeats for each microsatellite locus. E.g. a locus with a (CAT) repeat would have a replen value of 3. (Also see fix_replen)
add	if TRUE, genotypes with zero values will be treated under the genome addition model presented in Bruvo et al. 2004. See the **Note** section for options.
loss	if TRUE, genotypes with zero values will be treated under the genome loss model presented in Bruvo et al. 2004. See the **Note** section for options.

by_locus	indicator to get the results per locus. The default setting is by_locus = FALSE, indicating that Bruvo's distance is to be averaged over all loci. When by_locus = TRUE, a list of distance matrices will be returned.
query	a genind or genclone object
ref	a genind or genclone object

Details

Bruvo's distance between two alleles is calculated as

$$d = 1 - 2^{-|x|}$$

, where **x** is the number of repeat units between the two alleles (see the Algorithms and Equations vignette for more details). These distances are calculated over all combinations of alleles at a locus and then the minimum average distance between allele combinations is taken as the distance for that locus. All loci are then averaged over to obtain the distance between two samples. Missing data is ignored (in the same fashion as mean(c(1:9, NA), na.rm = TRUE)) if all alleles are missing. See the next section for other cases.

Polyploids: Ploidy is irrelevant with respect to calculation of Bruvo's distance. However, since it makes a comparison between all alleles at a locus, it only makes sense that the two loci need to have the same ploidy level. Unfortunately for polyploids, it's often difficult to fully separate distinct alleles at each locus, so you end up with genotypes that appear to have a lower ploidy level than the organism.

To help deal with these situations, Bruvo has suggested three methods for dealing with these differences in ploidy levels:

- **Infinite Model** - The simplest way to deal with it is to count all missing alleles as infinitely large so that the distance between it and anything else is 1. Aside from this being computationally simple, it will tend to **inflate distances between individuals**.
- **Genome Addition Model** - If it is suspected that the organism has gone through a recent genome expansion, **the missing alleles will be replace with all possible combinations of the observed alleles in the shorter genotype**. For example, if there is a genotype of [69, 70, 0, 0] where 0 is a missing allele, the possible combinations are: [69, 70, 69, 69], [69, 70, 69, 70], [69, 70, 70, 69], and [69, 70, 70, 70]. The resulting distances are then averaged over the number of comparisons.
- **Genome Loss Model** - This is similar to the genome addition model, except that it assumes that there was a recent genome reduction event and uses **the observed values in the full genotype to fill the missing values in the short genotype**. As with the Genome Addition Model, the resulting distances are averaged over the number of comparisons.
- **Combination Model** - Combine and average the genome addition and loss models.

As mentioned above, the infinite model is biased, but it is not nearly as computationally intensive as either of the other models. The reason for this is that both of the addition and loss models requires replacement of alleles and recalculation of Bruvo's distance. The number of replacements required is equal to n^k where where *n* is the number of potential replacements and *k* is the number of alleles to be replaced. To reduce the number of calculations and assumptions otherwise, Bruvo's distance will be calculated using the largest observed ploidy in pairwise comparisons. This means that when comparing [69,70,71,0] and [59,60,0,0], they will be treated as triploids.

Value

an object of class dist or a list of these objects if by_locus = TRUE

Functions

- bruvo.between(): Bruvo's distance between a query and a reference Only diferences between query individuals and reference individuals will be reported All other values are NaN

Note

Do not use missingno with this function.

Missing alleles and Bruvo's distance in poppr versions < 2.5: In earlier versions of **poppr**, the authors had assumed that, because the calculation of Bruvo's distance does not rely on orderd sets of alleles, the imputation methods in the genome addition and genome loss models would also assume unordered alleles for creating the hypothetical genotypes. This means that the results from this imputation did not consider all possible combinations of alleles, resulting in either an over- or under- estimation of Bruvo's distance between two samples with two or more missing alleles. This version of **poppr** considers all possible combinations when calculating Bruvo's distance for incomplete genotype with a negligable gain in computation time.

If you want to see the effect of this change on your data, you can use the global **poppr** option old.bruvo.model. Currently, this option is FALSE and you can set it by using options(old.bruvo.model = TRUE), but make sure to reset it to FALSE afterwards.

Repeat Lengths (replen): The replen argument is crucial for proper analysis of Bruvo's distance since the calculation relies on the knowledge of the number of steps between alleles. To calculate Bruvo's distance, your raw allele calls are first divided by the repeat lengths and then rounded. This can create a problem with repeat lengths of even size due to the IEC 60559 standard that says rounding at 0.5 is to the nearest even number, meaning that it is possible for two alleles that are one step apart may appear to be exactly the same. This can be fixed by subtracting a tiny number from the repeat length with the function fix_replen. Please consider using this before running Bruvo's distance.

Model Choice: The add and loss arguments modify the model choice accordingly:

- **Infinite Model:** add = FALSE, loss = FALSE
- **Genome Addition Model:** add = TRUE, loss = FALSE
- **Genome Loss Model:** add = FALSE, loss = TRUE
- **Combination Model** *(DEFAULT):* add = TRUE, loss = TRUE

Details of each model choice are described in the **Details** section, above. Additionally, genotypes containing all missing values at a locus will return a value of NA and not contribute to the average across loci.

Repeat Lengths: If the user does not provide a vector of appropriate length for replen , it will be estimated by taking the minimum difference among represented alleles at each locus. IT IS NOT RECOMMENDED TO RELY ON THIS ESTIMATION.

Author(s)

Zhian N. Kamvar

David Folarin

References

Ruzica Bruvo, Nicolaas K. Michiels, Thomas G. D'Souza, and Hinrich Schulenburg. A simple method for the calculation of microsatellite genotype distances irrespective of ploidy level. Molecular Ecology, 13(7):2101-2106, 2004.

See Also

fix_replen, test_replen, bruvo.boot, bruvo.msn

Examples

```
# Please note that the data presented is assuming that the nancycat dataset
# contains all dinucleotide repeats, it most likely is not an accurate
# representation of the data.

# Load the nancycats dataset and construct the repeat vector.
data(nancycats)
names(alleles(nancycats)) <- locNames(nancycats) # small bug in this data set
# Assume the alleles are all dinucleotide repeats.
ssr <- rep(2, nLoc(nancycats))
test_replen(nancycats, ssr)         # Are the repeat lengths consistent?
(ssr <- fix_replen(nancycats, ssr)) # Nope. We need to fix them.

# Analyze the first population in nancycats
bruvo.dist(popsub(nancycats, 1), replen = ssr)

## Not run:

# get the per locus estimates:
bruvo.dist(popsub(nancycats, 1), replen = ssr, by_locus = TRUE)

# View each population as a heatmap.
sapply(popNames(nancycats), function(x)
heatmap(as.matrix(bruvo.dist(popsub(nancycats, x), replen = ssr)), symm=TRUE))

## End(Not run)
```

---

bruvo.msn — *Create minimum spanning network of selected populations using Bruvo's distance.*

Description

Create minimum spanning network of selected populations using Bruvo's distance.

Usage

```
bruvo.msn(
  gid,
  replen = 1,
  add = TRUE,
  loss = TRUE,
  mlg.compute = "original",
  palette = topo.colors,
  sublist = "All",
  exclude = NULL,
  blacklist = NULL,
  vertex.label = "MLG",
  gscale = TRUE,
  glim = c(0, 0.8),
  gadj = 3,
  gweight = 1,
  wscale = TRUE,
  showplot = TRUE,
  include.ties = FALSE,
  threshold = NULL,
  clustering.algorithm = NULL,
  ...
)
```

Arguments

gid	a genind or genclone object
replen	a vector of integers indicating the length of the nucleotide repeats for each microsatellite locus.
add	if TRUE, genotypes with zero values will be treated under the genome addition model presented in Bruvo et al. 2004.
loss	if TRUE, genotypes with zero values will be treated under the genome loss model presented in Bruvo et al. 2004.
mlg.compute	if the multilocus genotypes are set to "custom" (see mll.custom for details) in your genclone object, this will specify which mlg level to calculate the nodes from. See details.
palette	a vector or function defining the color palette to be used to color the populations on the graph. It defaults to topo.colors. See examples for details.
sublist	a vector of population names or indexes that the user wishes to keep. Default to "ALL".
exclude	a vector of population names or indexes that the user wishes to discard. Default to NULL.
blacklist	DEPRECATED, use exclude.
vertex.label	a vector of characters to label each vertex. There are two defaults: "MLG" will label the nodes with the multilocus genotype from the original data set and "inds" will label the nodes with the representative individual names.

gscale	"grey scale". If this is TRUE, this will scale the color of the edges proportional to the observed distance, with the lines becoming darker for more related nodes. See greycurve for details.
glim	"grey limit". Two numbers between zero and one. They determine the upper and lower limits for the gray function. Default is 0 (black) and 0.8 (20% black). See greycurve for details.
gadj	"grey adjust". a positive integer greater than zero that will serve as the exponent to the edge weight to scale the grey value to represent that weight. See greycurve for details.
gweight	"grey weight". an integer. If it's 1, the grey scale will be weighted to emphasize the differences between closely related nodes. If it is 2, the grey scale will be weighted to emphasize the differences between more distantly related nodes. See greycurve for details.
wscale	"width scale". If this is TRUE, the edge widths will be scaled proportional to the inverse of the observed distance , with the lines becoming thicker for more related nodes.
showplot	logical. If TRUE, the graph will be plotted. If FALSE, it will simply be returned.
include.ties	logical. If TRUE, the graph will include all edges that were arbitrarily passed over in favor of another edge of equal weight. If FALSE, which is the default, one edge will be arbitrarily selected when two or more edges are tied, resulting in a pure minimum spanning network.
threshold	numeric. By default, this is NULL, which will have no effect. Any threshold value passed to this argument will be used in mlg.filter prior to creating the MSN. If you have a data set that contains contracted MLGs, this argument will override the threshold in the data set. See Details.
clustering.algorithm	string. By default, this is NULL. If threshold = NULL, this argument will have no effect. When supplied with either "farthest_neighbor", "average_neighbor", or "nearest_neighbor", it will be passed to mlg.filter prior to creating the MSN. If you have a data set that contains contracted MLGs, this argument will override the algorithm in the data set. See Details.
...	any other arguments that could go into plot.igraph

Details

The minimum spanning network generated by this function is generated via igraph's minimum.spanning.tree. The resultant graph produced can be plotted using igraph functions, or the entire object can be plotted using the function plot_poppr_msn, which will give the user a scale bar and the option to layout your data.

node sizes: The area of the nodes are representative of the number of samples. Because **igraph** scales nodes by radius, the node sizes in the graph are represented as the square root of the number of samples.

mlg.compute: Each node on the graph represents a different multilocus genotype. The edges on the graph represent genetic distances that connect the multilocus genotypes. In genclone objects, it is possible to set the multilocus genotypes to a custom definition. This creates a problem for clone

correction, however, as it is very possible to define custom lineages that are not monophyletic. When clone correction is performed on these definitions, information is lost from the graph. To circumvent this, The clone correction will be done via the computed multilocus genotypes, either "original" or "contracted". This is specified in the `mlg.compute` argument, above.

contracted multilocus genotypes: If your incoming data set is of the class `genclone`, and it contains contracted multilocus genotypes, this function will retain that information for creating the minimum spanning network. You can use the arguments `threshold` and `clustering.algorithm` to change the threshold or clustering algorithm used in the network. For example, if you have a data set that has a threshold of 0.1 and you wish to have a minimum spanning network without a threshold, you can simply add `threshold = 0.0`, and no clustering will happen.

The `threshold` and `clustering.algorithm` arguments can also be used to filter un-contracted data sets.

Value

`graph`	a minimum spanning network with nodes corresponding to MLGs within the data set. Colors of the nodes represent population membership. Width and color of the edges represent distance.
`populations`	a vector of the population names corresponding to the vertex colors
`colors`	a vector of the hexadecimal representations of the colors used in the vertex colors

Note

- **Please see the documentation for `bruvo.dist` for details on the algorithm**.
- The edges of these graphs may cross each other if the graph becomes too large.
- The nodes in the graph represent multilocus genotypes. The colors of the nodes are representative of population membership. It is not uncommon to see different populations containing the same multilocus genotype.

Author(s)

Zhian N. Kamvar, Javier F. Tabima

References

Ruzica Bruvo, Nicolaas K. Michiels, Thomas G. D'Souza, and Hinrich Schulenburg. A simple method for the calculation of microsatellite genotype distances irrespective of ploidy level. Molecular Ecology, 13(7):2101-2106, 2004.

See Also

`bruvo.dist`, `nancycats`, `plot_poppr_msn`, `mst` `bruvo.boot`, `greycurve` `poppr.msn`

Examples

```
# Load the data set.
data(nancycats)

# View populations 8 and 9 with default colors.
bruvo.msn(nancycats, replen = rep(2, 9), sublist=8:9, vertex.label="inds",
          vertex.label.cex=0.7, vertex.label.dist=0.4)
## Not run:
# View heat colors.
bruvo.msn(nancycats, replen=rep(2, 9), sublist=8:9, vertex.label="inds",
palette=heat.colors, vertex.label.cex=0.7, vertex.label.dist=0.4)

# View custom colors. Here, we use black and orange.
bruvo.msn(nancycats, replen=rep(2, 9), sublist=8:9, vertex.label="inds",
palette = colorRampPalette(c("orange", "black")), vertex.label.cex=0.7,
vertex.label.dist=0.4)

# View with darker shades of grey (setting the upper limit to 1/2 black 1/2 white).
bruvo.msn(nancycats, replen=rep(2, 9), sublist=8:9, vertex.label="inds",
palette = colorRampPalette(c("orange", "black")), vertex.label.cex=0.7,
vertex.label.dist=0.4, glim=c(0, 0.5))

# View with no grey scaling.
bruvo.msn(nancycats, replen=rep(2, 9), sublist=8:9, vertex.label="inds",
palette = colorRampPalette(c("orange", "black")), vertex.label.cex=0.7,
vertex.label.dist=0.4, gscale=FALSE)

# View with no line widths.
bruvo.msn(nancycats, replen=rep(2, 9), sublist=8:9, vertex.label="inds",
palette = colorRampPalette(c("orange", "black")), vertex.label.cex=0.7,
vertex.label.dist=0.4, wscale=FALSE)

# View with no scaling at all.
bruvo.msn(nancycats, replen=rep(2, 9), sublist=8:9, vertex.label="inds",
palette = colorRampPalette(c("orange", "black")), vertex.label.cex=0.7,
vertex.label.dist=0.4, gscale=FALSE)

# View the whole population, but without labels.
bruvo.msn(nancycats, replen=rep(2, 9), vertex.label=NA)

## End(Not run)
```

---

clonecorrect *Remove potential bias caused by cloned genotypes in genind or genclone object.*

Description

This function removes any duplicated multilocus genotypes from any specified population strata.

Usage

```
clonecorrect(pop, strata = 1, combine = FALSE, keep = 1)
```

Arguments

pop	a genind, genclone, or snpclone object
strata	a hierarchical formula or numeric vector. This will define the columns of the data frame in the strata slot to use.
combine	logical. When set to TRUE, the strata will be combined to create a new population for the clone-corrected genind or genclone object.
keep	integer. When combine is set to FALSE, you can use this flag to choose the levels of your population strata. For example: if your clone correction strata is set to "Pop", "Subpop", and "Year", and you want to analyze your populations with respect to year, you can set keep = c(1,3).

Details

This function will clone correct based on the stratification provided. To clone correct indiscriminately of population structure, set strata = NA.

Value

a clone corrected genclone, snpclone, or genind object.

Author(s)

Zhian N. Kamvar

Examples

```
# LOAD A. euteiches data set
data(Aeut)

# Redefine it as a genclone object
Aeut <- as.genclone(Aeut)
strata(Aeut) <- other(Aeut)$population_hierarchy[-1]

# Check the number of multilocus genotypes
mlg(Aeut)
popNames(Aeut)

# Clone correct at the population level.
Aeut.pop <- clonecorrect(Aeut, strata =  ~Pop)
mlg(Aeut.pop)
popNames(Aeut.pop)

## Not run:
# Clone correct at the subpopulation level with respect to population and
# combine.
Aeut.subpop <- clonecorrect(Aeut, strata = ~Pop/Subpop, combine=TRUE)
```

```
mlg(Aeut.subpop)
popNames(Aeut.subpop)

# Do the same, but set to the population level.
Aeut.subpop2 <- clonecorrect(Aeut, strata = ~Pop/Subpop, keep=1)
mlg(Aeut.subpop2)
popNames(Aeut.subpop2)

# LOAD H3N2 dataset
data(H3N2)

strata(H3N2) <- other(H3N2)$x

# Extract only the individuals located in China
country <- clonecorrect(H3N2, strata =  ~country)

# How many isolates did we have from China before clone correction?
sum(strata(H3N2, ~country) == "China") # 155

# How many unique isolates from China after clone correction?
sum(strata(country, ~country) == "China") # 79

# Something a little more complicated. (This could take a few minutes on
# slower computers)

# setting the hierarchy to be Country > Year > Month
c.y.m <- clonecorrect(H3N2, strata =  ~year/month/country)

# How many isolates in the original data set?
nInd(H3N2) # 1903

# How many after we clone corrected for country, year, and month?
nInd(c.y.m) # 1190

## End(Not run)
```

`cutoff_predictor` *Predict cutoff thresholds for use with mlg.filter*

Description

Given a series of thresholds for a data set that collapse it into one giant cluster, this will search the top fraction of threshold differences to find the largest difference. The average between the thresholds spanning that difference is the cutoff threshold defining the clonal lineage threshold.

Usage

```
cutoff_predictor(thresholds, fraction = 0.5)
```

Arguments

thresholds	a vector of numerics coming from mlg.filter where the threshold has been set to the maximum threshold theoretically possible.
fraction	the fraction of the data to seek the threshold.

Value

a numeric value representing the threshold at which multilocus lineages should be defined.

Note

This function originally appeared in doi:10.5281/zenodo.17424. This is a bit of a blunt instrument.

Author(s)

Zhian N. Kamvar

References

ZN Kamvar, JC Brooks, and NJ Grünwald. 2015. Supplementary Material for Frontiers Plant Genetics and Genomics 'Novel R tools for analysis of genome-wide population genetic data with emphasis on clonality'. DOI: doi:10.5281/zenodo.17424

Kamvar ZN, Brooks JC and Grünwald NJ (2015) Novel R tools for analysis of genome-wide population genetic data with emphasis on clonality. Front. Genet. 6:208. doi: doi:10.3389/fgene.2015.00208

See Also

filter_stats mlg.filter

Examples

```
data(Pinf)
pinfreps <- fix_replen(Pinf, c(2, 2, 6, 2, 2, 2, 2, 2, 3, 3, 2))
pthresh  <- filter_stats(Pinf, distance = bruvo.dist, replen = pinfreps,
                         plot = TRUE, stats = "THRESHOLD", threads = 1L)

# prediction for farthest neighbor
cutoff_predictor(pthresh$farthest)

# prediction for all algorithms
sapply(pthresh, cutoff_predictor)
```

diss.dist *Calculate a distance matrix based on relative dissimilarity*

Description

diss.dist uses the same discrete dissimilarity matrix utilized by the index of association (see ia for details). By default, it returns a distance reflecting the number of allelic differences between two individuals. When percent = TRUE, it returns a ratio of the number of observed differences by the number of possible differences. Eg. two individuals who share half of the same alleles will have a distance of 0.5. This function can analyze distances for any marker system.

Usage

```
diss.dist(x, percent = FALSE, mat = FALSE)
```

Arguments

x	a genind object.
percent	logical. Should the distance be represented as a percent? If set to FALSE (default), the distance will be reflected as the number of alleles differing between to individuals. When set to TRUE, These will be divided by the ploidy multiplied by the number of loci.
mat	logical. Return a matrix object. Default set to FALSE, returning a dist object. TRUE returns a matrix object.

Details

The distance calculated here is quite simple and goes by many names, depending on its application. The most familiar name might be the Hamming distance, or the number of differences between two strings.

Value

Pairwise distances between individuals present in the genind object.

Note

When percent = TRUE, this is exactly the same as provesti.dist, except that it performs better for large numbers of individuals (n > 125) at the cost of available memory.

Author(s)

Zhian N. Kamvar

See Also

prevosti.dist, bitwise.dist (for SNP data)

Examples

```
# A simple example. Let's analyze the mean distance among populations of A.
# euteiches.

data(Aeut)
mean(diss.dist(popsub(Aeut, 1)))
## Not run:
mean(diss.dist(popsub(Aeut, 2)))
mean(diss.dist(Aeut))

## End(Not run)
```

`diversity_boot` *Perform a bootstrap analysis on diversity statistics*

Description

This function is intended to perform bootstrap statistics on a matrix of multilocus genotype counts in different populations. Results from this function should be interpreted carefully as the default statistics are known to have a downward bias. See the details for more information.

Usage

```
diversity_boot(
  tab,
  n,
  n.boot = 1L,
  n.rare = NULL,
  H = TRUE,
  G = TRUE,
  lambda = TRUE,
  E5 = TRUE,
  ...
)
```

Arguments

`tab`	a table produced from the **poppr** function `mlg.table()`. MLGs in columns and populations in rows
`n`	an integer > 0 specifying the number of bootstrap replicates to perform (corresponds to `R` in the function `boot::boot()`.
`n.boot`	an integer specifying the number of samples to be drawn in each bootstrap replicate. If `n.boot` < 2 (default), the number of samples drawn for each bootstrap replicate will be equal to the number of samples in the data set.
`n.rare`	a sample size at which all resamplings should be performed. This should be no larger than the smallest sample size. Defaults to `NULL`, indicating that each population will be sampled at its own size.

H	logical whether or not to calculate Shannon's index
G	logical whether or not to calculate Stoddart and Taylor's index (aka inverse Simpson's index).
lambda	logical whether or not to calculate Simpson's index
E5	logical whether or not to calculate Evenness
...	other parameters passed on to boot::boot() and diversity_stats().

Details

Bootstrapping is performed in three ways:

- if n.rare is a number greater than zero, then bootstrapping is performed by randomly sampling without replacement *n.rare* samples from the data.

```
\item if `n.boot` is greater than 1, bootstrapping is performed by
sampling n.boot samples from a multinomial distribution weighted by the
proportion of each MLG in the data.

\item if `n.boot` is less than 2, bootstrapping is performed by
sampling N samples from a multinomial distribution weighted by the
proportion of each MLG in the data.
```

Downward Bias: When sampling with replacement, the diversity statistics here present a downward bias partially due to the small number of samples in the data. The result is that the mean of the bootstrapped samples will often be much lower than the observed value. Alternatively, you can increase the sample size of the bootstrap by increasing the size of n.boot. Both of these methods should be taken with caution in interpretation. There are several R packages freely available that will calculate and perform bootstrap estimates of Shannon and Simpson diversity metrics (eg. **entropart**, **entropy**, **simboot**, and **EntropyEstimation**. These packages also offer unbiased estimators of Shannon and Simpson diversity. Please take care when attempting to interpret the results of this function.

Value

a list of objects of class "boot".

Author(s)

Zhian N. Kamvar

See Also

diversity_stats() for basic statistic calculation, diversity_ci() for confidence intervals and plotting, and poppr(). For bootstrap sampling: stats::rmultinom() boot::boot()

Examples

```
library(poppr)
data(Pinf)
tab <- mlg.table(Pinf, plot = FALSE)
diversity_boot(tab, 10L)
## Not run:
# This can be done in a parallel fashion (OSX uses "multicore", Windows uses "snow")
system.time(diversity_boot(tab, 10000L, parallel = "multicore", ncpus = 4L))
system.time(diversity_boot(tab, 10000L))

## End(Not run)
```

---

diversity_ci *Perform bootstrap statistics, calculate, and plot confidence intervals.*

Description

This function is for calculating bootstrap statistics and their confidence intervals. It is important to note that the calculation of confidence intervals is not perfect (See Details). Please be cautious when interpreting the results.

Usage

```
diversity_ci(
  tab,
  n = 1000,
  n.boot = 1L,
  ci = 95,
  total = TRUE,
  rarefy = FALSE,
  n.rare = 10,
  plot = TRUE,
  raw = TRUE,
  center = TRUE,
  ...
)
```

Arguments

tab	a adegenet::genind(), genclone(), snpclone(), OR a matrix produced from mlg.table().
n	an integer defining the number of bootstrap replicates (defaults to 1000).
n.boot	an integer specifying the number of samples to be drawn in each bootstrap replicate. If n.boot < 2 (default), the number of samples drawn for each bootstrap replicate will be equal to the number of samples in the data set. See Details.
ci	the percent for confidence interval.

total	argument to be passed on to mlg.table() if tab is a genind object.
rarefy	if TRUE, bootstrapping will be performed on the smallest population size or the value of n.rare, whichever is larger. Defaults to FALSE, indicating that bootstrapping will be performed respective to each population size.
n.rare	an integer specifying the smallest size at which to resample data. This is only used if rarefy = TRUE.
plot	If TRUE (default), boxplots will be produced for each population, grouped by statistic. Colored dots will indicate the observed value.This plot can be retrieved by using p <- last_plot() from the **ggplot2** package.
raw	if TRUE (default) a list containing three elements will be returned
center	if TRUE (default), the confidence interval will be centered around the observed statistic. Otherwise, if FALSE, the confidence interval will be bias-corrected normal CI as reported from boot::boot.ci()
...	parameters to be passed on to boot::boot() and diversity_stats()

Details

Bootstrapping: For details on the bootstrapping procedures, see diversity_boot(). Default bootstrapping is performed by sampling **N** samples from a multinomial distribution weighted by the relative multilocus genotype abundance per population where **N** is equal to the number of samples in the data set. If **n.boot** > 2, then **n.boot** samples are taken at each bootstrap replicate. When rarefy = TRUE, then samples are taken at the smallest population size without replacement. This will provide confidence intervals for all but the smallest population.

Confidence intervals: Confidence intervals are derived from the function boot::norm.ci(). This function will attempt to correct for bias between the observed value and the bootstrapped estimate. When center = TRUE (default), the confidence interval is calculated from the bootstrapped distribution and centered around the bias-corrected estimate as prescribed in Marcon (2012). This method can lead to undesirable properties, such as the confidence interval lying outside of the maximum possible value. For rarefaction, the confidence interval is simply determined by calculating the percentiles from the bootstrapped distribution. If you want to calculate your own confidence intervals, you can use the results of the permutations stored in the $boot element of the output.

Rarefaction: Rarefaction in the sense of this function is simply sampling a subset of the data at size **n.rare**. The estimates derived from this method have straightforward interpretations and allow you to compare diversity across populations since you are controlling for sample size.

Plotting: Results are plotted as boxplots with point estimates. If there is no rarefaction applied, confidence intervals are displayed around the point estimates. The boxplots represent the actual values from the bootstrapping and will often appear below the estimates and confidence intervals.

Value

raw = TRUE:

- **obs** a matrix with observed statistics in columns, populations in rows
- **est** a matrix with estimated statistics in columns, populations in rows

- **CI** an array of 3 dimensions giving the lower and upper bound, the index measured, and the population.
- **boot** a list containing the output of `boot::boot()` for each population.

raw = FALSE: a data frame with the statistic observations, estimates, and confidence intervals in columns, and populations in rows. Note that the confidence intervals are converted to characters and rounded to three decimal places.

Note

Confidence interval calculation: Almost all of the statistics supplied here have a maximum when all genotypes are equally represented. This means that bootstrapping the samples will always be downwardly biased. In many cases, the confidence intervals from the bootstrapped distribution will fall outside of the observed statistic. The reported confidence intervals here are reported by assuming the variance of the bootstrapped distribution is the same as the variance around the observed statistic. As different statistics have different properties, there will not always be one clear method for calculating confidence intervals. A suggestion for correction in Shannon's index is to center the CI around the observed statistic (Marcon, 2012), but there are theoretical limitations to this. For details, see https://stats.stackexchange.com/q/156235/49413.

User-defined functions: While it is possible to use custom functions with this, there are three important things to remember when using these functions:

```
1. The function must return a single value.
2. The function must allow for both matrix and vector inputs
3. The function name cannot match or partially match any arguments
from [boot::boot()]
```

Anonymous functions are okay
(e.g. `function(x) vegan::rarefy(t(as.matrix(x)), 10)`).

Author(s)

Zhian N. Kamvar

References

Marcon, E., Herault, B., Baraloto, C. and Lang, G. (2012). The Decomposition of Shannon's Entropy and a Confidence Interval for Beta Diversity. *Oikos* 121(4): 516-522.

See Also

`diversity_boot()` `diversity_stats()` `poppr()` `boot::boot()` `boot::norm.ci()` `boot::boot.ci()`

Examples

```
library(poppr)
data(Pinf)
diversity_ci(Pinf, n = 100L)
## Not run:
# With pretty results
```

```
diversity_ci(Pinf, n = 100L, raw = FALSE)

# This can be done in a parallel fasion (OSX uses "multicore", Windows uses "snow")
system.time(diversity_ci(Pinf, 10000L, parallel = "multicore", ncpus = 4L))
system.time(diversity_ci(Pinf, 10000L))

# We often get many requests for a clonal fraction statistic. As this is
# simply the number of observed MLGs over the number of samples, we
# recommended that people calculate it themselves. With this function, you
# can add it in:

CF <- function(x){
 x <- drop(as.matrix(x))
 if (length(dim(x)) > 1){
   res <- rowSums(x > 0)/rowSums(x)
 } else {
   res <- sum(x > 0)/sum(x)
 }
 return(res)
}
# Show pretty results

diversity_ci(Pinf, 1000L, CF = CF, center = TRUE, raw = FALSE)
diversity_ci(Pinf, 1000L, CF = CF, rarefy = TRUE, raw = FALSE)

## End(Not run)
```

`diversity_stats` *Produce a table of diversity statistics*

Description

Calculate diversity statistics on a matrix containing counts of multilocus genotypes per population.

Usage

```
diversity_stats(z, H = TRUE, G = TRUE, lambda = TRUE, E5 = TRUE, ...)
```

Arguments

`z`	a table of integers representing counts of MLGs (columns) per population (rows)
`H`	logical whether or not to calculate Shannon's index
`G`	logical whether or not to calculate Stoddart and Taylor's index (aka inverse Simpson's index).
`lambda`	logical whether or not to calculate Simpson's index
`E5`	logical whether or not to calculate Evenness
`...`	any functions that can be calculated on a vector or matrix of genotype counts.

Details

This function will calculate any diversity statistic for counts of multilocus genotypes per population. This does not count allelic diversity. The calculations of H, G, and lambda are all performed by `vegan::diversity()`. E5 is calculated as

$$E_5 = \frac{(1/\lambda) - 1}{e^H - 1}$$

.

Value

a numeric matrix giving statistics (columns) for each population (rows).

Author(s)

Zhian N. Kamvar

See Also

`diversity_boot()` `diversity_ci()` `poppr()`

Examples

```
library(poppr)
data(Pinf)
tab <- mlg.table(Pinf, plot = FALSE)
diversity_stats(tab)
## Not run:
# Example using the poweRlaw package to calculate the negative slope of the
# Pareto distribution.

library("poweRlaw")
power_law_beta <- function(x){
  xpow <- displ(x[x > 0])                 # Generate the distribution
  xpow$setPars(estimate_pars(xpow))       # Estimate the parameters
  xdat <- plot(xpow, draw = FALSE)        # Extract the data
  xlm <- lm(log(y) ~ log(x), data = xdat) # Run log-log linear model for slope
  return(-coef(xlm)[2])
}

Beta <- function(x){
  x <- drop(as.matrix(x))
  if (length(dim(x)) > 1){
    res <- apply(x, 1, power_law_beta)
  } else {
    res <- power_law_beta(x)
  }
  return(res)
}

diversity_stats(tab, B = Beta)
```

```
## End(Not run)
```

`filter_stats` *Utilize all algorithms of mlg.filter*

Description

This function is a wrapper to mlg.filter. It will calculate all of the stats for mlg.filter utilizing all of the algorithms.

Usage

```
filter_stats(
  x,
  distance = bitwise.dist,
  threshold = 1e+06 + .Machine$double.eps^0.5,
  stats = "All",
  missing = "ignore",
  plot = FALSE,
  cols = NULL,
  nclone = NULL,
  hist = "Scott",
  threads = 1L,
  ...
)
```

Arguments

`x`	a `genind`, `genclone`, `genlight`, or `snpclone` object
`distance`	a distance function or matrix
`threshold`	a threshold to be passed to `mlg.filter` (Default: 1e6)
`stats`	what statistics should be calculated.
`missing`	how to treat missing data with mlg.filter
`plot`	If the threshold is a maximum threshold, should the statistics be plotted (Figure 2)
`cols`	the colors to use for each algorithm (defaults to set1 of **RColorBrewer**).
`nclone`	the number of multilocus genotypes you expect for the data. This will draw horizontal line on the graph at the value nclone and then vertical lines showing the cutoff thresholds for each algorithm.
`hist`	if you want a histogram to be plotted behind the statistics, select a method here. Available methods are "sturges", "fd", or "scott" (default) as documented in `hist`. If you don't want to plot the histogram, set `hist = NULL`.
`threads`	(unused) Previously the number of threads to be used. As of poppr version 2.4.1, this is by default set to 1.
`...`	extra parameters passed on to the distance function.

Value

a list of results from mlg.filter from the three algorithms. (returns invisibly if `plot = TRUE`)

Note

This function originally appeared in doi:10.5281/zenodo.17424

Author(s)

Zhian N. Kamvar, Jonah C. Brooks

References

ZN Kamvar, JC Brooks, and NJ Grünwald. 2015. Supplementary Material for Frontiers Plant Genetics and Genomics 'Novel R tools for analysis of genome-wide population genetic data with emphasis on clonality'. DOI: doi:10.5281/zenodo.17424

Kamvar ZN, Brooks JC and Grünwald NJ (2015) Novel R tools for analysis of genome-wide population genetic data with emphasis on clonality. Front. Genet. 6:208. doi: doi:10.3389/fgene.2015.00208

See Also

`mlg.filter` `cutoff_predictor` `bitwise.dist` `diss.dist`

Examples

```
# Basic usage example: Bruvo's Distance --------------------------------
data(Pinf)
pinfreps <- fix_replen(Pinf, c(2, 2, 6, 2, 2, 2, 2, 2, 3, 3, 2))
bres <- filter_stats(Pinf, distance = bruvo.dist, replen = pinfreps, plot = TRUE, threads = 1L)
print(bres) # shows all of the statistics

# Use these results with cutoff_filter()
print(thresh <- cutoff_predictor(bres$farthest$THRESHOLDS))
mlg.filter(Pinf, distance = bruvo.dist, replen = pinfreps) <- thresh
Pinf

# Different distances will give different results -----------------------
nres <- filter_stats(Pinf, distance = nei.dist, plot = TRUE, threads = 1L, missing = "mean")
print(thresh <- cutoff_predictor(nres$farthest$THRESHOLDS))
mlg.filter(Pinf, distance = nei.dist, missing = "mean") <- thresh
Pinf
```

`fix_replen` *Find and fix inconsistent repeat lengths*

Description

Attempts to fix inconsistent repeat lengths found by `test_replen`

Usage

```
fix_replen(gid, replen, e = 1e-05, fix_some = TRUE)
```

Arguments

`gid`	a `genind` or `genclone` object
`replen`	a numeric vector of repeat motif lengths.
`e`	a number to be subtracted or added to inconsistent repeat lengths to allow for proper rounding.
`fix_some`	if `TRUE` (default), when there are inconsistent repeat lengths that cannot be fixed by subtracting or adding e, those than can be fixed will. If `FALSE`, the original repeat lengths will not be fixed.

Details

This function is modified from the version used in doi:10.5281/zenodo.13007.
Before being fed into the algorithm to calculate Bruvo's distance, the amplicon length is divided by the repeat unit length. Because of the amplified primer sequence attached to sequence repeat, this division does not always result in an integer and so the resulting numbers are rounded. The rounding also protects against slight mis-calls of alleles. Because we know that

$$\frac{(A-e)-(B-e)}{r}$$

is equivalent to

$$\frac{A-B}{r}$$

, we know that the primer sequence will not alter the relationships between the alleles. Unfortunately for nucleotide repeats that have powers of 2, rounding in R is based off of the IEC 60559 standard (see `round`), that means that any number ending in 5 is rounded to the nearest *even* digit. This function will attempt to alleviate this problem by adding a very small amount to the repeat length so that division will not result in a 0.5. If this fails, the same amount will be subtracted. If neither of these work, a warning will be issued and it is up to the user to determine if the fault is in the allele calls or the repeat lengths.

Value

a numeric vector of corrected repeat motif lengths.

Author(s)

Zhian N. Kamvar

References

Zhian N. Kamvar, Meg M. Larsen, Alan M. Kanaskie, Everett M. Hansen, & Niklaus J. Grünwald. Sudden_Oak_Death_in_Oregon_Forests: Spatial and temporal population dynamics of the sudden oak death epidemic in Oregon Forests. ZENODO, doi:10.5281/zenodo.13007, 2014.

Kamvar, Z. N., Larsen, M. M., Kanaskie, A. M., Hansen, E. M., & Grünwald, N. J. (2015). Spatial and temporal analysis of populations of the sudden oak death pathogen in Oregon forests. Phytopathology 105:982-989. doi: doi:10.1094/PHYTO12140350FI

Ruzica Bruvo, Nicolaas K. Michiels, Thomas G. D'Souza, and Hinrich Schulenburg. A simple method for the calculation of microsatellite genotype distances irrespective of ploidy level. Molecular Ecology, 13(7):2101-2106, 2004.

See Also

`test_replen` `bruvo.dist` `bruvo.msn` `bruvo.boot`

Examples

```
data(Pram)
(Pram_replen <- setNames(c(3, 2, 4, 4, 4), locNames(Pram)))
fix_replen(Pram, Pram_replen)
# Let's start with an example of a tetranucleotide repeat motif and imagine
# that there are twenty alleles all 1 step apart:
(x <- 1:20L * 4L)
# These are the true lengths of the different alleles. Now, let's add the
# primer sequence to them.
(PxP <- x + 21 + 21)
# Now we make sure that x / 4 is equal to 1:20, which we know each have
# 1 difference.
x/4
# Now, we divide the sequence with the primers by 4 and see what happens.
(PxPc <- PxP/4)
(PxPcr <- round(PxPc))
diff(PxPcr) # we expect all 1s

# Let's try that again by subtracting a tiny amount from 4
(PxPc <- PxP/(4 - 1e-5))
(PxPcr <- round(PxPc))
diff(PxPcr)
```

`genclone-class` *GENclone and SNPclone classes*

Description

GENclone is an S4 class that extends the genind object.
SNPclone is an S4 class that extends the genlight object.

They will have all of the same attributes as their parent classes, but they will contain one extra slot to store extra information about multilocus genotypes.

Details

The genclone and snpclone classes will allow for more optimized methods of clone correction.

Previously for genind and genlight objects, multilocus genotypes were not retained after a data set was subset by population. The new `mlg` slot allows us to assign the multilocus genotypes and retain that information no matter how we subset the data set. This new slot can either contain numeric values for multilocus genotypes OR it can contain a special internal `MLG` class that allows for custom multilocus genotype definitions and filtering.

Slots

`mlg` a vector representing multilocus genotypes for the data set OR an object of class `MLG`.

Extends

The `genclone` class extends class `"genind"`, directly.
The `snpclone` class extends class `"genlight"`, directly.

Note

When calculating multilocus genotypes for genclone objects, a rank function is used, but calculation of multilocus genotypes for snpclone objects is distance-based (via `bitwise.dist` and `mlg.filter`). This means that genclone objects are sensitive to missing data, whereas snpclone objects are insensitive.

Author(s)

Zhian N. Kamvar

See Also

`as.genclone` `as.snpclone` `genind` `genlight` `strata` `setPop` `MLG` `mll` `mlg.filter`

Examples

```
## Not run:

# genclone objects can be created from genind objects
#
data(partial_clone)
partial_clone
(pc <- as.genclone(partial_clone))
```

```
# snpclone objects can be created from genlight objects
#
set.seed(999)
(gl <- glSim(100, 0, n.snp.struc = 1e3, ploidy = 2, parallel = FALSE))
(sc <- as.snpclone(rbind(gl, gl, parallel = FALSE), parallel = FALSE))
#
# Use mlg.filter to create a distance threshold to define multilocus genotypes.
mlg.filter(sc, threads = 1L) <- 0.25
sc # 82 mlgs


## End(Not run)
```

---

genind2genalex *Export data from genind objects to genalex formatted *.csv files.*

Description

genind2genalex will export a genclone or genind object to a *.csv file formatted for use in genalex.

Usage

```
genind2genalex(
  gid,
  filename = "",
  overwrite = FALSE,
  quiet = FALSE,
  pop = NULL,
  allstrata = TRUE,
  geo = FALSE,
  geodf = "xy",
  sep = ",",
  sequence = FALSE
)
```

Arguments

gid	a genclone or adegenet::genind object.
filename	a string indicating the name and/or path of the file you wish to create. If this is left unchanged, the results will be saved in a temporary file and a warning will be displayed for six seconds before the file is written. This process should give you time to cancel the process and choose a file path. Otherwise, the name of the file is returned, so you can copy that to a file of your choice with `file.copy()`
overwrite	`logical` if `FALSE` (default) and `filename` exists, then the file will not be overwritten. Set this option to `TRUE` to overwrite the file.
quiet	`logical` If `FALSE` a message will be printed to the screen.

pop	a character vector OR formula specifying the population factor. This can be used to specify a specific subset of strata or custom population factor for the output. Note that the `allstrata` command has precedence over this unless the value of this is a new population factor.
allstrata	if this is TRUE, the strata will be combined into a single population factor in the genalex file.
geo	logical Default is FALSE. If it is set to TRUE, the resulting file will have two columns for geographic data.
geodf	character Since the other slot in the adegenet object can contain many different items, you must specify the name of the data frame in the other slot containing your geographic coordinates. It defaults to "xy".
sep	a character specifying what character to use to separate columns. Defaults to ",".
sequence	when TRUE, sequence data will be converted to integers as per the GenAlEx specifications.

Value

The the file path or connection where the data were written.

Note

If your data set lacks a population structure, it will be coded in the new file as a single population labeled "Pop". Likewise, if you don't have any labels for your individuals, they will be labeled as "ind1" through "ind*N*", with *N* being the size of your population.

Author(s)

Zhian N. Kamvar

See Also

read.genalex(), clonecorrect(), genclone, adegenet::genind

Examples

```
## Not run:
data(nancycats)
genind2genalex(nancycats, "~/Documents/nancycats.csv", geo=TRUE)

## End(Not run)
```

genotype_curve *Produce a genotype accumulation curve*

Description

Genotype accumulation curves are useful for determining the minimum number of loci necessary to discriminate between individuals in a population. This function will randomly sample loci without replacement and count the number of multilocus genotypes observed.

Usage

```
genotype_curve(
  gen,
  sample = 100,
  maxloci = 0L,
  quiet = FALSE,
  thresh = 1,
  plot = TRUE,
  drop = TRUE,
  dropna = TRUE
)
```

Arguments

gen	a genclone, genind, or loci object.
sample	an integer defining the number of times loci will be resampled without replacement.
maxloci	the maximum number of loci to sample. By default, maxloci = 0, which indicates that n - 1 loci are to be used. Note that this will always take min(n - 1, maxloci)
quiet	if FALSE (default), Progress of the iterations will be displayed. If TRUE, nothing is printed to screen as the function runs.
thresh	a number from 0 to 1. This will draw a line at that fraction of multilocus genotypes, rounded. Defaults to 1, which will draw a line at the maximum number of observable genotypes.
plot	if TRUE (default), the genotype curve will be plotted via ggplot2. If FALSE, the resulting matrix will be visibly returned.
drop	if TRUE (default), monomorphic loci will be removed before analysis as these loci affect the shape of the curve.
dropna	if TRUE (default) and drop = TRUE, NAs will be ignored when determining if a locus is monomorphic. When FALSE, presence of NAs will result in the locus being retained. This argument has no effect when drop = FALSE

Details

Internally, this function works by converting the data into a loci object, which represents genotypes as a data frame of factors. Random samples are taken of 1 to n-1 columns of the matrix and the number of unique rows are counted to determine the number of multilocus genotypes in that random sample. This function does not take into account any definitions of MLGs via mlg.filter or mll.custom.

Value

(invisibly by deafuls) a matrix of integers showing the results of each randomization. Columns represent the number of loci sampled and rows represent an independent sample.

Author(s)

Zhian N. Kamvar

Examples

```
data(nancycats)
nan_geno <- genotype_curve(nancycats)
## Not run:

# Marker Type Comparison ----------------------------------------------------
# With AFLP data, it is often necessary to include more markers for resolution
data(Aeut)
Ageno <- genotype_curve(Aeut)

# Many microsatellite data sets have hypervariable markers
data(microbov)
mgeno <- geotype_curve(microbov)

# Adding a trendline --------------------------------------------------------

# Trendlines: you can add a smoothed trendline with geom_smooth()
library("ggplot2")
p <- last_plot()
p + geom_smooth()

# Producing Figures for Publication -----------------------------------------

# This data set has been pre filtered
data(monpop)
mongeno <- genotype_curve(monpop)

# Here, we add a curve and a title for publication
p <- last_plot()
mytitle <- expression(paste("Genotype Accumulation Curve for ",
                            italic("M. fructicola")))
p + geom_smooth() +
  theme_bw() +
  theme(text = element_text(size = 12, family = "serif")) +
```

```
  theme(title = element_text(size = 14)) +
  ggtitle(mytitle)

## End(Not run)
```

---

getfile *Get a file name and path and store them in a list.*

Description

getfile is a convenience function that serves as a wrapper for the functions `file.choose()`, `file.path()`, and `list.files()`. If the user is working in a GUI environment, a window will pop up, allowing the user to choose a specified file regardless of path.

Usage

```
getfile(multi = FALSE, pattern = NULL, combine = TRUE)
```

Arguments

`multi`	this is an indicator to allow the user to store the names of multiple files found in the directory. This is useful in conjunction with `poppr.all()`.
`pattern`	a `regex()` pattern for use while `multi == TRUE`. This will grab all files matching this pattern.
`combine`	`logical`. When this is set to `TRUE` (default), the `$files` vector will have the path appended to them. When it is set to `FALSE`, it will have the basename.

Value

`path`	a character string of the absolute path to the chosen file or files
`files`	a character vector containing the chosen file name or names.

Author(s)

Zhian N. Kamvar

Examples

```
## Not run:

x <- getfile()
poppr(x$files)

y <- getfile(multi=TRUE, pattern="^.+?dat$")
#useful for reading in multiple FSTAT formatted files.

yfiles <- poppr.all(y$files)
```

```
## End(Not run)
```

---

greycurve *Display a greyscale gradient adjusted to specific parameters*

Description

This function has one purpose. It is for deciding the appropriate scaling for a grey palette to be used for edge weights of a minimum spanning network.

Usage

```
greycurve(
  data = seq(0, 1, length = 1000),
  glim = c(0, 0.8),
  gadj = 3,
  gweight = 1,
  scalebar = FALSE
)
```

Arguments

`data`	a sequence of numbers to be converted to greyscale.
`glim`	"grey limit". Two numbers between zero and one. They determine the upper and lower limits for the `gray` function. Default is 0 (black) and 0.8 (20% black).
`gadj`	"grey adjust". a positive `integer` greater than zero that will serve as the exponent to the edge weight to scale the grey value to represent that weight.
`gweight`	"grey weight". an `integer`. If it's 1, the grey scale will be weighted to emphasize the differences between closely related nodes. If it is 2, the grey scale will be weighted to emphasize the differences between more distantly related nodes.
`scalebar`	When this is set to `TRUE`, two scalebars will be plotted. The purpose of this is for adding a scale bar to minimum spanning networks produced in earlier versions of poppr.

Value

A plot displaying a grey gradient from 0.001 to 1 with minimum and maximum values displayed as yellow lines, and an equation for the correction displayed in red.

Author(s)

Zhian N. Kamvar

Examples

```
# Normal grey curve with an adjustment of 3, an upper limit of 0.8, and
# weighted towards smaller values.
greycurve()
## Not run:
# 1:1 relationship grey curve.
greycurve(gadj=1, glim=1:0)

# Grey curve weighted towards larger values.
greycurve(gweight=2)

# Same as the first, but the limit is 1.
greycurve(glim=1:0)

# Setting the lower limit to 0.1 and weighting towards larger values.
greycurve(glim=c(0.1,0.8), gweight=2)

## End(Not run)
```

---

ia *Index of Association*

Description

Calculate the Index of Association and Standardized Index of Association.

Usage

```
ia(
  gid,
  sample = 0,
  method = 1,
  quiet = FALSE,
  missing = "ignore",
  plot = TRUE,
  hist = TRUE,
  index = "rbarD",
  valuereturn = FALSE
)

pair.ia(
  gid,
  sample = 0L,
  quiet = FALSE,
  plot = TRUE,
  low = "blue",
  high = "red",
  limits = NULL,
```

```
  index = "rbarD",
  method = 1L
)

resample.ia(gid, n = NULL, reps = 999, quiet = FALSE, use_psex = FALSE, ...)

jack.ia(gid, n = NULL, reps = 999, quiet = FALSE)
```

Arguments

gid	a adegenet::genind() or genclone() object.
sample	an integer indicating the number of permutations desired (eg 999).
method	an integer from 1 to 4 indicating the sampling method desired. see shufflepop() for details.
quiet	Should the function print anything to the screen while it is performing calculations? TRUE prints nothing. FALSE (default) will print the population name and progress bar.
missing	a character string. see missingno() for details.
plot	When TRUE (default), a heatmap of the values per locus pair will be plotted (for pair.ia()). When sampling > 0, different things happen with ia() and pair.ia(). For ia(), a histogram for the data set is plotted. For pair.ia(), p-values are added as text on the heatmap.
hist	logical Deprecated. Use plot.
index	character either "Ia" or "rbarD". If hist = TRUE, this indicates which index you want represented in the plot (default: "rbarD").
valuereturn	logical if TRUE, the index values from the reshuffled data is returned. If FALSE (default), the index is returned with associated p-values in a 4 element numeric vector.
low	(for pair.ia) a color to use for low values when 'plot = TRUE'
high	(for pair.ia) a color to use for low values when 'plot = TRUE'
limits	(for pair.ia) the limits to be used for the color scale. Defaults to 'NULL'. If you want to use a custom range, supply two numbers between -1 and 1, (e.g. 'limits = c(-0.15, 1)')
n	an integer specifying the number of samples to be drawn. Defaults to NULL, which then uses the number of multilocus genotypes.
reps	an integer specifying the number of replicates to perform. Defaults to 999.
use_psex	a logical. If TRUE, the samples will be weighted by the value of psex. Defaults to FALSE.
...	arguments passed on to psex

Details

- ia() calculates the index of association over all loci in the data set.
- pair.ia() calculates the index of association in a pairwise manner among all loci.

- `resample.ia()` calculates the index of association on a reduced data set multiple times to create a distribution, showing the variation of values observed at a given sample size (previously `jack.ia()`).

The index of association was originally developed by A.H.D. Brown analyzing population structure of wild barley (Brown, 1980). It has been widely used as a tool to detect clonal reproduction within populations . Populations whose members are undergoing sexual reproduction, whether it be selfing or out-crossing, will produce gametes via meiosis, and thus have a chance to shuffle alleles in the next generation. Populations whose members are undergoing clonal reproduction, however, generally do so via mitosis. This means that the most likely mechanism for a change in genotype is via mutation. The rate of mutation varies from species to species, but it is rarely sufficiently high to approximate a random shuffling of alleles. The index of association is a calculation based on the ratio of the variance of the raw number of differences between individuals and the sum of those variances over each locus . You can also think of it as the observed variance over the expected variance. If they are the same, then the index is zero after subtracting one (from Maynard-Smith, 1993):

$$I_A = \frac{V_O}{V_E} - 1$$

Since the distance is more or less a binary distance, any sort of marker can be used for this analysis. In the calculation, phase is not considered, and any difference increases the distance between two individuals. Remember that each column represents a different allele and that each entry in the table represents the fraction of the genotype made up by that allele at that locus. Notice also that the sum of the rows all equal one. Poppr uses this to calculate distances by simply taking the sum of the absolute values of the differences between rows.

The calculation for the distance between two individuals at a single locus with *a* allelic states and a ploidy of *k* is as follows (except for Presence/Absence data):

$$d = \frac{k}{2} \sum_{i=1}^{a} \mid A_i - B_i \mid$$

To find the total number of differences between two individuals over all loci, you just take *d* over *m* loci, a value we'll call *D*:

$$D = \sum_{i=1}^{m} d_i$$

These values are calculated over all possible combinations of individuals in the data set, $\binom{n}{2}$ after which you end up with $\binom{n}{2} \cdot m$ values of *d* and $\binom{n}{2}$ values of *D*. Calculating the observed variances is fairly straightforward (modified from Agapow and Burt, 2001):

$$V_O = \frac{\sum_{i=1}^{\binom{n}{2}} D_i^2 - \frac{(\sum_{i=1}^{\binom{n}{2}} D_i)^2}{\binom{n}{2}}}{\binom{n}{2}}$$

Calculating the expected variance is the sum of each of the variances of the individual loci. The calculation at a single locus, *j* is the same as the previous equation, substituting values of *D* for *d*:

$$var_j = \frac{\sum_{i=1}^{\binom{n}{2}} d_i^2 - \frac{(\sum_{i=1}^{\binom{n}{2}} d_i)^2}{\binom{n}{2}}}{\binom{n}{2}}$$

The expected variance is then the sum of all the variances over all *m* loci:

$$V_E = \sum_{j=1}^{m} var_j$$

Agapow and Burt showed that I_A increases steadily with the number of loci, so they came up with an approximation that is widely used, $\bar{r}_d$. For the derivation, see the manual for *multilocus*.

$$\bar{r}_d = \frac{V_O - V_E}{2\sum_{j=1}^{m}\sum_{k\neq j}^{m}\sqrt{var_j \cdot var_k}}$$

Value

for `pair.ia()`:

A matrix with two columns and choose(nLoc(gid), 2) rows representing the values for Ia and rbarD per locus pair.

If no sampling has occurred::

A named number vector of length 2 giving the Index of Association, "Ia"; and the Standardized Index of Association, "rbarD"

If there is sampling::

A a named numeric vector of length 4 with the following values:

- Ia - numeric. The index of association.
- p.Ia - A number indicating the p-value resulting from a one-sided permutation test based on the number of samples indicated in the original call.
- rbarD - numeric. The standardized index of association.
- p.rD - A factor indicating the p-value resulting from a one-sided permutation test based on the number of samples indicated in the original call.

If there is sampling and `valureturn = TRUE`:

A list with the following elements:

- index The above vector
- samples A data frame with s by 2 column data frame where s is the number of samples defined. The columns are for the values of Ia and rbarD, respectively.

resample.ia(): a data frame with the index of association and standardized index of association in columns. Number of rows represents the number of reps.

Note

jack.ia() is deprecated as the name was misleading. Please use resample.ia()

Author(s)

Zhian N. Kamvar

References

Paul-Michael Agapow and Austin Burt. Indices of multilocus linkage disequilibrium. *Molecular Ecology Notes*, 1(1-2):101-102, 2001

A.H.D. Brown, M.W. Feldman, and E. Nevo. Multilocus structure of natural populations of *Hordeum spontaneum*. *Genetics*, 96(2):523-536, 1980.

J M Smith, N H Smith, M O'Rourke, and B G Spratt. How clonal are bacteria? Proceedings of the National Academy of Sciences, 90(10):4384-4388, 1993.

See Also

poppr(), missingno(), adegenet::import2genind(), read.genalex(), clonecorrect(), win.ia(), samp.ia()

Examples

```
data(nancycats)
ia(nancycats)

# Pairwise over all loci:
data(partial_clone)
res <- pair.ia(partial_clone)
plot(res, low = "black", high = "green", index = "Ia")

# Resampling
data(Pinf)
resample.ia(Pinf, reps = 99)

## Not run:

# Pairwise IA with p-values (this will take about a minute)
res <- pair.ia(partial_clone, sample = 999)
head(res)

# Plot the results of resampling rbarD.
library("ggplot2")
Pinf.resamp <- resample.ia(Pinf, reps = 999)
ggplot(Pinf.resamp[2], aes(x = rbarD)) +
  geom_histogram() +
  geom_vline(xintercept = ia(Pinf)[2]) +
  geom_vline(xintercept = ia(clonecorrect(Pinf))[2], linetype = 2) +
  xlab(expression(bar(r)[d]))
```

```
# Get the indices back and plot the distributions.
nansamp <- ia(nancycats, sample = 999, valuereturn = TRUE)

plot(nansamp, index = "Ia")
plot(nansamp, index = "rbarD")

# You can also adjust the parameters for how large to display the text
# so that it's easier to export it for publication/presentations.
library("ggplot2")
plot(nansamp, labsize = 5, linesize = 2) +
  theme_bw() +                                      # adding a theme
  theme(text = element_text(size = rel(5))) +       # changing text size
  theme(plot.title = element_text(size = rel(4))) + # changing title size
  ggtitle("Index of Association of nancycats")      # adding a new title

# Get the index for each population.
lapply(seppop(nancycats), ia)
# With sampling
lapply(seppop(nancycats), ia, sample = 999)

# Plot pairwise ia for all populations in a grid with cowplot
# Set up the library and data
library("cowplot")
data(monpop)
splitStrata(monpop) <- ~Tree/Year/Symptom
setPop(monpop)      <- ~Tree

# Need to set up a list in which to store the plots.
plotlist        <- vector(mode = "list", length = nPop(monpop))
names(plotlist) <- popNames(monpop)

# Loop throgh the populations, calculate pairwise ia, plot, and then
# capture the plot in the list
for (i in popNames(monpop)){
  x <- pair.ia(monpop[pop = i], limits = c(-0.15, 1)) # subset, calculate, and plot
  plotlist[[i]] <- ggplot2::last_plot() # save the last plot
}

# Use the plot_grid function to plot.
plot_grid(plotlist = plotlist, labels = paste("Tree", popNames(monpop)))


## End(Not run)
```

`imsn` *Create minimum spanning networks interactively*

Description

This function will launch an interactive interface that allows you to create, plot, manipulate, and save minimum spanning networks. It runs using the **shiny** R package.

Usage

```
imsn()
```

Details

Creating and plotting MSNs requires three steps:

1. Create a distance matrix from your data
2. Create a minimum spanning network with your data and the matrix
3. Visualize the minimum spanning network

The function `plot_poppr_msn` is currently the most flexible way of visualizing your minimum spanning network, but with 20 parameters, it can become pretty intimidating trying to find the right display for your MSN.

With this function, all three steps are combined into one interactive interface that will allow you to intuitively modify your minimum spanning network and even save the results to a pdf or png file.

Value

NULL, invisibly

Interface

Buttons: In the left hand panel, there are three buttons to execute the functions. These allow you to run the data set after you manipulate all of the parameters.

- **GO!** - This button will start the application with the specified parameters
- **reData** - Use this button when you have changed any parameters under the section **Data Parameters**. This involves recalculating the distance matrix and msn.
- **reGraph** - Use this button when you have changed any parameters under the section **Graphical Parameters**. This involves superficial changes to the display of the minimum spanning network.

Tabs:

The right hand panel contains different tabs related to your data set of choice.

- **Plot** - The minimum spanning network itself
- **Data** - A display of your data set
- **Command** - The commands used to create the plot. You can copy and paste this to an R file for reproducibility.
- **Save Plot** - This provides a tool for you to save the plot to a PDF or PNG image.
- **Session Information** - displays the result of `sessionInfo` for reproducibility.

Author(s)

Zhian N. Kamvar

See Also

`plot_poppr_msn` `diss.dist` `bruvo.dist` `bruvo.msn` `poppr.msn` `nei.dist` `popsub` `missingno`

Examples

```
## Not run:

# Set up some data
library("poppr")
library("magrittr")
data(monpop)
splitStrata(monpop) <- ~Tree/Year/Symptom
summary(monpop)
monpop_ssr <- c(CHMFc4 = 7, CHMFc5 = 2, CHMFc12 = 4,
                SEA = 4, SED = 4, SEE = 2, SEG = 6,
                SEI = 3, SEL = 4, SEN = 2, SEP = 4,
                SEQ = 2, SER = 4)
t26 <- monpop %>% setPop(~Tree) %>% popsub("26") %>% setPop(~Year/Symptom)
t26
if (interactive()) {
  imsn() # select Bruvo's distance and enter "monpop_ssr" into the Repeat Length field.

  # It is also possible to run this from github if you are connected to the internet.
  # This allows you to access any bug fixes that may have been updated before a formal
  # release on CRAN

  shiny::runGitHub("grunwaldlab/poppr", subdir = "inst/shiny/msn_explorer")

  # You can also use your own distance matrices, but there's a small catch.
  # in order to do so, you must write a function that will subset the matrix
  # to whatever populations are in your data. Here's an example with the above

  mondist <- bruvo.dist(monpop, replen = monpop_ssr)
  myDist <- function(x, d = mondist){
   dm <- as.matrix(d)          # Convert the dist object to a square matrix
   xi <- indNames(x)           # Grab the sample names that exist
   return(as.dist(dm[xi, xi])) # return only the elements that have the names
                               # in the data set
  }
  # After executing imsn, choose:
  # Distance: custom
  # myDist
  imsn()
}

## End(Not run)
```

---

incomp *Check for samples that are incomparable due to missing data*

Description

If two samples share no loci typed in common, they are incomparable and will produce missing data in a distance matrix, which could lead to problems with further analyses. This function finds these samples and returns a matrix of how many other samples these are incomparable with.

Usage

```
incomp(gid)
```

Arguments

gid	a genind or genclone object

Value

a square matrix with samples that are incomparable

Examples

```
data(nancycats)
# These two populations have no samples that are incomparable
incomp(nancycats[pop = c(1, 17)])

# If you reduce the number of loci, we find that there are
# incomparable samples.
incomp(nancycats[pop = c(1, 17), loc = c(1, 4)])
```

---

info_table *Create a table summarizing missing data or ploidy information of a genind or genclone object*

Description

Create a table summarizing missing data or ploidy information of a genind or genclone object

Usage

```
info_table(
  gen,
  type = c("missing", "ploidy"),
  percent = TRUE,
  plot = FALSE,
  df = FALSE,
  returnplot = FALSE,
  low = "blue",
  high = "red",
  plotlab = TRUE,
  scaled = TRUE
)
```

Arguments

gen	a genind or genclone object.
type	character. What information should be returned. Choices are "missing" (Default) and "ploidy". See Description.
percent	logical. (ONLY FOR type = 'missing') If TRUE (default), table and plot will represent missing data as a percentage of each cell. If FALSE, the table and plot will represent missing data as raw counts. (See details)
plot	logical. If TRUE, a simple heatmap will be produced. If FALSE (default), no heatmap will be produced.
df	logical. If TRUE, the data will be returned as a long form data frame. If FALSE (default), a matrix with samples in rows and loci in columns will be returned.
returnplot	logical. If TRUE, a list is returned with two elements: table - the normal output and plot - the ggplot object. If FALSE, the table is returned.
low	character. What color should represent no missing data or lowest observed ploidy? (default: "blue")
high	character. What color should represent the highest amount of missing data or observed ploidy? (default: "red")
plotlab	logical. (ONLY FOR type = 'missing') If TRUE (default), values of missing data greater than 0% will be plotted. If FALSE, the plot will appear un-appended.
scaled	logical. (ONLY FOR type = 'missing') This is for when percent = TRUE. If TRUE (default), the color specified in high will represent the highest observed value of missing data. If FALSE, the color specified in high will represent 100%.

Details

Missing data is accounted for on a per-population level.
Ploidy is accounted for on a per-individual level.

For type = 'missing': This data is potentially useful for identifying areas of systematic missing data. There are a few caveats to be aware of.

- **Regarding counts of missing data**: Each count represents the number of individuals with missing data at each locus. The last column, "mean" can be thought of as the average number of individuals with missing data per locus.
- **Regarding percentage missing data**: This percentage is **relative to the population and locus**, not to the entire data set. The last column, "mean" represents the average percent of the population with missing data per locus.

For type = 'ploidy': This option is useful for data that has been imported with mixed ploidies. It will summarize the relative levels of ploidy per individual per locus. This is simply based off of observed alleles and does not provide any further estimates.

Value

a matrix, data frame (df = TRUE), or a list (returnplot = TRUE) representing missing data per population (type = 'missing') or ploidy per individual (type = 'ploidy') in a genind or genclone object.

Author(s)

Zhian N. Kamvar

Examples

```
data(nancycats)
nancy.miss <- info_table(nancycats, plot = TRUE, type = "missing")
data(Pinf)
Pinf.ploid <- info_table(Pinf, plot = TRUE, type = "ploidy")
```

`informloci` *Remove all non-phylogentically informative loci*

Description

This function will facilitate in removing phylogenetically uninformative loci from a `genclone` or `genind` object. The user has the ability to define what uninformative means by setting a cutoff value for either percentage of differentiating genotypes or minor allele frequency.

Usage

```
informloci(pop, cutoff = 2/nInd(pop), MAF = 0.01, quiet = FALSE)
```

Arguments

`pop`	a `genclone` or `genind` object.
`cutoff`	`numeric`. A number from 0 to 1 defining the minimum number of differentiating samples.
`MAF`	`numeric`. A number from 0 to 1 defining the minimum minor allele frequency. This is passed as the `thresh` parameter of `isPoly`.
`quiet`	`logical`. When `quiet = TRUE` (default), messages indicating the loci removed will be printed to screen. When `quiet = FALSE`, nothing will be printed to screen.

Details

This function will remove uninformative loci using a traditional MAF cutoff (using `isPoly` from **adegenet**) as well as analyzing the number of observed genotypes in a locus. This is important for clonal organisms that can have fixed heterozygous sites not detected by MAF methods.

Value

A `genind` object with user-defined informative loci.

Note

This will have a few side effects that affect certain analyses. First, the number of multilocus genotypes might be reduced due to the reduced number of markers (if you are only using a genind object). Second, if you plan on using this data for analysis of the index of association, be sure to use the standardized version (rbarD) that corrects for the number of observed loci.

Author(s)

Zhian N. Kamvar

Examples

```
# We will use a dummy data set to demonstrate how this detects uninformative
# loci using both MAF and a cutoff.

genos <- c("A/A", "A/B", "A/C", "B/B", "B/C", "C/C")

v <- sample(genos, 100, replace = TRUE)
w <- c(rep(genos[2], 99), genos[3]) # found by cutoff
x <- c(rep(genos[1], 98), genos[3], genos[2]) # found by MAF
y <- c(rep(genos[1], 99), genos[2]) # found by both
z <- sample(genos, 100, replace = TRUE)
dat <- df2genind(data.frame(v = v, w = w, x = x, y = y, z = z), sep = "/")

informloci(dat)

## Not run:
# Ignore MAF
informloci(dat, MAF = 0)

# Ignore cutoff
informloci(dat, cutoff = 0)

# Real data
data(H3N2)
informloci(H3N2)

## End(Not run)
```

---

is.snpclone *Check for validity of a genclone or snpclone object*

Description

Check for validity of a genclone or snpclone object

Usage

```
is.snpclone(x)

is.clone(x)

is.genclone(x)
```

Arguments

x	a genclone or snpclone object

Note

a genclone object will always be a valid genind object and a snpclone object will always be a valid genlight object.

Author(s)

Zhian N. Kamvar

Examples

```
(sc <- as.snpclone(glSim(100, 1e3, ploid=2, parallel = FALSE),
                   parallel = FALSE, n.cores = 1L))
is.snpclone(sc)
is.clone(sc)
data(nancycats)
nanclone <- as.genclone(nancycats)
is.genclone(nanclone)
```

---

locus_table *Create a table of summary statistics per locus.*

Description

Create a table of summary statistics per locus.

Usage

```
locus_table(
  x,
  index = "simpson",
  lev = "allele",
  population = "ALL",
  information = TRUE
)
```

Arguments

x	a adegenet::genind or genclone object.
index	Which diversity index to use. Choices are • "simpson" (Default) to give Simpson's index • "shannon" to give the Shannon-Wiener index • "invsimpson" to give the Inverse Simpson's index aka the Stoddard and Tayor index.
lev	At what level do you want to analyze diversity? Choices are "allele" (Default) or "genotype".
population	Select the populations to be analyzed. This is the parameter sublist passed on to the function popsub(). Defaults to "ALL".
information	When TRUE (Default), this will print out a header of information to the R console.

Value

a table with 4 columns indicating the Number of alleles/genotypes observed, Diversity index chosen, Nei's 1978 gene diversity (expected heterozygosity), and Evenness.

Note

The calculation of Hexp is $(\frac{n}{n-1})1-\sum_{i=1}^{k} p_i^2$ where p is the allele frequencies at a given locus and n is the number of observed alleles (Nei, 1978) in each locus and then returning the average. Caution should be exercised in interpreting the results of Hexp with polyploid organisms with ambiguous ploidy. The lack of allelic dosage information will cause rare alleles to be over-represented and artificially inflate the index. This is especially true with small sample sizes.

If lev = "genotype", then all statistics reflect **genotypic** diversity within each locus. This includes the calculation for Hexp, which turns into the unbiased Simpson's index.

Author(s)

Zhian N. Kamvar

References

Jari Oksanen, F. Guillaume Blanchet, Roeland Kindt, Pierre Legendre, Peter R. Minchin, R. B. O'Hara, Gavin L. Simpson, Peter Solymos, M. Henry H. Stevens, and Helene Wagner. vegan: Community Ecology Package, 2012. R package version 2.0-5.

Niklaus J. Gr\"unwald, Stephen B. Goodwin, Michael G. Milgroom, and William E. Fry. Analysis of genotypic diversity data for populations of microorganisms. Phytopathology, 93(6):738-46, 2003

J.A. Ludwig and J.F. Reynolds. Statistical Ecology. A Primer on Methods and Computing. New York USA: John Wiley and Sons, 1988.

E.C. Pielou. Ecological Diversity. Wiley, 1975.

J.A. Stoddart and J.F. Taylor. Genotypic diversity: estimation and prediction in samples. Genetics, 118(4):705-11, 1988.

Masatoshi Nei. Estimation of average heterozygosity and genetic distance from a small number of individuals. Genetics, 89(3):583-590, 1978.

Claude Elwood Shannon. A mathematical theory of communication. Bell Systems Technical Journal, 27:379-423,623-656, 1948

See Also

vegan::diversity(), poppr()

Examples

```
data(nancycats)
locus_table(nancycats[pop = 5])
## Not run:
# Analyze locus statistics for the North American population of P. infestans.
# Note that due to the unknown dosage of alleles, many of these statistics
# will be artificially inflated for polyploids.
data(Pinf)
locus_table(Pinf, population = "North America")

## End(Not run)
```

---

make_haplotypes *Split samples from a genind object into pseudo-haplotypes*

Description

Split samples from a genind object into pseudo-haplotypes

Usage

```
make_haplotypes(gid)
```

Arguments

gid	a genind or genlight object.

Details

Certain analyses, such as amova work best if within-sample variance (error) can be estimated. Practically, this is performed by splitting the genotypes across all loci to create multiple haplotypes. This way, the within-sample distance can be calculated and incorporated into the model. Please note that the haplotypes generated are based on the order of the unphased alleles in the genind object and do not represent true haplotypes.

Haploid data will be returned un-touched.

Value

a haploid genind object with an extra strata column called "Individual".

Note

The other slot will not be copied over to the new genind object.

See Also

poppr.amova() pegas::amova() as.genambig()

Examples

```
# Diploid data is doubled ------------------------------------------------

data(nancycats)
nan9 <- nancycats[pop = 9]
nan9hap <- make_haplotypes(nan9)
nan9              # 9 individuals from population 9
nan9hap           # 18 haplotypes
strata(nan9hap)   # strata gains a new column: Individual
indNames(nan9hap) # individuals are renamed sequentially


# Mix ploidy data can be split, but should be treated with caution --------
#
# For example, the Pinf data set contains 86 tetraploid individuals,
# but there appear to only be diploids and triploid genotypes. When
# we convert to haplotypes, those with all missing data are dropped.
data(Pinf)
Pinf
pmiss <- info_table(Pinf, type = "ploidy", plot = TRUE)

# No samples appear to be triploid across all loci. This will cause
# several haplotypes to have a lot of missing data.
p_haps <- make_haplotypes(Pinf)
p_haps
head(genind2df(p_haps), n = 20)
```

---

missingno *Treat missing data*

Description

missingno gives the user four options to deal with missing data: remove loci, remove samples, replace with zeroes, or replace with average allele counts.

Usage

```
missingno(pop, type = "loci", cutoff = 0.05, quiet = FALSE, freq = FALSE)
```

Arguments

pop	a genclone or genind object.
type	a character string: can be "ignore", "zero", "mean", "loci", or "geno" (see Details for definitions).
cutoff	numeric. A number from 0 to 1 indicating the allowable rate of missing data in either genotypes or loci. This will be ignored for type values of "mean" or "zero".
quiet	if TRUE, it will print to the screen the action performed.
freq	defaults to FALSE. This option is passed on to the tab function. If TRUE, the matrix in the genind object will be replaced by a numeric matrix (as opposed to integer). THIS IS NOT RECOMMENDED. USE THE FUNCTION tab instead.

Details

These methods provide a way to deal with systematic missing data and to give a wrapper for adegenet's tab function. ALL OF THESE ARE TO BE USED WITH CAUTION.

Using this function with polyploid data (where missing data is coded as "0") may give spurious results.

Treatment types:

- "ignore" - does not remove or replace missing data.
- "loci" - removes all loci containing missing data in the entire data set.
- "genotype" - removes any genotypes/isolates/individuals with missing data.
- "mean" - replaces all NA's with the mean of the alleles for the entire data set.
- "zero" or "0" - replaces all NA's with "0". Introduces more diversity.

Value

a genclone or genind object.

Note

"wild missingno appeared!"

Author(s)

Zhian N. Kamvar

See Also

tab, poppr, poppr.amova, nei.dist, aboot

Examples

```
data(nancycats)

nancy.locina <- missingno(nancycats, type = "loci")

## Found 617 missing values.
## 2 loci contained missing values greater than 5%.
## Removing 2 loci : fca8 fca45

nancy.genona <- missingno(nancycats, type = "geno")

## Found 617 missing values.
## 38 genotypes contained missing values greater than 5%.
## Removing 38 genotypes : N215 N216 N188 N189 N190 N191 N192 N302 N304 N310
## N195 N197 N198 N199 N200 N201 N206 N182 N184 N186 N298 N299 N300 N301 N303
## N282 N283 N288 N291 N292 N293 N294 N295 N296 N297 N281 N289 N290

# Replacing all NA with "0" (see tab in the adegenet package).
nancy.0 <- missingno(nancycats, type = "0")

## Replaced 617 missing values

# Replacing all NA with the mean of each column (see tab in the
# adegenet package).
nancy.mean <- missingno(nancycats, type = "mean")

## Replaced 617 missing values
```

---

mlg *Create counts, vectors, and matrices of multilocus genotypes.*

Description

Create counts, vectors, and matrices of multilocus genotypes.

Usage

```
mlg(gid, quiet = FALSE)

mlg.table(
  gid,
  strata = NULL,
  sublist = "ALL",
  exclude = NULL,
  blacklist = NULL,
  mlgsub = NULL,
  bar = TRUE,
  plot = TRUE,
  total = FALSE,
```

```
  color = FALSE,
  background = FALSE,
  quiet = FALSE
)

mlg.vector(gid, reset = FALSE)

mlg.crosspop(
  gid,
  strata = NULL,
  sublist = "ALL",
  exclude = NULL,
  blacklist = NULL,
  mlgsub = NULL,
  indexreturn = FALSE,
  df = FALSE,
  quiet = FALSE
)

mlg.id(gid)
```

Arguments

gid	a adegenet::genind, genclone, adegenet::genlight, or snpclone object.
quiet	Logical. If FALSE, progress of functions will be printed to the screen.
strata	a formula specifying the strata at which computation is to be performed.
sublist	a vector of population names or indices that the user wishes to keep. Default to "ALL".
exclude	a vector of population names or indexes that the user wishes to discard. Default to NULL.
blacklist	DEPRECATED, use exclude.
mlgsub	a vector of multilocus genotype indices with which to subset mlg.table and mlg.crosspop. NOTE: The resulting table from mlg.table will only contain countries with those MLGs
bar	deprecated. Same as plot. Retained for compatibility.
plot	logical If TRUE, a bar graph for each population will be displayed showing the relative abundance of each MLG within the population.
total	logical If TRUE, a row containing the sum of all represented MLGs is appended to the matrix produced by mlg.table.
color	an option to display a single barchart for mlg.table, colored by population (note, this becomes facetted if 'background = TRUE').
background	an option to display the the total number of MLGs across populations per facet in the background of the plot.
reset	logical. For genclone objects, the MLGs are defined by the input data, but they do not change if more or less information is added (i.e. loci are dropped). Setting

	'reset = TRUE' will recalculate MLGs. Default is 'FALSE', returning the MLGs defined in the @mlg slot.
`indexreturn`	`logical` If `TRUE`, a vector will be returned to index the columns of `mlg.table`.
`df`	`logical` If `TRUE`, return a data frame containing the counts of the MLGs and what countries they are in. Useful for making graphs with ggplot2::ggplot.

Details

Multilocus genotypes are the unique combination of alleles across all loci. For details of how these are calculated see `vignette("mlg", package = "poppr")`. In short, for genind and genclone objects, they are calculated by using a rank function on strings of alleles, which is sensitive to missing data. For genlight and snpclone objects, they are calculated with distance methods via bitwise.dist and mlg.filter, which means that these are insensitive to missing data. Three different types of MLGs can be defined in **poppr**:

- **original** the default definition of multilocus genotypes as detailed above
- **contracted** these are multilocus genotypes collapsed into multilocus lineages (mll) with genetic distance via mlg.filter
- **custom** user-defined multilocus genotypes. These are useful for information such as mycelial compatibility groups

All of the functions documented here will work on any of the MLG types defined in poppr

Value

mlg:

an integer describing the number of multilocus genotypes observed.

mlg.table:

a matrix with columns indicating unique multilocus genotypes and rows indicating populations. This table can be used with the funciton diversity_stats to calculate the Shannon-Weaver index (H), Stoddart and Taylor's index (aka inverse Simpson's index; G), Simpson's index (lambda), and evenness (E5).

mlg.vector:

a numeric vector naming the multilocus genotype of each individual in the dataset.

mlg.crosspop:

- **default** a `list` where each element contains a named integer vector representing the number of individuals represented from each population in that MLG
- `indexreturn = TRUE` a `vector` of integers defining the multilocus genotypes that have individuals crossing populations
- `df = TRUE` A long form data frame with the columns: MLG, Population, Count. Useful for graphing with ggplot2

mlg.id:

a list of multilocus genotypes with the associated individual names per MLG.

Note

The resulting matrix of ‘mlg.table‘ can be used for analysis with the **vegan** package.

mlg.vector will recalculate the mlg vector for [adegenet::genind] objects and will return the contents of the mlg slot in [genclone][genclone-class] objects. This means that MLGs will be different for subsetted [adegenet::genind] objects.

Author(s)

Zhian N. Kamvar

See Also

vegan::diversity() diversity_stats popsub mll mlg.filter mll.custom

Examples

```
# Load the data set
data(Aeut)

# Investigate the number of multilocus genotypes.
amlg <- mlg(Aeut)
amlg # 119

# show the multilocus genotype vector
avec <- mlg.vector(Aeut)
avec

# Get a table
atab <- mlg.table(Aeut, color = TRUE)
atab

# See where multilocus genotypes cross populations
acrs <- mlg.crosspop(Aeut) # MLG.59: (2 inds) Athena Mt. Vernon

# See which individuals belong to each MLG
aid <- mlg.id(Aeut)
aid["59"] # individuals 159 and 57

## Not run:

# For the mlg.table, you can also choose to display the number of MLGs across
# populations in the background

mlg.table(Aeut, background = TRUE)
mlg.table(Aeut, background = TRUE, color = TRUE)

# A simple example. 10 individuals, 5 genotypes.
mat1 <- matrix(ncol=5, 25:1)
mat1 <- rbind(mat1, mat1)
mat <- matrix(nrow=10, ncol=5, paste(mat1,mat1,sep="/"))
mat.gid <- df2genind(mat, sep="/")
```

```
mlg(mat.gid)
mlg.vector(mat.gid)
mlg.table(mat.gid)

# Now for a more complicated example.
# Data set of 1903 samples of the H3N2 flu virus genotyped at 125 SNP loci.
data(H3N2)
mlg(H3N2, quiet = FALSE)

H.vec <- mlg.vector(H3N2)

# Changing the population vector to indicate the years of each epidemic.
pop(H3N2) <- other(H3N2)$x$country
H.tab <- mlg.table(H3N2, plot = FALSE, total = TRUE)

# Show which genotypes exist accross populations in the entire dataset.
res <- mlg.crosspop(H3N2, quiet = FALSE)

# Let's say we want to visualize the multilocus genotype distribution for the
# USA and Russia
mlg.table(H3N2, sublist = c("USA", "Russia"), bar=TRUE)

# An exercise in subsetting the output of mlg.table and mlg.vector.
# First, get the indices of each MLG duplicated across populations.
inds <- mlg.crosspop(H3N2, quiet = FALSE, indexreturn = TRUE)

# Since the columns of the table from mlg.table are equal to the number of
# MLGs, we can subset with just the columns.
H.sub <- H.tab[, inds]

# We can also do the same by using the mlgsub flag.
H.sub <- mlg.table(H3N2, mlgsub = inds)

# We can subset the original data set using the output of mlg.vector to
# analyze only the MLGs that are duplicated across populations.
new.H <- H3N2[H.vec %in% inds, ]


## End(Not run)
```

`mlg.filter` *MLG definitions based on genetic distance*

Description

Multilocus genotypes are initially defined by naive string matching, but this definition does not take into account missing data or genotyping error, casting these as unique genotypes. Defining multilocus genotypes by genetic distance allows you to incorporate genotypes that have missing data o genotyping error into their parent clusters.

Usage

```
mlg.filter(
  pop,
  threshold = 0,
  missing = "asis",
  memory = FALSE,
  algorithm = "farthest_neighbor",
  distance = "diss.dist",
  threads = 1L,
  stats = "MLGs",
  ...
)

mlg.filter(
  pop,
  missing = "asis",
  memory = FALSE,
  algorithm = "farthest_neighbor",
  distance = "diss.dist",
  threads = 1L,
  ...
) <- value
```

Arguments

pop	a genclone, snpclone, or genind object.
threshold	a number indicating the minimum distance two MLGs must be separated by to be considered different. Defaults to 0, which will reflect the original (naive) MLG definition.
missing	any method to be used by missingno: "mean", "zero", "loci", "genotype", or "asis" (default).
memory	whether this function should remember the last distance matrix it generated. TRUE will attempt to reuse the last distance matrix if the other parameters are the same. (default) FALSE will ignore any stored matrices and not store any it generates.
algorithm	determines the type of clustering to be done. **"farthest_neighbor"** *(default)* merges clusters based on the maximum distance between points in either cluster. This is the strictest of the three. **"nearest_neighbor"** merges clusters based on the minimum distance between points in either cluster. This is the loosest of the three. **"average_neighbor"** merges clusters based on the average distance between every pair of points between clusters.
distance	a character or function defining the distance to be applied to pop. Defaults to diss.dist for genclone objects and bitwise.dist for snpclone objects. A matrix or table containing distances between individuals (such as the output of rogers.dist) is also accepted for this parameter.

threads	(unused) Previously, this was the maximum number of parallel threads to be used within this function. Default is 1 indicating that this function will run serially. Any other number will result in a warning.
stats	a character vector specifying which statistics should be returned (details below). Choices are "MLG", "THRESHOLDS", "DISTANCES", "SIZES", or "ALL". If choosing "ALL" or more than one, a named list will be returned.
...	any parameters to be passed off to the distance method.
value	the threshold at which genotypes should be collapsed.

Details

This function will take in any distance matrix or function and collapse multilocus genotypes below a given threshold. If you use this function as the assignment method (mlg.filter(myData, distance = myDist) <- 0.5), the distance function or matrix will be remembered by the object. This means that if you define your own distance matrix or function, you must keep it in memory to further utilize mlg.filter.

Value

Default, a vector of collapsed multilocus genotypes. Otherwise, any combination of the following:

MLGs: a numeric vector defining the multilocus genotype cluster of each individual in the dataset. Each genotype cluster is separated from every other genotype cluster by at least the defined threshold value, as calculated by the selected algorithm.

THRESHOLDS: A numeric vector representing the thresholds **beyond** which clusters of multilocus genotypes were collapsed.

DISTANCES: A square matrix representing the distances between each cluster.

SIZES: The sizes of the multilocus genotype clusters in order.

Note

`mlg.vector` makes use of `mlg.vector` grouping prior to applying the given threshold. Genotype numbers returned by `mlg.vector` represent the lowest numbered genotype (as returned by `mlg.vector`) in in each new multilocus genotype. Therefore `mlg.filter` **and** `mlg.vector` **return the same vector when threshold is set to 0 or less**.

See Also

filter_stats, cutoff_predictor, mll, genclone, snpclone, diss.dist, bruvo.dist

Examples

```
data(partial_clone)
pc <- as.genclone(partial_clone, threads = 1L) # convert to genclone object

# Basic Use ------------------------------------------------------------
```

```
# Show MLGs at threshold 0.05
mlg.filter(pc, threshold = 0.05, distance = "nei.dist", threads = 1L)
pc # 26 mlgs

# Set MLGs at threshold 0.05
mlg.filter(pc, distance = "nei.dist", threads = 1L) <- 0.05
pc # 25 mlgs

## Not run:

# The distance definition is persistant
mlg.filter(pc) <- 0.1
pc # 24 mlgs

# But you can still change the definition
mlg.filter(pc, distance = "diss.dist", percent = TRUE) <- 0.1
pc

# Choosing a threshold ----------------------------------------------------


# Thresholds for collapsing multilocus genotypes should not be arbitrary. It
# is important to consider what threshold is suitable. One method of choosing
# a threshold is to find a gap in the distance distribution that represents
# clonal groups. You can look at this by analyzing the distribution of all
# possible thresholds with the function "cutoff_predictor".

# For this example, we'll use Bruvo's distance to predict the cutoff for
# P. infestans.

data(Pinf)
Pinf
# Repeat lengths are necessary for Bruvo's distance
(pinfreps <- fix_replen(Pinf, c(2, 2, 6, 2, 2, 2, 2, 2, 3, 3, 2)))

# Now we can collect information of the thresholds. We can set threshold = 1
# because we know that this will capture the maximum possible distance:
(thresholds <- mlg.filter(Pinf, distance = bruvo.dist, stats = "THRESHOLDS",
                          replen = pinfreps, threshold = 1))
# We can use these thresholds to find an appropriate cutoff
(pcut <- cutoff_predictor(thresholds))
mlg.filter(Pinf, distance = bruvo.dist, replen = pinfreps) <- pcut
Pinf

# This can also be visualized with the "filter_stats" function.

# Special case: threshold = 0 ---------------------------------------------


# It's important to remember that a threshold of 0 is equal to the original
# MLG definition. This example will show a data set that contains genotypes
# with missing data that share all alleles with other genotypes except for
```

```
# the missing one.

data(monpop)
monpop # 264 mlg
mlg.filter(monpop) <- 0
nmll(monpop) # 264 mlg

# In order to merge these genotypes with missing data, we should set the
# threshold to be slightly higher than 0. We will use the smallest fraction
# the computer can store.

mlg.filter(monpop) <- .Machine$double.eps ^ 0.5
nmll(monpop) # 236 mlg

# Custom distance ----------------------------------------------------------

# Custom genetic distances can be used either in functions from other
# packages or user-defined functions

data(Pinf)
Pinf
mlg.filter(Pinf, distance = function(x) dist(tab(x))) <- 3
Pinf
mlg.filter(Pinf) <- 4
Pinf

# genlight / snpclone objects ----------------------------------------------


set.seed(999)
gc <- as.snpclone(glSim(100, 0, n.snp.struc = 1e3, ploidy = 2))
gc # 100 mlgs
mlg.filter(gc) <- 0.25
gc # 82 mlgs


## End(Not run)
```

`mll` *Access and manipulate multilocus lineages.*

Description

The following methods allow the user to access and manipulate multilocus lineages in genclone or snpclone objects.

Usage

```
mll(x, type = NULL)
```

```
nmll(x, type = NULL)

mll(x) <- value
```

Arguments

x	a genclone or snpclone object.
type	a character specifying "original", "contracted", or "custom" defining they type of mlgs to return. Defaults to what is set in the object.
value	a character specifying which mlg type is visible in the object. See details.

Details

genclone and snpclone objects have a slot for an internal class of object called MLG. This class allows the storage of flexible mll definitions:

- "original" - naive mlgs defined by string comparison. This is default.
- "contracted" - mlgs defined by a genetic distance threshold.
- "custom" - user-defined MLGs

Value

an object of the same type as x.

Author(s)

Zhian N. Kamvar

See Also

mll.custom mlg.table

Examples

```
data(partial_clone)
pc <- as.genclone(partial_clone)
mll(pc)
mll(pc) <- "custom"
mll(pc)
mll.levels(pc) <- LETTERS
mll(pc)
```

`mll.custom` *Define custom multilocus lineages*

Description

This function will allow you to define custom multilocus lineages for your data set.

Usage

```
mll.custom(x, set = TRUE, value)

mll.custom(x, set = TRUE) <- value

mll.levels(x, set = TRUE, value)

mll.levels(x, set = TRUE) <- value
```

Arguments

`x`	a genclone or snpclone object.
`set`	logical. If `TRUE` (default), the visible mlls will be set to 'custom'.
`value`	a vector that defines the multilocus lineages for your data. This can be a vector of ANYTHING that can be turned into a factor.

Value

an object of the same type as x

Author(s)

Zhian N. Kamvar

See Also

`mll` `mlg.table`

Examples

```
data(partial_clone)
pc <- as.genclone(partial_clone)
mll.custom(pc) <- LETTERS[mll(pc)]
mll(pc)

# Let's say we had a mistake and the A mlg was actually B.
mll.levels(pc)[mll.levels(pc) == "A"] <- "B"
mll(pc)

# Set the MLL back to the original definition.
```

```
mll(pc) <- "original"
mll(pc)
```

`mll.reset` *Reset multilocus lineages*

Description

This function will allow you to reset multilocus lineages for your data set.

Usage

```
mll.reset(x, value)
```

Arguments

`x`	a genclone or snpclone object.
`value`	a character vector that specifies which levels you wish to be reset.

Value

an object of the same type as x

Note

This method has no assignment method. If "original" is not contained in "value", it is assumed that the "original" definition will be used to reset the MLGs.

Author(s)

Zhian N. Kamvar

See Also

`mll` `mlg.table` `mll.custom`

Examples

```
# This data set was a subset of a larger data set, so the multilocus
# genotypes are not all sequential
data(Pinf)
mll(Pinf) <- "original"
mll(Pinf)

# If we use mll.reset, then it will become sequential
Pinf.new <- mll.reset(Pinf, TRUE) # reset all
mll(Pinf.new)

## Not run:
```

```
# It is possible to reset only specific mll definitions. For example, let's
# say that we wanted to filter our multilocus genotypes by nei's distance
mlg.filter(Pinf, dist = nei.dist, missing = "mean") <- 0.02

# And we wanted to set those as custom genotypes,
mll.custom(Pinf) <- mll(Pinf, "contracted")
mll.levels(Pinf) <- .genlab("MLG", nmll(Pinf, "custom"))

# We could reset just the original and the filtered if we wanted to and keep
# the custom as it were.

Pinf.new <- mll.reset(Pinf, c("original", "contracted"))

mll(Pinf.new, "original")
mll(Pinf.new, "contracted")
mll(Pinf.new, "custom")

# If "original" is not one of the values, then that is used as a baseline.
Pinf.orig <- mll.reset(Pinf, "contracted")
mll(Pinf.orig, "contracted")
mll(Pinf.new, "contracted")

## End(Not run)
```

---

monpop *Peach brown rot pathogen *Monilinia fructicola**

Description

This is microsatellite data for a population of the haploid plant pathogen *Monilinia fructicola* that causes disease within peach tree canopies (Everhart & Scherm, 2014). Entire populations within trees were sampled across 3 years (2009, 2010, and 2011) in a total of four trees, where one tree was sampled in all three years, for a total of 6 within-tree populations. Within each year, samples in the spring were taken from affected blossoms (termed "BB" for blossom blight) and in late summer from affected fruits (termed "FR" for fruit rot). There are a total of 694 isolates with 65 to 173 isolates within each canopy population that were characterized using a set of 13 microsatellite markers.

Usage

```
data(monpop)
```

Format

a [genclone-class] object with 3 hierarchical levels coded into one population factor. These are named "Tree", "Year", and "Symptom"

References

SE Everhart, H Scherm, (2015) Fine-scale genetic structure of *Monilinia fructicola* during brown rot epidemics within individual peach tree canopies. Phytopathology 105:542-549 doi: doi:10.1094/PHYTO03140088R

Examples

```
data(monpop)
splitStrata(monpop) <- ~Tree/Year/Symptom
setPop(monpop) <- ~Symptom/Year
monpop
```

`nei.dist` *Calculate Genetic Distance for a genind or genclone object.*

Description

These functions are modified from the function dist.genpop to be applicable for distances between individuals.

Usage

```
nei.dist(x, warning = TRUE)

edwards.dist(x)

rogers.dist(x)

reynolds.dist(x)

provesti.dist(x)

prevosti.dist
```

Arguments

`x`	a genind, genclone, or matrix object.
`warning`	If `TRUE`, a warning will be printed if any infinite values are detected and replaced. If `FALSE`, these values will be replaced without warning. See Details below.

Format

An object of class `function` of length 1.

Details

It is important to be careful with the interpretation of these distances as they were originally intended for calculation of between-population distance. As Nei's distance is the negative log of 0:1, this means that it is very possible to obtain distances of infinity. When this happens, infinite values are corrected to be 10 * max(D) where D is the distance matrix without infinite values.

Value

an object of class dist with the same number of observations as the number of individuals in your data.

Note

Prevosti's distance is identical to `diss.dist`, except that `diss.dist` is optimized for a larger number of individuals (n > 125) at the cost of required memory. Both `prevosti.dist` and `provesti.dist` are the same function, `provesti.dist` is a spelling error and exists for backwards compatibility.

These distances were adapted from the **adegenet** function `dist.genpop` to work with `genind` objects.

Author(s)

Zhian N. Kamvar (poppr adaptation) Thibaut Jombart (adegenet adaptation) Daniel Chessel (ade4)

References

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.

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).

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.

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.

Prevosti A. (1974) La distancia genetica entre poblaciones. Miscellanea Alcobe, 68, 109-118.

Prevosti A., Ocana 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 Euclidean 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

aboot diss.dist poppr.amova

Examples

```
data(nancycats)
(nan9   <- popsub(nancycats, 9))
(neinan <- nei.dist(nan9))
(ednan  <- edwards.dist(nan9))
(rodnan <- rogers.dist(nan9))
(reynan <- reynolds.dist(nan9))
(pronan <- prevosti.dist(nan9))
```

---

old2new_genclone *Convert an old genclone object to a new genclone object*

Description

Convert an old genclone object to a new genclone object

Usage

```
old2new_genclone(object, donor = new(class(object)))
```

Arguments

object	a genclone object from poppr v. 1.1
donor	a new genclone object from poppr v. 2.0

Author(s)

Zhian N. Kamvar

partial_clone — *Simulated data illustrating a Minimum Spanning Network based on Bruvo's Distance*

Description

These data were simulated using SimuPOP version 1.0.8 with 99.9% clonal reproduction over 10,000 generations. Populations were assigned post-hoc and are simply present for the purposes of demonstrating a minimum spanning network with Bruvo's distance.

Usage

```
data(partial_clone)
```

Format

a [genind()] object with 50 individuals, 10 loci, and four populations.

References

Bo Peng and Christopher Amos (2008) Forward-time simulations of nonrandom mating populations using simuPOP. *bioinformatics*, 24 (11): 1408-1409.

pgen — *Genotype Probability*

Description

Calculate the probability of genotypes based on the product of allele frequencies over all loci.

Usage

```
pgen(gid, pop = NULL, by_pop = TRUE, log = TRUE, freq = NULL, ...)
```

Arguments

gid	a genind or genclone object.
pop	either a formula to set the population factor from the strata slot or a vector specifying the population factor for each sample. Defaults to NULL.
by_pop	When this is TRUE (default), the calculation will be done by population.
log	a logical if log =TRUE (default), the values returned will be log(Pgen). If log = FALSE, the values returned will be Pgen.
freq	a vector or matrix of allele frequencies. This defaults to NULL, indicating that the frequencies will be determined via round-robin approach in rraf. **If this matrix or vector is not provided, zero-value allele frequencies will automatically be corrected.** For details, please see the documentation on correcting rare alleles.

... options from correcting rare alleles. The default is to correct allele frequencies to 1/n

Details

Pgen is the probability of a given genotype occuring in a population assuming HWE. Thus, the value for diploids is

$$P_{gen} = \left(\prod_{i=1}^{m} p_i\right) 2^h$$

where p_i are the allele frequencies and h is the count of the number of heterozygous sites in the sample (Arnaud-Haond et al. 2007; Parks and Werth, 1993). The allele frequencies, by default, are calculated using a round-robin approach where allele frequencies at a particular locus are calculated on the clone-censored genotypes without that locus.

To avoid issues with numerical precision of small numbers, this function calculates pgen per locus by adding up log-transformed values of allele frequencies. These can easily be transformed to return the true value (see examples).

Value

A vector containing Pgen values per locus for each genotype in the object.

Note

For haploids, Pgen at a particular locus is the allele frequency. This function cannot handle polyploids. Additionally, when the argument `pop` is not `NULL`, `by_pop` is automatically `TRUE`.

Author(s)

Zhian N. Kamvar, Jonah Brooks, Stacy A. Krueger-Hadfield, Erik Sotka

References

Arnaud-Haond, S., Duarte, C. M., Alberto, F., & Serrão, E. A. 2007. Standardizing methods to address clonality in population studies. *Molecular Ecology*, 16(24), 5115-5139.

Parks, J. C., & Werth, C. R. 1993. A study of spatial features of clones in a population of bracken fern, *Pteridium aquilinum* (Dennstaedtiaceae). *American Journal of Botany*, 537-544.

See Also

`psex`, `rraf`, `rrmlg`, `rare_allele_correction`

Examples

```
data(Pram)
head(pgen(Pram, log = FALSE))

## Not run:
```

```
# You can also supply the observed allele frequencies
pramfreq <- Pram %>% genind2genpop() %>% tab(freq = TRUE)
head(pgen(Pram, log = FALSE, freq = pramfreq))

# You can get the Pgen values over all loci by summing over the logged results:
pgen(Pram, log = TRUE) %>%  # calculate pgen matrix
  rowSums(na.rm = TRUE) %>% # take the sum of each row
  exp()                     # take the exponent of the results

# You can also take the product of the non-logged results:
apply(pgen(Pram, log = FALSE), 1, prod, na.rm = TRUE)

## Rare Allele Correction ------------------------------------------------
##
# If you don't supply a table of frequencies, they are calculated with rraf
# with correction = TRUE. This is normally benign when analyzing large
# populations, but it can have a great effect on small populations. To help
# control this, you can supply arguments described in
# help("rare_allele_correction").


# Default is to correct by 1/n per population. Since the calculation is
# performed on a smaller sample size due to round robin clone correction, it
# would be more appropriate to correct by 1/rrmlg at each locus. This is
# acheived by setting d = "rrmlg". Since this is a diploid, we would want to
# account for the number of chromosomes, and so we set mul = 1/2
head(pgen(Pram, log = FALSE, d = "rrmlg", mul = 1/2)) # compare with the output above

# If you wanted to treat all alleles as equally rare, then you would set a
# specific value (let's say the rare alleles are 1/100):
head(pgen(Pram, log = FALSE, e = 1/100))

## End(Not run)
```

`Pinf` *Phytophthora infestans data from Mexico and South America.*

Description

The Pinf data set contains 86 isolates genotyped over 11 microsatellite loci collected from Mexico, Peru, Columbia, and Ecuador. This is a subset of the data used for the reference below.

Usage

```
data(Pinf)
```

Format

a [genclone-class] object with 2 hierarchical levels called "Continent" and "Country" that contain 2 and 4 populations, respectively.

References

Goss, Erica M., Javier F. Tabima, David EL Cooke, Silvia Restrepo, William E. Fry, Gregory A. Forbes, Valerie J. Fieland, Martha Cardenas, and Niklaus J. Grünwald. "The Irish potato famine pathogen *Phytophthora infestans* originated in central Mexico rather than the Andes." Proceedings of the National Academy of Sciences 111:8791-8796. doi: doi:10.1073/pnas.1401884111

`plot_poppr_msn` *Plot minimum spanning networks produced in poppr.*

Description

This function allows you to take the output of poppr.msn and bruvo.msn and customize the plot by labeling groups of individuals, size of nodes, and adjusting the palette and scale bar.

Usage

```
plot_poppr_msn(
  x,
  poppr_msn,
  gscale = TRUE,
  gadj = 3,
  mlg.compute = "original",
  glim = c(0, 0.8),
  gweight = 1,
  wscale = TRUE,
  nodescale = 10,
  nodebase = NULL,
  nodelab = 2,
  inds = "ALL",
  mlg = FALSE,
  quantiles = TRUE,
  cutoff = NULL,
  palette = NULL,
  layfun = layout.auto,
  beforecut = FALSE,
  pop.leg = TRUE,
  size.leg = TRUE,
  scale.leg = TRUE,
  ...
)
```

Arguments

x	a genind, genclone, genlight, or snpclone object from which poppr_msn was derived.

poppr_msn — a list produced from either poppr.msn or bruvo.msn. This list should contain a graph, a vector of population names and a vector of hexadecimal color definitions for each population.

gscale — "grey scale". If this is TRUE, this will scale the color of the edges proportional to the observed distance, with the lines becoming darker for more related nodes. See greycurve for details.

gadj — "grey adjust". a positive integer greater than zero that will serve as the exponent to the edge weight to scale the grey value to represent that weight.

mlg.compute — if the multilocus genotypes are set to "custom" (see mll.custom for details) in your genclone object, this will specify which mlg level to calculate the nodes from. See details.

glim — "grey limit". Two numbers between zero and one. They determine the upper and lower limits for the gray function. Default is 0 (black) and 0.8 (20% black).

gweight — "grey weight". an integer. If it's 1, the grey scale will be weighted to emphasize the differences between closely related nodes. If it is 2, the grey scale will be weighted to emphasize the differences between more distantly related nodes.

wscale — "width scale". If this is TRUE, the edge widths will be scaled proportional to the inverse of the observed distance , with the lines becoming thicker for more related nodes.

nodescale — a numeric indicating how to scale the node sizes (scales by area).

nodebase — **deprecated** a numeric indicating what base logarithm should be used to scale the node sizes. Defaults to 1.15. See details.

nodelab — an integer specifying the smallest size of node to label. See details.

inds — a character or numeric vector indicating which samples or multilocus genotypes to label on the graph. See details.

mlg — logical When TRUE, the nodes will be labeled by multilocus genotype. When FALSE (default), nodes will be labeled by sample names.

quantiles — logical. When set to TRUE (default), the scale bar will be composed of the quantiles from the observed edge weights. When set to FALSE, the scale bar will be composed of a smooth gradient from the minimum edge weight to the maximum edge weight.

cutoff — a number indicating the longest distance to display in your graph. This is performed by removing edges with weights greater than this number.

palette — a function or character corresponding to a specific palette you want to use to delimit your populations. The default is whatever palette was used to produce the original graph.

layfun — a function specifying the layout of nodes in your graph. It defaults to layout.auto.

beforecut — if TRUE, the layout of the graph will be computed before any edges are removed with cutoff. If FALSE (Default), the layout will be computed after any edges are removed.

pop.leg — if TRUE, a legend indicating the populations will appear in the top right corner of the graph, but will not overlap. Setting pop.leg = FALSE disables this legend. See details.

size.leg	if TRUE, a legend displyaing the number of samples per node will appear either below the population legend or in the top right corner of the graph. Setting size.leg = FALSE disables this legend.
scale.leg	if TRUE, a scale bar indicating the distance will appear under the graph. Setting scale.leg = FALSE suppresses this bar. See details.
...	any other parameters to be passed on to plot.igraph.

Details

The previous incarnation of msn plotting in poppr simply plotted the minimum spanning network with the legend of populations, but did not provide a scale bar and it did not provide the user a simple way of manipulating the layout or labels. This function allows the user to manipulate many facets of graph creation, making the creation of minimum spanning networks ever so slightly more user friendly.

This function must have both the source data and the output msn to work. The source data must contain the same population structure as the graph. Every other parameter has a default setting.

Parameter details:

- inds By default, the graph will label each node (circle) with all of the samples (individuals) that are contained within that node. As each node represents a single multilocus genotype (MLG) or individuals (n >= 1), this argument is designed to allow you to selectively label the nodes based on query of sample name or MLG number. If the option mlg = TRUE, the multilocus genotype assignment will be used to label the node. If you do not want to label the nodes by individual or multilocus genotype, simply set this to a name that doesn't exist in your data.
- nodescale The nodes (circles) on the graph represent different multilocus genotypes. The area of the nodes represent the number of individuals. Setting nodescale will scale the area of the nodes.
- nodelab If a node is not labeled by individual, this will label the size of the nodes greater than or equal to this value. If you don't want to label the size of the nodes, simply set this to a very high number.
- cutoff This is useful for when you want to investigate groups of multilocus genotypes separated by a specific distance or if you have two distinct populations and you want to physically separate them in your network.
- beforecut This is an indicator useful if you want to maintain the same position of the nodes before and after removing edges with the cutoff argument. This works best if you set a seed before you run the function.

mlg.compute: Each node on the graph represents a different multilocus genotype. The edges on the graph represent genetic distances that connect the multilocus genotypes. In genclone objects, it is possible to set the multilocus genotypes to a custom definition. This creates a problem for clone correction, however, as it is very possible to define custom lineages that are not monophyletic. When clone correction is performed on these definitions, information is lost from the graph. To circumvent this, The clone correction will be done via the computed multilocus genotypes, either "original" or "contracted". This is specified in the mlg.compute argument, above.

legends: To avoid drawing the legend over the graph, legends are separated by different plotting areas. This means that if legends are included, you cannot plot multiple MSNs in a single plot.

The scale bar (to be added in manually) can be obtained from greycurve and the legend can be plotted with legend.

Value

the modified msn list, invisibly.

Author(s)

Zhian N. Kamvar

See Also

layout.auto plot.igraph poppr.msn bruvo.msn greycurve delete_edges palette

Examples

```
# Using a data set of the Aphanomyces eutieches root rot pathogen.
data(Aeut)
adist <- diss.dist(Aeut, percent = TRUE)
amsn <- poppr.msn(Aeut, adist, showplot = FALSE)

# Default
library("igraph") # To get all the layouts.
set.seed(500)
plot_poppr_msn(Aeut, amsn, gadj = 15)

## Not run:

# Different layouts (from igraph) can be used by supplying the function name.
set.seed(500)
plot_poppr_msn(Aeut, amsn, gadj = 15, layfun = layout_with_kk)

# Removing link between populations (cutoff = 0.2) and labelling no individuals
set.seed(500)
plot_poppr_msn(Aeut, amsn, inds = "none", gadj = 15, beforecut = TRUE, cutoff = 0.2)

# Labelling individual #57 because it is an MLG that crosses populations
# Showing clusters of MLGS with at most 5% variation
# Notice that the Mt. Vernon population appears to be more clonal
set.seed(50)
plot_poppr_msn(Aeut, amsn, gadj = 15, cutoff = 0.05, inds = "057")


data(partial_clone)
pcmsn <- bruvo.msn(partial_clone, replen = rep(1, 10))

# You can plot using a color palette or a vector of named colors
# Here's a way to define the colors beforehand
pc_colors <- nPop(partial_clone) %>%
  RColorBrewer::brewer.pal("Set2") %>%
  setNames(popNames(partial_clone))
```

```
pc_colors

# Labelling the samples contained in multilocus genotype 9
set.seed(999)
plot_poppr_msn(partial_clone, pcmsn, palette = pc_colors, inds = 9)

# Doing the same thing, but using one of the sample names as input.
set.seed(999)
plot_poppr_msn(partial_clone, pcmsn, palette = pc_colors, inds = "sim 20")

# Note that this is case sensitive. Nothing is labeled.
set.seed(999)
plot_poppr_msn(partial_clone, pcmsn, palette = pc_colors, inds = "Sim 20")

# Something pretty
data(microbov)
mdist <- diss.dist(microbov, percent = TRUE)
micmsn <- poppr.msn(microbov, mdist, showplot = FALSE)

plot_poppr_msn(microbov, micmsn, palette = "terrain.colors", inds = "n",
  quantiles = FALSE)
plot_poppr_msn(microbov, micmsn, palette = "terrain.colors", inds = "n",
  cutoff = 0.3, quantiles = FALSE)

### Utilizing vectors for palettes

data(Pram)
Pram_sub <- popsub(Pram, exclude = c("Nursery_CA", "Nursery_OR"))

# Creating the network for the forest
min_span_net_sub <- bruvo.msn(Pram_sub, replen = other(Pram)$REPLEN,
                              add = TRUE, loss = TRUE, showplot = FALSE,
                              include.ties = TRUE)

# Creating the network with nurseries
min_span_net     <- bruvo.msn(Pram, replen = other(Pram)$REPLEN,
                              add = TRUE, loss = TRUE, showplot = FALSE,
                              include.ties = TRUE)

# Only forest genotypes
set.seed(70)
plot_poppr_msn(Pram,
               min_span_net_sub,
               inds = "ALL",
               mlg = TRUE,
               gadj = 9,
               nodescale = 5,
               palette = other(Pram)$comparePal,
               cutoff = NULL,
               quantiles = FALSE,
               beforecut = TRUE)
```

```
# With Nurseries
set.seed(70)
plot_poppr_msn(Pram,
               min_span_net,
               inds = "ALL",
               mlg = TRUE,
               gadj = 9,
               nodescale = 5,
               palette = other(Pram)$comparePal,
               cutoff = NULL,
               quantiles = FALSE,
               beforecut = TRUE)

## End(Not run)
```

---

poppr *Produce a basic summary table for population genetic analyses.*

Description

For the **poppr** package description, please see `package?poppr`

This function allows the user to quickly view indices of heterozygosity, evenness, and linkage to aid in the decision of a path to further analyze a specified dataset. It natively takes adegenet::genind and genclone objects, but can convert any raw data formats that adegenet can take (fstat, structure, genetix, and genpop) as well as genalex files exported into a csv format (see `read.genalex()` for details).

Usage

```
poppr(
  dat,
  total = TRUE,
  sublist = "ALL",
  exclude = NULL,
  blacklist = NULL,
  sample = 0,
  method = 1,
  missing = "ignore",
  cutoff = 0.05,
  quiet = FALSE,
  clonecorrect = FALSE,
  strata = 1,
  keep = 1,
  plot = TRUE,
  hist = TRUE,
  index = "rbarD",
  minsamp = 10,
  legend = FALSE,
```

```
  ...
)
```

Arguments

dat	a adegenet::genind object OR a genclone object OR any fstat, structure, genetix, genpop, or genalex formatted file.
total	When TRUE (default), indices will be calculated for the pooled populations.
sublist	a list of character strings or integers to indicate specific population names (accessed via adegenet::popNames()). Defaults to "ALL".
exclude	a vector of population names or indexes that the user wishes to discard. Default to NULL.
blacklist	DEPRECATED, use exclude.
sample	an integer indicating the number of permutations desired to obtain p-values. Sampling will shuffle genotypes at each locus to simulate a panmictic population using the observed genotypes. Calculating the p-value includes the observed statistics, so set your sample number to one off for a round p-value (eg. sample = 999 will give you p = 0.001 and sample = 1000 will give you p = 0.000999001).
method	an integer from 1 to 4 indicating the method of sampling desired. see shufflepop() for details.
missing	how should missing data be treated? "zero" and "mean" will set the missing values to those documented in adegenet::tab(). "loci" and "geno" will remove any loci or genotypes with missing data, respectively (see missingno() for more information.
cutoff	numeric a number from 0 to 1 indicating the percent missing data allowed for analysis. This is to be used in conjunction with the flag missing (see missingno() for details)
quiet	FALSE (default) will display a progress bar for each population analyzed.
clonecorrect	default FALSE. must be used with the strata parameter, or the user will potentially get undesired results. see clonecorrect() for details.
strata	a formula indicating the hierarchical levels to be used. The hierarchies should be present in the strata slot. See adegenet::strata() for details.
keep	an integer. This indicates which strata you wish to keep after clone correcting your data sets. To combine strata, just set keep from 1 to the number of straifications set in strata. see clonecorrect() for details.
plot	logical if TRUE (default) and sampling > 0, a histogram will be produced for each population.
hist	logical Deprecated. Use plot.
index	character Either "Ia" or "rbarD". If hist = TRUE, this will determine the index used for the visualization.
minsamp	an integer indicating the minimum number of individuals to resample for rarefaction analysis. See vegan::rarefy() for details.
legend	logical. When this is set to TRUE, a legend describing the resulting table columns will be printed. Defaults to FALSE
...	arguments to be passed on to diversity_stats()

Details

This table is intended to be a first look into the dynamics of mutlilocus genotype diversity. Many of the statistics (except for the the index of association) are simply based on counts of multilocus genotypes and do not take into account the actual allelic states. **Descriptions of the statistics can be found in the Algorithms and Equations vignette**: `vignette("algo", package = "poppr")`.

sampling:

The sampling procedure is explicitly for testing the index of association. None of the other diversity statistics (H, G, lambda, E.5) are tested with this sampling due to the differing data types. To obtain confidence intervals for these statistics, please see `diversity_ci()`.

rarefaction:

Rarefaction analysis is performed on the number of multilocus genotypes because it is relatively easy to estimate (Grünwald et al., 2003). To obtain rarefied estimates of diversity, it is possible to use `diversity_ci()` with the argument `rarefy = TRUE`

graphic:

This function outputs a **ggplot2** graphic of histograms. These can be manipulated to be visualized in another manner by retrieving the plot with the last plot command from **ggplot2**. A useful manipulation would be to arrange the graphs into a single column so that the values of the statistic line up: `p <- last_plot(); p + facet_wrap(~population, ncol = 1, scales = "free_y")` The name for the groupings is "population" and the name for the x axis is "value".

Value

A data frame with populations in rows and the following columns:

- **Pop**: A vector indicating the population factor
- **N**: An integer vector indicating the number of individuals/isolates in the specified population.
- **MLG**: An integer vector indicating the number of multilocus genotypes found in the specified population, (see: `mlg()`)
- **eMLG**: The expected number of MLG at the lowest common sample size (set by the parameter `minsamp`).
- **SE**: The standard error for the rarefaction analysis
- **H**: Shannon-Weiner Diversity index
- **G**: Stoddard and Taylor's Index
- **lambda**: Simpson's index
- **E.5**: Evenness
- **Hexp**: Nei's gene diversity (expected heterozygosity)
- **Ia**: A numeric vector giving the value of the Index of Association for each population factor, (see `ia()`).
- **p.Ia**: A numeric vector indicating the p-value for Ia from the number of reshufflings indicated in `sample`. Lowest value is 1/n where n is the number of observed values.
- **rbarD**: A numeric vector giving the value of the Standardized Index of Association for each population factor, (see `ia()`).

- **p.rD**: A numeric vector indicating the p-value for rbarD from the number of reshuffles indicated in `sample`. Lowest value is 1/n where n is the number of observed values.
- **File**: A vector indicating the name of the original data file.

Note

The calculation of `Hexp` has changed from **poppr** 1.x. It was previously calculated based on the diversity of multilocus genotypes, resulting in a value of 1 for sexual populations. This was obviously not Nei's 1978 expected heterozygosity. We have thus changed the statistic to be the true value of Hexp by calculating $(\frac{n}{n-1})1 - \sum_{i=1}^{k} p_i^2$ where p is the allele frequencies at a given locus and n is the number of observed alleles (Nei, 1978) in each locus and then returning the average. Caution should be exercised in interpreting the results of Hexp with polyploid organisms with ambiguous ploidy. The lack of allelic dosage information will cause rare alleles to be over-represented and artificially inflate the index. This is especially true with small sample sizes.

Author(s)

Zhian N. Kamvar

References

Paul-Michael Agapow and Austin Burt. Indices of multilocus linkage disequilibrium. *Molecular Ecology Notes*, 1(1-2):101-102, 2001

A.H.D. Brown, M.W. Feldman, and E. Nevo. Multilocus structure of natural populations of *Hordeum spontaneum*. *Genetics*, 96(2):523-536, 1980.

Niklaus J. Gr\"unwald, Stephen B. Goodwin, Michael G. Milgroom, and William E. Fry. Analysis of genotypic diversity data for populations of microorganisms. Phytopathology, 93(6):738-46, 2003

Bernhard Haubold and Richard R. Hudson. Lian 3.0: detecting linkage disequilibrium in multilocus data. Bioinformatics, 16(9):847-849, 2000.

Kenneth L.Jr. Heck, Gerald van Belle, and Daniel Simberloff. Explicit calculation of the rarefaction diversity measurement and the determination of sufficient sample size. Ecology, 56(6):pp. 1459-1461, 1975

Masatoshi Nei. Estimation of average heterozygosity and genetic distance from a small number of individuals. Genetics, 89(3):583-590, 1978.

S H Hurlbert. The nonconcept of species diversity: a critique and alternative parameters. Ecology, 52(4):577-586, 1971.

J.A. Ludwig and J.F. Reynolds. Statistical Ecology. A Primer on Methods and Computing. New York USA: John Wiley and Sons, 1988.

Simpson, E. H. Measurement of diversity. Nature 163: 688, 1949 doi:10.1038/163688a0

Good, I. J. (1953). On the Population Frequency of Species and the Estimation of Population Parameters. *Biometrika* 40(3/4): 237-264.

Lande, R. (1996). Statistics and partitioning of species diversity, and similarity among multiple communities. *Oikos* 76: 5-13.

Jari Oksanen, F. Guillaume Blanchet, Roeland Kindt, Pierre Legendre, Peter R. Minchin, R. B. O'Hara, Gavin L. Simpson, Peter Solymos, M. Henry H. Stevens, and Helene Wagner. vegan: Community Ecology Package, 2012. R package version 2.0-5.

E.C. Pielou. Ecological Diversity. Wiley, 1975.

Claude Elwood Shannon. A mathematical theory of communication. Bell Systems Technical Journal, 27:379-423,623-656, 1948

J M Smith, N H Smith, M O'Rourke, and B G Spratt. How clonal are bacteria? Proceedings of the National Academy of Sciences, 90(10):4384-4388, 1993.

J.A. Stoddart and J.F. Taylor. Genotypic diversity: estimation and prediction in samples. Genetics, 118(4):705-11, 1988.

See Also

clonecorrect(), poppr.all(), ia(), missingno(), mlg(), diversity_stats(), diversity_ci()

Examples

```
data(nancycats)
poppr(nancycats)

## Not run:
# Sampling
poppr(nancycats, sample = 999, total = FALSE, plot = TRUE)

# Customizing the plot
library("ggplot2")
p <- last_plot()
p + facet_wrap(~population, scales = "free_y", ncol = 1)

# Turning off diversity statistics (see get_stats)
poppr(nancycats, total=FALSE, H = FALSE, G = FALSE, lambda = FALSE, E5 = FALSE)

# The previous version of poppr contained a definition of Hexp, which
# was calculated as (N/(N - 1))*lambda. It basically looks like an unbiased
# Simpson's index. This statistic was originally included in poppr because it
# was originally included in the program multilocus. It was finally figured
# to be an unbiased Simpson's diversity metric (Lande, 1996; Good, 1953).

data(Aeut)

uSimp <- function(x){
  lambda <- vegan::diversity(x, "simpson")
  x <- drop(as.matrix(x))
  if (length(dim(x)) > 1){
    N <- rowSums(x)
  } else {
    N <- sum(x)
  }
  return((N/(N-1))*lambda)
}
poppr(Aeut, uSimp = uSimp)


# Demonstration with viral data
```

```
# Note: this is a larger data set that could take a couple of minutes to run
# on slower computers.
data(H3N2)
strata(H3N2) <- data.frame(other(H3N2)$x)
setPop(H3N2) <- ~country
poppr(H3N2, total = FALSE, sublist=c("Austria", "China", "USA"),
  clonecorrect = TRUE, strata = ~country/year)

## End(Not run)
```

`poppr.all` *Process a list of files with poppr*

Description

poppr.all is a wrapper function that will loop through a list of files from the working directory, execute [poppr()], and concatenate the output into one data frame.

Usage

```
poppr.all(filelist, ...)
```

Arguments

`filelist`	a list of files in the current working directory
`...`	arguments passed on to poppr

Value

see [poppr()]

Author(s)

Zhian N. Kamvar

See Also

[poppr()], [getfile()]

Examples

```
## Not run:
# Obtain a list of fstat files from a directory.
x <- getfile(multi=TRUE, pattern="^.+?dat$")

# run the analysis on each file.
poppr.all(file.path(x$path, x$files))

## End(Not run)
```

poppr.amova *Perform Analysis of Molecular Variance (AMOVA) on genind or genclone objects.*

Description

This function simplifies the process necessary for performing AMOVA in R. It gives user the choice of utilizing either the **ade4** or the **pegas** implementation of AMOVA. See `ade4::amova()` (ade4) and `pegas::amova()` (pegas) for details on the specific implementation.

Usage

```
poppr.amova(
  x,
  hier = NULL,
  clonecorrect = FALSE,
  within = TRUE,
  dist = NULL,
  squared = TRUE,
  freq = TRUE,
  correction = "quasieuclid",
  sep = "_",
  filter = FALSE,
  threshold = 0,
  algorithm = "farthest_neighbor",
  threads = 1L,
  missing = "loci",
  cutoff = 0.05,
  quiet = FALSE,
  method = c("ade4", "pegas"),
  nperm = 0
)
```

Arguments

x	a genind, genclone, genlight, or snpclone object
hier	a hierarchical formula that defines your population hierarchy. (e.g.: `~Population/Subpopulation`). **See Details below**.
clonecorrect	`logical` if `TRUE`, the data set will be clone corrected with respect to the lowest level of the hierarchy. The default is set to `FALSE`. See `clonecorrect()` for details.
within	`logical`. When this is set to `TRUE` (Default), variance within individuals are calculated as well. If this is set to `FALSE`, The lowest level of the hierarchy will be the sample level. See Details below.
dist	an optional distance matrix calculated on your data. If this is set to `NULL` (default), the raw pairwise distances will be calculated via `dist()`.

squared — if a distance matrix is supplied, this indicates whether or not it represents squared distances.

freq — logical. If within = FALSE, the parameter rho is calculated (Ronfort et al. 1998; Meirmans and Liu 2018). By setting freq = TRUE, (default) allele counts will be converted to frequencies before the distance is calculated, otherwise, the distance will be calculated on allele counts, which can bias results in mixed-ploidy data sets. Note that this option has no effect for haploid or presence/absence data sets.

correction — a character defining the correction method for non-euclidean distances. Options are ade4::quasieuclid() (Default), ade4::lingoes(), and ade4::cailliez(). See Details below.

sep — Deprecated. As of poppr version 2, this argument serves no purpose.

filter — logical When set to TRUE, mlg.filter will be run to determine genotypes from the distance matrix. It defaults to FALSE. You can set the parameters with algorithm and threshold arguments. Note that this will not be performed when within = TRUE. Note that the threshold should be the number of allowable substitutions if you don't supply a distance matrix.

threshold — a number indicating the minimum distance two MLGs must be separated by to be considered different. Defaults to 0, which will reflect the original (naive) MLG definition.

algorithm — determines the type of clustering to be done.

- **"farthest_neighbor"** *(default)* merges clusters based on the maximum distance between points in either cluster. This is the strictest of the three.
- **"nearest_neighbor"** merges clusters based on the minimum distance between points in either cluster. This is the loosest of the three.
- **"average_neighbor"** merges clusters based on the average distance between every pair of points between clusters.

threads — integer When using filtering or genlight objects, this parameter specifies the number of parallel processes passed to mlg.filter() and/or bitwise.dist().

missing — specify method of correcting for missing data utilizing options given in the function missingno(). Default is "loci". This only applies to genind or genclone objects.

cutoff — specify the level at which missing data should be removed/modified. See missingno() for details. This only applies to genind or genclone objects.

quiet — logical If FALSE (Default), messages regarding any corrections will be printed to the screen. If TRUE, no messages will be printed.

method — Which method for calculating AMOVA should be used? Choices refer to package implementations: "ade4" (default) or "pegas". See details for differences.

nperm — the number of permutations passed to the pegas implementation of amova.

Details

The poppr implementation of AMOVA is a very detailed wrapper for the ade4 implementation. The output is an ade4::amova() class list that contains the results in the first four elements. The inputs are contained in the last three elements. The inputs required for the ade4 implementation are:

1. a distance matrix on all unique genotypes (haplotypes)
2. a data frame defining the hierarchy of the distance matrix
3. a genotype (haplotype) frequency table.

All of this data can be constructed from a genind or genlight object, but can be daunting for a novice R user. *This function automates the entire process*. Since there are many variables regarding genetic data, some points need to be highlighted:

On Hierarchies:: The hierarchy is defined by different population strata that separate your data hierarchically. These strata are defined in the **strata** slot of genind, genlight, genclone, and snpclone objects. They are useful for defining the population factor for your data. See the function `adegenet::strata()` for details on how to properly define these strata.

On Within Individual Variance:: Heterozygosities within genotypes are sources of variation from within individuals and can be quantified in AMOVA. When `within = TRUE`, poppr will split genotypes into haplotypes with the function `make_haplotypes()` and use those to calculate within-individual variance. No estimation of phase is made. This acts much like the default settings for AMOVA in the Arlequin software package. Within individual variance will not be calculated for haploid individuals or dominant markers as the haplotypes cannot be split further. Setting `within = FALSE` uses the euclidean distance of the allele frequencies within each individual. **Note:** `within = TRUE` is incompatible with `filter = TRUE`. In this case, `within` will be set to `FALSE`

On Euclidean Distances:: With the **ade4** implementation of AMOVA (utilized by **poppr**), distances must be Euclidean (due to the nature of the calculations). Unfortunately, many genetic distance measures are not always euclidean and must be corrected for before being analyzed. Poppr automates this with three methods implemented in **ade4**, `ade4::quasieuclid()`, `ade4::lingoes()`, and `ade4::cailliez()`. The correction of these distances should not adversely affect the outcome of the analysis.

On Filtering:: Filtering multilocus genotypes is performed by `mlg.filter()`. This can necessarily only be done AMOVA tests that do not account for within-individual variance. The distance matrix used to calculate the amova is derived from using `mlg.filter()` with the option `stats = "distance"`, which reports the distance between multilocus genotype clusters. One useful way to utilize this feature is to correct for genotypes that have equivalent distance due to missing data. (See example below.)

On Methods:: Both **ade4** and **pegas** have implementations of AMOVA, both of which are appropriately called "amova". The ade4 version is faster, but there have been questions raised as to the validity of the code utilized. The pegas version is slower, but careful measures have been implemented as to the accuracy of the method. It must be noted that there appears to be a bug regarding permuting analyses where within individual variance is accounted for (`within = TRUE`) in the pegas implementation. If you want to perform permutation analyses on the pegas implementation, you must set `within = FALSE`. In addition, while clone correction is implemented for both methods, filtering is only implemented for the ade4 version.

On Polyploids:: As of **poppr** version 2.7.0, this function is able to calculate phi statistics for within-individual variance for polyploid data with **full dosage information**. When a data set does not contain full dosage information for all samples, then the resulting pseudo-haplotypes will contain missing data, which would result in an incorrect estimate of variance.

Instead, the AMOVA will be performed on the distance matrix derived from allele counts or allele frequencies, depending on the `freq` option. This has been shown to be robust to estimates with mixed ploidy (Ronfort et al. 1998; Meirmans and Liu 2018). If you wish to brute-force your way to estimating AMOVA using missing values, you can split your haplotypes with the `make_haplotypes()` function.

One strategy for addressing ambiguous dosage in your polyploid data set would be to convert your data to **polysat**'s `genambig` class with the `as.genambig()`, estimate allele frequencies with `polysat::deSilvaFreq()`, and use these frequencies to randomly sample alleles to fill in the ambiguous alleles.

Value

a list of class `amova` from the ade4 or pegas package. See `ade4::amova()` or `pegas::amova()` for details.

References

Excoffier, L., Smouse, P.E. and Quattro, J.M. (1992) Analysis of molecular variance inferred from metric distances among DNA haplotypes: application to human mitochondrial DNA restriction data. *Genetics*, **131**, 479-491.

Ronfort, J., Jenczewski, E., Bataillon, T., and Rousset, F. (1998). Analysis of population structure in autotetraploid species. *Genetics*, **150**, 921–930.

Meirmans, P., Liu, S. (2018) Analysis of Molecular Variance (AMOVA) for Autopolyploids *Submitted*.

See Also

`ade4::amova()`, `pegas::amova()`, `clonecorrect()`, `diss.dist()`, `missingno()`, `ade4::is.euclid()`, `adegenet::strata()`, `make_haplotypes()`, `as.genambig()`

Examples

```
data(Aeut)
strata(Aeut) <- other(Aeut)$population_hierarchy[-1]
agc <- as.genclone(Aeut)
agc
amova.result <- poppr.amova(agc, ~Pop/Subpop)
amova.result
amova.test <- randtest(amova.result) # Test for significance
plot(amova.test)
amova.test

## Not run:

# You can get the same results with the pegas implementation
amova.pegas <- poppr.amova(agc, ~Pop/Subpop, method = "pegas")
amova.pegas
amova.pegas$varcomp/sum(amova.pegas$varcomp)

# Clone correction is possible
```

```
amova.cc.result <- poppr.amova(agc, ~Pop/Subpop, clonecorrect = TRUE)
amova.cc.result
amova.cc.test <- randtest(amova.cc.result)
plot(amova.cc.test)
amova.cc.test


# Example with filtering
data(monpop)
splitStrata(monpop) <- ~Tree/Year/Symptom
poppr.amova(monpop, ~Symptom/Year) # gets a warning of zero distances
poppr.amova(monpop, ~Symptom/Year, filter = TRUE, threshold = 0.1) # no warning



## End(Not run)
```

---

poppr.msn *Create a minimum spanning network of selected populations using a distance matrix.*

Description

Create a minimum spanning network of selected populations using a distance matrix.

Usage

```
poppr.msn(
  gid,
  distmat,
  palette = topo.colors,
  mlg.compute = "original",
  sublist = "All",
  exclude = NULL,
  blacklist = NULL,
  vertex.label = "MLG",
  gscale = TRUE,
  glim = c(0, 0.8),
  gadj = 3,
  gweight = 1,
  wscale = TRUE,
  showplot = TRUE,
  include.ties = FALSE,
  threshold = NULL,
  clustering.algorithm = NULL,
  ...
)
```

Arguments

gid	a genind, genclone, genlight, or snpclone object
distmat	a distance matrix that has been derived from your data set.
palette	a vector or function defining the color palette to be used to color the populations on the graph. It defaults to topo.colors. See examples for details.
mlg.compute	if the multilocus genotypes are set to "custom" (see mll.custom for details) in your genclone object, this will specify which mlg level to calculate the nodes from. See details.
sublist	a vector of population names or indexes that the user wishes to keep. Default to "ALL".
exclude	a vector of population names or indexes that the user wishes to discard. Default to NULL.
blacklist	DEPRECATED, use exclude.
vertex.label	a vector of characters to label each vertex. There are two defaults: "MLG" will label the nodes with the multilocus genotype from the original data set and "inds" will label the nodes with the representative individual names.
gscale	"grey scale". If this is TRUE, this will scale the color of the edges proportional to the observed distance, with the lines becoming darker for more related nodes. See greycurve for details.
glim	"grey limit". Two numbers between zero and one. They determine the upper and lower limits for the gray function. Default is 0 (black) and 0.8 (20% black). See greycurve for details.
gadj	"grey adjust". a positive integer greater than zero that will serve as the exponent to the edge weight to scale the grey value to represent that weight. See greycurve for details.
gweight	"grey weight". an integer. If it's 1, the grey scale will be weighted to emphasize the differences between closely related nodes. If it is 2, the grey scale will be weighted to emphasize the differences between more distantly related nodes. See greycurve for details.
wscale	"width scale". If this is TRUE, the edge widths will be scaled proportional to the inverse of the observed distance , with the lines becoming thicker for more related nodes.
showplot	logical. If TRUE, the graph will be plotted. If FALSE, it will simply be returned.
include.ties	logical. If TRUE, the graph will include all edges that were arbitrarily passed over in favor of another edge of equal weight. If FALSE, which is the default, one edge will be arbitrarily selected when two or more edges are tied, resulting in a pure minimum spanning network.
threshold	numeric. By default, this is NULL, which will have no effect. Any threshold value passed to this argument will be used in mlg.filter prior to creating the MSN. If you have a data set that contains contracted MLGs, this argument will override the threshold in the data set. See Details.
clustering.algorithm	string. By default, this is NULL. If threshold = NULL, this argument will have no effect. When supplied with either "farthest_neighbor", "average_neighbor", or

"nearest_neighbor", it will be passed to `mlg.filter` prior to creating the MSN. If you have a data set that contains contracted MLGs, this argument will override the algorithm in the data set. See Details.

`...` any other arguments that could go into plot.igraph

Details

The minimum spanning network generated by this function is generated via igraph's `minimum.spanning.tree`. The resultant graph produced can be plotted using igraph functions, or the entire object can be plotted using the function `plot_poppr_msn`, which will give the user a scale bar and the option to layout your data.

node sizes: The area of the nodes are representative of the number of samples. Because **igraph** scales nodes by radius, the node sizes in the graph are represented as the square root of the number of samples.

mlg.compute: Each node on the graph represents a different multilocus genotype. The edges on the graph represent genetic distances that connect the multilocus genotypes. In genclone objects, it is possible to set the multilocus genotypes to a custom definition. This creates a problem for clone correction, however, as it is very possible to define custom lineages that are not monophyletic. When clone correction is performed on these definitions, information is lost from the graph. To circumvent this, The clone correction will be done via the computed multilocus genotypes, either "original" or "contracted". This is specified in the `mlg.compute` argument, above.

contracted multilocus genotypes: If your incoming data set is of the class `genclone`, and it contains contracted multilocus genotypes, this function will retain that information for creating the minimum spanning network. You can use the arguments `threshold` and `clustering.algorithm` to change the threshold or clustering algorithm used in the network. For example, if you have a data set that has a threshold of 0.1 and you wish to have a minimum spanning network without a threshold, you can simply add `threshold = 0.0`, and no clustering will happen.

The `threshold` and `clustering.algorithm` arguments can also be used to filter un-contracted data sets.

All filtering will use the distance matrix supplied in the argument `distmat`.

Value

`graph` a minimum spanning network with nodes corresponding to MLGs within the data set. Colors of the nodes represent population membership. Width and color of the edges represent distance.

`populations` a vector of the population names corresponding to the vertex colors

`colors` a vector of the hexadecimal representations of the colors used in the vertex colors

Note

The edges of these graphs may cross each other if the graph becomes too large.

Author(s)

Javier F. Tabima, Zhian N. Kamvar, Jonah C. Brooks

See Also

plot_poppr_msn nancycats, upgma, nj, nodelabels, tab, missingno, bruvo.msn, greycurve

Examples

```
# Load the data set and calculate the distance matrix for all individuals.
data(Aeut)
A.dist <- diss.dist(Aeut)

# Graph it.
A.msn <- poppr.msn(Aeut, A.dist, gadj = 15, vertex.label = NA)

# Find the sizes of the nodes (number of individuals per MLL):
igraph::vertex_attr(A.msn$graph, "size")^2

## Not run:
# Set subpopulation structure.
Aeut.sub <- as.genclone(Aeut)
setPop(Aeut.sub) <- ~Pop/Subpop

# Plot respective to the subpopulation structure
As.msn <- poppr.msn(Aeut.sub, A.dist, gadj=15, vertex.label=NA)

# Show only the structure of the Athena population.
As.msn <- poppr.msn(Aeut.sub, A.dist, gadj=15, vertex.label=NA, sublist=1:10)

# Let's look at the structure of the microbov data set

library("igraph")
data(microbov)
micro.dist <- diss.dist(microbov, percent = TRUE)
micro.msn <- poppr.msn(microbov, micro.dist, vertex.label=NA)

# Let's plot it and show where individuals have < 15% of their genotypes
# different.

edge_weight <- E(micro.msn$graph)$weight
edge_labels <- ifelse(edge_weight < 0.15, round(edge_weight, 3), NA)
plot.igraph(micro.msn$graph, edge.label = edge_labels, vertex.size = 2,
edge.label.color = "red")


## End(Not run)
```

---

poppr_has_parallel *Determines whether openMP is support on this system.*

Description

Determines whether openMP is support on this system.

Usage

```
poppr_has_parallel()
```

Value

FALSE if openMP is not supported, TRUE if it is

Author(s)

Zhian N. Kamvar, Jonah C. Brooks

Examples

```
poppr_has_parallel()
```

---

popsub *Subset data by population*

Description

Create a new dataset with specified populations or exclude specified populations from the dataset.

Usage

```
popsub(
  gid,
  sublist = "ALL",
  exclude = NULL,
  blacklist = NULL,
  mat = NULL,
  drop = TRUE
)
```

Arguments

gid	a genind, genclone, genlight, or snpclone object.
sublist	a vector of population names or indexes that the user wishes to keep. Default to "ALL".
exclude	a vector of population names or indexes that the user wishes to discard. Default to NULL.
blacklist	DEPRECATED, use exclude.
mat	a matrix object produced by mlg.table to be subsetted. If this is present, the subsetted matrix will be returned instead of the genind object
drop	logical. If TRUE, unvarying alleles will be dropped from the population.

Value

A genind object or a matrix.

Author(s)

Zhian N. Kamvar

Examples

```
# Load the dataset microbov.
data(microbov)

# List the population names.
popNames(microbov)

# Analyze only the populations with exactly 50 individuals
mic.50 <- popsub(microbov, sublist=c(1:6, 11:15), exclude=c(3,4,13,14))

## Not run:
# Analyze the first 10 populations, except for "Bazadais"
mic.10 <- popsub(microbov, sublist=1:10, exclude="Bazadais")

# Take out the two smallest populations
micbig <- popsub(microbov, exclude=c("NDama", "Montbeliard"))

# Analyze the two largest populations
miclrg <- popsub(microbov, sublist=c("BlondeAquitaine", "Charolais"))

## End(Not run)
```

Pram *Phytophthora ramorum data from OR Forests and Nurseries (OR and CA)*

Description

This is the data set from doi:10.5281/zenodo.13007. It has been converted to the genclone object as of poppr version 2.0. It contains 729 samples of the Sudden Oak Death pathogen *Phytophthora ramorum* genotyped over five microsatellite loci (Kamvar et. al., 2015). 513 samples were collected from forests in Curry County, OR from 2001 to mid-2014 (labeled by watershed region). The other 216 samples represents genotypes collected from Nurseries in OR and CA from Goss et. al. (2009).

Usage

```
data(Pram)
```

Format

a [genclone-class] object with 3 hierarchical levels called "SOURCE", "YEAR", and, "STATE". The **other** slot contains a named vector of repeat lengths called **"REPLEN"**, a matrix of xy coordinates for the forest samples called **"xy"**, and a palette to color the ~SOURCE/STATE stratification called **"comparePal"**.

References

Kamvar, Z. N., Larsen, M. M., Kanaskie, A. M., Hansen, E. M., & Grünwald, N. J. (2015). Spatial and temporal analysis of populations of the sudden oak death pathogen in Oregon forests. Phytopathology 105:982-989. doi: doi:10.1094/PHYTO12140350FI

Zhian N. Kamvar, Meg M. Larsen, Alan M. Kanaskie, Everett M. Hansen, & Niklaus J. Grünwald. 2014. Sudden_Oak_Death_in_Oregon_Forests: Spatial and temporal population dynamics of the sudden oak death epidemic in Oregon Forests. ZENODO, doi: doi:10.5281/zenodo.13007

Goss, E. M., Larsen, M., Chastagner, G. A., Givens, D. R., and Grünwald, N. J. 2009. Population genetic analysis infers migration pathways of *Phytophthora ramorum* in US nurseries. PLoS Pathog. 5:e1000583. doi: doi:10.1371/journal.ppat.1000583

Examples

```
data(Pram)

# Repeat lengths (previously processed via fix_replen)
other(Pram)$REPLEN

# Color palette for source by state. Useful for minimum spanning networks
other(Pram)$comparePal
```

---

private_alleles *Tabulate alleles the occur in only one population.*

Description

Tabulate alleles the occur in only one population.

Usage

```
private_alleles(
  gid,
  form = alleles ~ .,
  report = "table",
  level = "population",
  count.alleles = TRUE,
  drop = FALSE
)
```

Arguments

gid	a adegenet::genind or genclone object.
form	a formula() giving the levels of markers and hierarchy to analyze. See Details.
report	one of "table", "vector", or "data.frame". Tables (Default) and data frame will report counts along with populations or individuals. Vectors will simply report which populations or individuals contain private alleles. Tables are matrices with populations or individuals in rows and alleles in columns. Data frames are long form.
level	one of "population" (Default) or "individual".
count.alleles	logical. If TRUE (Default), The report will return the observed number of alleles private to each population. If FALSE, each private allele will be counted once, regardless of dosage.
drop	logical. if TRUE, populations/individuals without private alleles will be dropped from the result. Defaults to FALSE.

Details

the argument form allows for control over the strata at which private alleles should be computed. It takes a form where the left hand side of the formula can be either "allele", "locus", or "loci". The right hand of the equation, by default is ".". If you change it, it must correspond to strata located in the adegenet::strata() slot. Note, that the right hand side is disabled for genpop objects.

Value

a matrix, data.frame, or vector defining the populations or individuals containing private alleles. If vector is chosen, alleles are not defined.

Author(s)

Zhian N. Kamvar

Examples

```
data(Pinf) # Load P. infestans data.
private_alleles(Pinf)

## Not run:
# Analyze private alleles based on the country of interest:
private_alleles(Pinf, alleles ~ Country)

# Number of observed alleles per locus
private_alleles(Pinf, locus ~ Country, count.alleles = TRUE)

# Get raw number of private alleles per locus.
(pal <- private_alleles(Pinf, locus ~ Country, count.alleles = FALSE))

# Get percentages.
sweep(pal, 2, nAll(Pinf)[colnames(pal)], FUN = "/")
```

```
# An example of how these data can be displayed.
library("ggplot2")
Pinfpriv <- private_alleles(Pinf, report = "data.frame")
ggplot(Pinfpriv) + geom_tile(aes(x = population, y = allele, fill = count))

## End(Not run)
```

---

psex *Probability of encountering a genotype more than once by chance*

Description

Probability of encountering a genotype more than once by chance

Usage

```
psex(
  gid,
  pop = NULL,
  by_pop = TRUE,
  freq = NULL,
  G = NULL,
  method = c("single", "multiple"),
  ...
)
```

Arguments

gid	a genind or genclone object.
pop	either a formula to set the population factor from the strata slot or a vector specifying the population factor for each sample. Defaults to NULL.
by_pop	When this is TRUE (default), the calculation will be done by population.
freq	a vector or matrix of allele frequencies. This defaults to NULL, indicating that the frequencies will be determined via round-robin approach in rraf. **If this matrix or vector is not provided, zero-value allele frequencies will automatically be corrected.** For details, please see the documentation on correcting rare alleles.
G	an integer vector specifying the number of observed genets. If NULL, this will be the number of original multilocus genotypes for method = "single" and the number of populations for method = "multiple". G can also be a **named** integer vector for each population if by_pop = TRUE. Unnamed vectors with a lengths greater than 1 will throw an error.
method	which method of calculating psex should be used? Using method = "single" (default) indicates that the calculation for psex should reflect the probability of encountering a second genotype. Using method = "multiple" gives the probability of encountering multiple samples of the same genotype (see details).
...	options from correcting rare alleles. The default is to correct allele frequencies to 1/n

Details

single encounter:: Psex is the probability of encountering a given genotype more than once by chance. The basic equation from Parks and Werth (1993) is

$$p_{sex} = 1 - (1 - p_{gen})^G)$$

where *G* is the number of multilocus genotypes and *pgen* is the probability of a given genotype (see `pgen` for its calculation). For a given value of alpha (e.g. alpha = 0.05), genotypes with psex < alpha can be thought of as a single genet whereas genotypes with psex > alpha do not have strong evidence that members belong to the same genet (Parks and Werth, 1993).

multiple encounters:: When `method = "multiple"`, the method from Arnaud-Haond et al. (1997) is used where the sum of the binomial density is taken.

$$p_{sex} = \sum_{i=n}^{N} \binom{N}{i} (p_{gen})^i (1 - p_{gen})^{N-i}$$

where *N* is the number of sampling units *i* is the ith - 1 encounter of a given genotype, and *pgen* is the value of pgen for that genotype. This procedure is performed for all samples in the data. For example, if you have a genotype whose pgen value was 0.0001, with 5 observations out of 100 samples, the value of psex is computed like so:

```
dbinom(0:4, 100, 0.0001)
```

using by_pop = TRUE and modifying G:: It is possible to modify `G` for single or multiple encounters. With `method = "single"`, `G` takes place of the exponent, whereas with `method = "multiple"`, `G` replaces `N` (see above). If you supply a named vector for `G` with the population names and `by_pop = TRUE`, then the value of `G` will be different for each population.

For example, in the case of `method = "multiple"`, let's say you have two populations that share a genotype between them. The size of population A and B are 25 and 75, respectively, The values of pgen for that genotype in population A and B are 0.005 and 0.0001, respectively, and the number of samples with the genotype in popualtions A and B are 4 and 6, respectively. In this case psex for this genotype would be calculated for each population separately if we don't specify `G`:

```
psexA = dbinom(0:3, 25, 0.005)
psexB = dbinom(0:5, 75, 0.0001)
```

If we specify `G = 100`, then it changes to:

```
psexA = dbinom(0:3, 100, 0.005)
psexB = dbinom(0:5, 100, 0.0001)
```

We could also specify G to be the number of genotypes observed in the population (let's say A = 10, B = 20)

```
psexA = dbinom(0:3, 10, 0.005)
psexB = dbinom(0:5, 20, 0.0001)
```

Unless `freq` is supplied, the function will automatically calculate the round-robin allele frequencies with `rraf` and *G* with `nmll`.

Value

a vector of Psex for each sample.

Note

The values of Psex represent the value for each multilocus genotype. Additionally, when the argument pop is not NULL, by_pop is automatically TRUE.

Author(s)

Zhian N. Kamvar, Jonah Brooks, Stacy A. Krueger-Hadfield, Erik Sotka

References

Arnaud-Haond, S., Duarte, C. M., Alberto, F., & Serrão, E. A. 2007. Standardizing methods to address clonality in population studies. *Molecular Ecology*, 16(24), 5115-5139.

Parks, J. C., & Werth, C. R. 1993. A study of spatial features of clones in a population of bracken fern, *Pteridium aquilinum* (Dennstaedtiaceae). *American Journal of Botany*, 537-544.

See Also

pgen, rraf, rrmlg, rare_allele_correction

Examples

```
data(Pram)

# With multiple encounters
Pram_psex <- psex(Pram, by_pop = FALSE, method = "multiple")
plot(Pram_psex, log = "y", col = ifelse(Pram_psex > 0.05, "red", "blue"))
abline(h = 0.05, lty = 2)
title("Probability of multiple encounters")
## Not run:

# For a single encounter (default)
Pram_psex <- psex(Pram, by_pop = FALSE)
plot(Pram_psex, log = "y", col = ifelse(Pram_psex > 0.05, "red", "blue"))
abline(h = 0.05, lty = 2)
title("Probability of second encounter")

# This can be also done assuming populations structure
Pram_psex <- psex(Pram, by_pop = TRUE, method = "multiple")
plot(Pram_psex, log = "y", col = ifelse(Pram_psex > 0.05, "red", "blue"))
abline(h = 0.05, lty = 2)
title("Probability of multiple encounters\nwith pop structure")

# The above, but correcting zero-value alleles by 1/(2*rrmlg) with no
# population structure assumed
# Type ?rare_allele_correction for details.
Pram_psex2 <- psex(Pram, by_pop = FALSE, d = "rrmlg", mul = 1/2, method = "multiple")
plot(Pram_psex2, log = "y", col = ifelse(Pram_psex2 > 0.05, "red", "blue"))
```

```
abline(h = 0.05, lty = 2)
title("Probability of multiple encounters\nwith pop structure (1/(2*rrmlg))")

# We can also set G to the total population size
(G <- nInd(Pram))
Pram_psex <- psex(Pram, by_pop = TRUE, method = "multiple", G = G)
plot(Pram_psex, log = "y", col = ifelse(Pram_psex > 0.05, "red", "blue"))
abline(h = 0.05, lty = 2)
title("Probability of multiple encounters\nwith pop structure G = 729")

# Or we can set G to the number of unique MLGs
(G <- rowSums(mlg.table(Pram, plot = FALSE) > 0))
Pram_psex <- psex(Pram, by_pop = TRUE, method = "multiple", G = G)
plot(Pram_psex, log = "y", col = ifelse(Pram_psex > 0.05, "red", "blue"))
abline(h = 0.05, lty = 2)
title("Probability of multiple encounters\nwith pop structure G = nmll")

## An example of supplying previously calculated frequencies and G
# From Parks and Werth, 1993, using the first three genotypes.

# The row names indicate the number of samples found with that genotype
x <- "
 Hk Lap Mdh2 Pgm1 Pgm2 X6Pgd2
54 12 12 12 23 22 11
36 22 22 11 22 33 11
10 23 22 11 33 13 13"

# Since we aren't representing the whole data set here, we are defining the
# allele frequencies before the analysis.
afreq <- c(Hk.1 = 0.167, Hk.2 = 0.795, Hk.3 = 0.038,
           Lap.1 = 0.190, Lap.2 = 0.798, Lap.3 = 0.012,
           Mdh2.0 = 0.011, Mdh2.1 = 0.967, Mdh2.2 = 0.022,
           Pgm1.2 = 0.279, Pgm1.3 = 0.529, Pgm1.4 = 0.162, Pgm1.5 = 0.029,
           Pgm2.1 = 0.128, Pgm2.2 = 0.385, Pgm2.3 = 0.487,
           X6Pgd2.1 = 0.526, X6Pgd2.2 = 0.051, X6Pgd2.3 = 0.423)

xtab <- read.table(text = x, header = TRUE, row.names = 1)

# Here we are expanding the number of samples to their observed values.
# Since we have already defined the allele frequencies, this step is actually
# not necessary.
all_samples <- rep(rownames(xtab), as.integer(rownames(xtab)))
xgid        <- df2genind(xtab[all_samples, ], ncode = 1)

freqs <- afreq[colnames(tab(xgid))] # only used alleles in the sample
pSex  <- psex(xgid, by_pop = FALSE, freq = freqs, G = 45)

# Note, pgen returns log values for each locus, here we take the sum across
# all loci and take the exponent to give us the value of pgen for each sample
pGen <- exp(rowSums(pgen(xgid, by_pop = FALSE, freq = freqs)))

res  <- matrix(c(unique(pGen), unique(pSex)), ncol = 2)
colnames(res) <- c("Pgen", "Psex")
```

```
res <- cbind(xtab, nRamet = rownames(xtab), round(res, 5))
rownames(res) <- 1:3
res # Compare to the first three rows of Table 2 in Parks & Werth, 1993

## End(Not run)
```

`rare_allele_correction`

Correcting rare allele frequencies

Description

The following is a set of arguments for use in `rraf`, `pgen`, and `psex` to correct rare allele frequencies that were lost in estimating round-robin allele frequencies.

Arguments

`e`	a numeric epsilon value to use for all missing allele frequencies.
`d`	the unit by which to take the reciprocal. `div = "sample"` will be 1/(n samples), `d = "mlg"` will be 1/(n mlg), and `d = "rrmlg"` will be 1/(n mlg at that locus). This is overridden by `e`.
`mul`	a multiplier for div. Default is `mul = 1`. This parameter is overridden by `e`
`sum_to_one`	when `TRUE`, the original frequencies will be reduced so that all allele frequencies will sum to one. **Default:** `FALSE`

Details

By default (`d = "sample", e = NULL, sum_to_one = FALSE, mul = 1`), this will add 1/(n samples) to all zero-value alleles. The basic formula is **1/(d * m)** unless **e** is specified. If `sum_to_one = TRUE`, then the frequencies will be scaled as x/sum(x) AFTER correction, indicating that the allele frequencies will be reduced. See the examples for details. The general pattern of correction is that the value of the MAF will be *rrmlg* > *mlg* > *sample*

Motivation

When calculating allele frequencies from a round-robin approach, rare alleles are often lost resulting in zero-valued allele frequencies (Arnaud-Haond et al. 2007, Parks and Werth 1993). This can be problematic when calculating values for `pgen` and `psex` because frequencies of zero will result in undefined values for samples that contain those rare alleles. The solution to this problem is to give an estimate for the frequency of those rare alleles, but the question of HOW to do that arises. These arguments provide a way to define how rare alleles are to be estimated/corrected.

Using these arguments

These arguments are for use in the functions rraf, pgen, and psex. They will replace the dots (...) that appear at the end of the function call. For example, if you want to set the minor allele frequencies to a specific value (let's say 0.001), regardless of locus, you can insert e = 0.001 along with any other arguments (note, position is not specific):

```
pgen(my_data, e = 0.001, log = FALSE)
psex(my_data, method = "multiple", e = 0.001)
```

Author(s)

Zhian N. Kamvar

References

Arnaud-Haond, S., Duarte, C. M., Alberto, F., & Serrão, E. A. 2007. Standardizing methods to address clonality in population studies. *Molecular Ecology*, 16(24), 5115-5139.

Parks, J. C., & Werth, C. R. 1993. A study of spatial features of clones in a population of bracken fern, *Pteridium aquilinum* (Dennstaedtiaceae). *American Journal of Botany*, 537-544.

See Also

rraf, pgen, psex, rrmlg

Examples

```
## Not run:

data(Pram)
#-------------------------------------

# If you set correction = FALSE, you'll notice the zero-valued alleles

rraf(Pram, correction = FALSE)

# By default, however, the data will be corrected by 1/n

rraf(Pram)

# Of course, this is a diploid organism, we might want to set 1/2n

rraf(Pram, mul = 1/2)

# To set MAF = 1/2mlg

rraf(Pram, d = "mlg", mul = 1/2)

# Another way to think about this is, since these allele frequencies were
# derived at each locus with different sample sizes, it's only appropriate to
# correct based on those sample sizes.
```

```
rraf(Pram, d = "rrmlg", mul = 1/2)

# If we were going to use these frequencies for simulations, we might want to
# ensure that they all sum to one.

rraf(Pram, d = "mlg", mul = 1/2, sum_to_one = TRUE)

#-------------------------------------
# When we calculate these frequencies based on population, they are heavily
# influenced by the number of observed mlgs.

rraf(Pram, by_pop = TRUE, d = "rrmlg", mul = 1/2)

# This can be fixed by specifying a specific value

rraf(Pram, by_pop = TRUE, e = 0.01)


## End(Not run)
```

`read.genalex` *Importing data from genalex formatted *.csv files.*

Description

read.genalex will read in a genalex-formatted file that has been exported in a comma separated format and will parse most types of genalex data. The output is a genclone-class or genind-class object.

Usage

```
read.genalex(
  genalex,
  ploidy = 2,
  geo = FALSE,
  region = FALSE,
  genclone = TRUE,
  sep = ",",
  recode = FALSE
)
```

Arguments

`genalex`	a *.csv file exported from genalex
`ploidy`	an integer to indicate the ploidy of the dataset
`geo`	indicates the presence of geographic data in the file. This data will be included in a data frame labeled `xy` in the `adegenet::other()` slot.
`region`	indicates the presence of regional data in the file.

genclone — when TRUE (default), the output will be a genclone object. When FALSE, the output will be a adegenet::genind object

sep — A character specifying the column separator of the data. Defaults to ",".

recode — **For polyploid data**: Do you want to recode your data to have varying ploidy? Default is FALSE, and the data will be returned with even ploidy where missing alleles are coded as "0". When TRUE, the data is run through the function `recode_polyploids()` before being returned. Note that this will prevent conversion to genpop objects in the future. See details.

Details

The resulting genclone-class or genind-class object will have a single strata defined in the strata slot. This will be called "Pop" and will reflect the population factor defined in the genalex input. If `region = TRUE`, a second column will be inserted and labeled "Region". If you have more than two strata within your data set, you should run the command `adegenet::splitStrata()` on your data set to define the unique stratifications.

FOR POLYPLOID (> 2n) DATA SETS: The genind object has an all-or-none approach to missing data. If a sample has missing data at a particular locus, then the entire locus is considered missing. This works for diploids and haploids where allelic dosage is unambiguous. For polyploids this poses a problem as much of the data set would be transformed into missing data. With this function, I have created a workaround.

When importing polyploid data sets, missing data is scored as "0" and kept within the genind object as an extra allele. This will break most analyses relying on allele frequencies*. All of the functions in poppr will work properly with these data sets as multilocus genotype analysis is agnostic of ploidy and we have written both Bruvo's distance and the index of association in such a way as to be able to handle polyploids presented in this manner.

* To restore functionality of analyses relying on allele frequencies, use the `recode_polyploids()` function.

Value

A genclone or adegenet::genind bject.

Note

This function cannot handle raw allele frequency data.

In the case that there are duplicated names within the file, this function will assume separate individuals and rename each one to a sequence of integers from 1 to the number of individuals. A vector of the original names will be saved in the `other` slot under `original_names`.

Author(s)

Zhian N. Kamvar

See Also

`genind2genalex()`, `clonecorrect()`, genclone, adegenet::genind, `recode_polyploids()`

Examples

```
## Not run:
Aeut <- read.genalex(system.file("files/rootrot.csv", package="poppr"))

genalex2 <- read.genalex("genalex2.csv", geo=TRUE)
# A genalex file with geographic coordinate data.

genalex3 <- read.genalex("genalex3.csv", region=TRUE)
# A genalex file with regional information.

genalex4 <- read.genalex("genalex4.csv", region=TRUE, geo=TRUE)
# A genalex file with both regional and geographic information.

## End(Not run)
```

`recode_polyploids` *Recode polyploid microsatellite data for use in frequency based statistics.*

Description

As the genind object requires ploidy to be consistent across loci, a workaround to importing polyploid data was to code missing alleles as "0" (for microsatellite data sets). The advantage of this is that users would be able to calculate Bruvo's distance, the index of association, and genotypic diversity statistics. The tradeoff was the fact that this broke all other analyses as they relied on allele frequencies and the missing alleles are treated as extra alleles. This function removes those alleles and returns a `genclone` or `genind` object where allele frequencies are coded based on the number of alleles observed at a single locus per individual. See the examples for more details.

Usage

```
recode_polyploids(poly, newploidy = FALSE, addzero = FALSE)
```

Arguments

`poly`	a `genclone`, `genind`, or `genpop` object that has a ploidy of > 2
`newploidy`	for genind or genclone objects: if `FALSE` (default), the user-defined ploidy will stay constant. if `TRUE`, the ploidy for each sample will be determined by the maximum ploidy observed for each genotype.
`addzero`	add zeroes onto genind or genclone objects with uneven ploidy? if `TRUE`, objects with uneven ploidies will have zeroes appended to all loci to allow conversion to genpop objects. Defaults to `FALSE`.

Details

The genind object has two caveats that make it difficult to work with polyploid data sets:

1. ploidy must be constant throughout the data set
2. missing data is treated as "all-or-none"

In an ideal world, polyploid genotypes would be just as unambiguous as diploid or haploid genotypes. Unfortunately, the world we live in is far from ideal and a genotype of AB in a tetraploid organism could be AAAB, AABB, or ABBB. In order to get polyploid data in to **adegenet** or **poppr**, we must code all loci to have the same number of allelic states as the ploidy or largest observed heterozygote (if ploidy is unknown). The way to do this is to insert zeroes to pad the alleles. So, to import two genotypes of:

NA	20	23	24
20	24	26	43

they should be coded as:

0	20	23	24
20	24	26	43

This zero is treated as an extra allele and is represented in the genind object as so:

0	**20**	**23**	**24**	**26**	**43**
1	1	1	1	0	0
0	1	0	1	1	1

This function remedies this problem by removing the zero column. The above table would become:

20	**23**	**24**	**26**	**43**
1	1	1	0	0
1	0	1	1	1

With this, the user is able to calculate frequency based statistics on the data set.

Value

a genclone, genind, or genpop object.

Note

This is an approximation, and a bad one at that. **Poppr** was not originally intended for polyploids, but with the inclusion of Bruvo's distance, it only made sense to attempt something beyond single use.

Author(s)

Zhian N. Kamvar

Examples

```
data(Pinf)
iPinf <- recode_polyploids(Pinf)

# Note that the difference between the number of alleles.
nAll(Pinf)
nAll(iPinf)

## Not run:
library("ape")

# Removing missing data.
setPop(Pinf) <- ~Country

# Calculating Rogers' distance.
rog <- rogers.dist(genind2genpop(Pinf))
irog <- rogers.dist(recode_polyploids(genind2genpop(Pinf)))

# We will now plot neighbor joining trees. Note the decreased distance in the
# original data.
plot(nj(rog), type = "unrooted")
add.scale.bar(lcol = "red", length = 0.02)
plot(nj(irog), type = "unrooted")
add.scale.bar(lcol = "red", length = 0.02)

## End(Not run)
```

---

rraf *Round Robin Allele Frequencies*

Description

This function utilizes rrmlg to calculate multilocus genotypes and then subsets each locus by the resulting MLGs to calculate the round-robin allele frequencies used for pgen and psex.

Usage

```
rraf(gid, pop = NULL, res = "list", by_pop = FALSE, correction = TRUE, ...)
```

Arguments

gid	a genind or genclone object
pop	either a formula to set the population factor from the strata slot or a vector specifying the population factor for each sample. Defaults to NULL.
res	either "list" (default), "vector", or "data.frame".
by_pop	When this is TRUE, the calculation will be done by population. Defaults to FALSE
correction	a logical indicating whether or not zero-valued allele frequencies should be corrected using the methods outlined in correcting rare alleles. (Default: TRUE)

... options from correcting rare alleles. The default is to correct allele frequencies to 1/n

Details

Calculating allele frequencies for clonal populations is a difficult task. Frequencies calculated on non-clone-corrected data suffer from bias due to non-independent samples. On the other hand, frequencies calculated on clone-corrected data artificially increases the significance of rare alleles. The method of round-robin allele frequencies as presented in Parks and Werth (1993) provides a method of calculating allele frequencies in a way that minimizes both of these effects.

Rare Alleles: Allele frequencies at a given locus are calculated based on samples that are **clone corrected without that locus**. When this happens, rare alleles have a high likelihood of dropping out, giving them a frequency of "0". For some analyses, this is a perfectly fine outcome, but for analyses such as pgen and psex, this could result in undefined values. Setting `correction = TRUE` will allow you to control how these zero-valued allele frequencies are corrected. For details, please see the documentation on correcting rare alleles and examples.

Value

a vector or list of allele frequencies

Note

When `by_pop = TRUE`, the output will be a matrix of allele frequencies. Additionally, when the argument `pop` is not `NULL`, `by_pop` is automatically `TRUE`.

Author(s)

Zhian N. Kamvar, Jonah Brooks, Stacy A. Krueger-Hadfield, Erik Sotka

References

Arnaud-Haond, S., Duarte, C. M., Alberto, F., & Serrão, E. A. 2007. Standardizing methods to address clonality in population studies. *Molecular Ecology*, 16(24), 5115-5139.

Parks, J. C., & Werth, C. R. 1993. A study of spatial features of clones in a population of bracken fern, *Pteridium aquilinum* (Dennstaedtiaceae). *American Journal of Botany*, 537-544.

See Also

rrmlg, pgen, psex, rare_allele_correction

Examples

```
data(Pram)

# Round robin allele frequencies, correcting zero-valued frequencies to 1/nInd(Pram)
rraf(Pram)


## Not run:
```

```
## Round robin allele frequencies will be different than observed

# Compare to without round robin:
PrLoc <- seploc(Pram, res = "mat") # get locus by matrix
lapply(PrLoc, colMeans, na.rm = TRUE)

# Without round robin, clone corrected:
Pcc    <- clonecorrect(Pram, strata = NA) # indiscriminantly clone correct
PccLoc <- seploc(Pcc, res = "mat")
lapply(PccLoc, colMeans, na.rm = TRUE)

## Different methods of obtaining round robin allele frequencies

# Get vector output.
rraf(Pram, res = "vector")

# Getting the output as a data frame allows us to use ggplot2 to visualize
(Prdf <- rraf(Pram, res = "data.frame"))
library("ggplot2")
ggplot(Prdf, aes(y = allele, x = frequency)) +
  geom_point() +
  facet_grid(locus ~ ., scale = "free_y", space = "free")

## Round Robin allele frequencies by population (matrix only)

# By default, allele frequencies will be corrected by 1/n per population
(Prbp <- rraf(Pram, by_pop = TRUE))

# This might be problematic because populations like PistolRSF_OR has a
# population size of four.

# By using the 'e' argument to rare_allele_correction, this can be set to a
# more reasonable value.
(Prbp <- rraf(Pram, by_pop = TRUE, e = 1/nInd(Pram)))




## End(Not run)
```

`rrmlg` *Round Robin Multilocus Genotypes*

Description

This function will mask each locus one by one and then calculate multilocus genotypes from the remaining loci in a round-robin fashion. This is used for calculating the round robin allele frequencies for pgen and psex.

Usage

```
rrmlg(gid)
```

Arguments

gid	a genind, genclone, or loci object.

Value

a matrix of multilocus genotype assignments by masked locus. There will be n rows and m columns where n = number of samples and m = number of loci.

Author(s)

Zhian N. Kamvar, Jonah Brooks, Stacy A. Krueger-Hadfield, Erik Sotka

References

Arnaud-Haond, S., Duarte, C. M., Alberto, F., & Serrão, E. A. 2007. Standardizing methods to address clonality in population studies. *Molecular Ecology*, 16(24), 5115-5139.

Parks, J. C., & Werth, C. R. 1993. A study of spatial features of clones in a population of bracken fern, *Pteridium aquilinum* (Dennstaedtiaceae). *American Journal of Botany*, 537-544.

See Also

rraf, pgen, psex

Examples

```
# Find out the round-robin multilocus genotype assignments for P. ramorum
data(Pram)
pmlg_rr <- rrmlg(Pram)
head(pmlg_rr)
## Not run:
# You can find out how many unique genotypes are found without each locus:

colSums(!apply(pmlg_rr, 2, duplicated))

## End(Not run)
```

`samp.ia` *Calculate random samples of the index of association for genlight objects.*

Description

Genlight objects can contain millions of loci. Since it does not make much sense to calculate the index of association over that many loci, this function will randomly sample sites to calculate the index of association.

Usage

```
samp.ia(x, n.snp = 100L, reps = 100L, threads = 1L, quiet = FALSE)
```

Arguments

`x`	a [genlight][genlight-class] or [snpclone][snpclone-class] object.
`n.snp`	the number of snps to be used to calculate standardized index of association.
`reps`	the number of times to perform the calculation.
`threads`	The maximum number of parallel threads to be used within this function. A value of 0 (default) will attempt to use as many threads as there are available cores/CPUs. In most cases this is ideal. A value of 1 will force the function to run serially, which may increase stability on some systems. Other values may be specified, but should be used with caution.
`quiet`	if 'FALSE', a progress bar will be printed to the screen.

Details

The index of association is a summary of linkage disequilibrium among many loci. More information on the index of association can be found associated with the funciton [ia()]. A value near or at zero indicator of linkage equilibrium, whereas values significantly greater than zero indicate linkage disequilibrium. However, if the observed variance in distance among individuals is less than the expected, mildly negative values may be observed (as the range of this index is negative one to one). This function will call the function [bitwise.ia()] for 'reps' times to calculate the index of association over 'n.snp' loci. The standardized index of association ('rbarD') will be calculated 'reps' times. These esitmates of linkage disequilibrium from random genomic fractions can then be summarized (e.g., using a histogram) as an estimate of genome-wide linkage disequilibrium.

This function currently only works for objects of class genlight or snpclone that are of a single ploidy level and that ploidy is either haploid or diploid.

Value

Index of association representing the samples in this genlight object.

Note

this will calculate the standardized index of association from Agapow 2001. See [ia()] for details.

Author(s)

Zhian N. Kamvar, Jonah C. Brooks

See Also

[genlight][genlight-class], [snpclone][snpclone-class], [win.ia()], [ia()], [bitwise.dist()] [bitwise.ia()]

Examples

```
# with structured snps assuming 1e4 positions
set.seed(999)
x <- glSim(n.ind = 10, n.snp.nonstruc = 5e2,
           n.snp.struc = 5e2, ploidy = 2,
           parallel = FALSE)
position(x) <- sort(sample(1e4, 1e3))
res <- samp.ia(x)
hist(res, breaks = "fd")

# with unstructured snps assuming 1e4 positions
set.seed(999)
x <- glSim(n.ind = 10, n.snp.nonstruc = 1e3, ploidy = 2)
position(x) <- sort(sample(1e4, 1e3))
res <- samp.ia(x)
hist(res, breaks = "fd")
```

---

shufflepop — *Shuffle individuals in a genclone or genind object independently over each locus.*

Description

Shuffle individuals in a genclone or genind object independently over each locus.

Usage

```
shufflepop(pop, method = 1)
```

Arguments

pop	a genclone or genind object
method	an integer between 1 and 4. See details below.

Details

This function will shuffle each locus in the data set independently of one another, rendering them essentially unlinked. The following methods are available to shuffle your data:

1. **Permute Alleles** This will redistribute all alleles in the sample throughout the locus. Missing data is fixed in place. This maintains allelic structure, but heterozygosity is variable.

2. **Parametric Bootstrap** This will redistribute available alleles within the locus based on their allelic frequencies. This means that both the allelic state and heterozygosity will vary. The resulting data set will not have missing data.
3. **Non-Parametric Bootstrap** This will shuffle the allelic state for each individual. Missing data is fixed in place.
4. **Multilocus Style Permutation** This will shuffle the genotypes at each locus, maintaining the heterozygosity and allelic structure.

Value

a genclone or genind object shuffled by a specified method

Author(s)

Zhian N. Kamvar

References

Paul-Michael Agapow and Austin Burt. 2001. Indices of multilocus linkage disequilibrium. *Molecular Ecology Notes*, 1(1-2):101-102

Examples

```
# load the microbov dataset
data(microbov)
# Let's look at a single population for now. Howsabout Zebu
Zebu <- popsub(microbov, "Zebu")
summary(Zebu)

# Take note of the Number of alleles per population and the Observed
# heterozygosity as we go through each method.

# Permute Alleles: maintain allelic state; heterozygosity varies.
summary(shufflepop(Zebu, method=1))
## Not run:
# Parametric Bootstrap: do not maintain allelic state or heterozygosity
summary(shufflepop(Zebu, method=2))

# Non-Parametric Bootstrap: do not maintain allelic state or heterozygosity.
summary(shufflepop(Zebu, method=3))

# Multilocus Style: maintain allelic state and heterozygosity.
summary(shufflepop(Zebu, method=4))

## End(Not run)
```

test_replen *Test repeat length consistency.*

Description

This function will test for consistency in the sense that all alleles are able to be represented as discrete units after division and rounding.

Usage

```
test_replen(gid, replen)
```

Arguments

gid	a genind or genclone object
replen	a numeric vector of repeat motif lengths.

Details

This function is modified from the version used in doi:10.5281/zenodo.13007.

Value

a logical vector indicating whether or not the repeat motif length is consistent.

Author(s)

Zhian N. Kamvar

References

Zhian N. Kamvar, Meg M. Larsen, Alan M. Kanaskie, Everett M. Hansen, & Niklaus J. Grünwald. Sudden_Oak_Death_in_Oregon_Forests: Spatial and temporal population dynamics of the sudden oak death epidemic in Oregon Forests. ZENODO, doi:10.5281/zenodo.13007, 2014.

Kamvar, Z. N., Larsen, M. M., Kanaskie, A. M., Hansen, E. M., & Grünwald, N. J. (2015). Spatial and temporal analysis of populations of the sudden oak death pathogen in Oregon forests. Phytopathology 105:982-989. doi: doi:10.1094/PHYTO12140350FI

Ruzica Bruvo, Nicolaas K. Michiels, Thomas G. D'Souza, and Hinrich Schulenburg. A simple method for the calculation of microsatellite genotype distances irrespective of ploidy level. Molecular Ecology, 13(7):2101-2106, 2004.

See Also

fix_replen bruvo.dist bruvo.msn bruvo.boot

Examples

```
data(Pram)
(Pram_replen <- setNames(c(3, 2, 4, 4, 4), locNames(Pram)))
test_replen(Pram, Pram_replen)
```

---

upgma *UPGMA*

Description

UPGMA clustering. Just a wrapper function around hclust.

Usage

```
upgma(d)
```

Arguments

d	A distance matrix.

Value

A phylogenetic tree of class phylo.

Author(s)

Klaus Schliep <klaus.schliep@gmail.com>

See Also

hclust, as.phylo

Examples

```
library(ape)
data(woodmouse)
dm <- dist.dna(woodmouse)
tree <- upgma(dm)
plot(tree)
```

win.ia *Calculate windows of the index of association for genlight objects.*

Description

Genlight objects can contain millions of loci. Since it does not make much sense to calculate the index of association over that many loci, this function will scan windows across the loci positions and calculate the index of association.

Usage

```
win.ia(
  x,
  window = 100L,
  min.snps = 3L,
  threads = 1L,
  quiet = FALSE,
  name_window = TRUE,
  chromosome_buffer = TRUE
)
```

Arguments

x	a genlight or snpclone object.
window	an integer specifying the size of the window.
min.snps	an integer specifying the minimum number of snps allowed per window. If a window does not meet this criteria, the value will return as NA.
threads	The maximum number of parallel threads to be used within this function. Defaults to 1 thread, in which the function will run serially. A value of 0 will attempt to use as many threads as there are available cores/CPUs. In most cases this is ideal for speed. Note: this option is passed to bitwise.ia() and does not parallelize the windowization process.
quiet	if FALSE (default), a progress bar will be printed to the screen.
name_window	if TRUE (default), the result vector will be named with the terminal position of the window. In the case where several chromosomes are represented, the position will be appended using a period/full stop.
chromosome_buffer	*DEPRECATED* if TRUE (default), buffers will be placed between adjacent chromosomal positions to prevent windows from spanning two chromosomes.

Value

A value of the standardized index of association for all windows in each chromosome.

Note

this will calculate the standardized index of association from Agapow and Burt, 2001. See ia() for details.

Author(s)

Zhian N. Kamvar, Jonah C. Brooks

See Also

genlight, snpclone, ia(), samp.ia(), bitwise.dist()

Examples

```
# with structured snps assuming 1e4 positions
set.seed(999)
x <- glSim(n.ind = 10, n.snp.nonstruc = 5e2, n.snp.struc = 5e2, ploidy = 2)
position(x) <- sort(sample(1e4, 1e3))
res <- win.ia(x, window = 300L) # Calculate for windows of size 300
plot(res, type = "l")

## Not run:

# unstructured snps
set.seed(999)
x <- glSim(n.ind = 10, n.snp.nonstruc = 1e3, ploidy = 2)
position(x) <- sort(sample(1e4, 1e3))
res <- win.ia(x, window = 300L) # Calculate for windows of size 300
plot(res, type = "l")

# Accounting for chromosome coordinates
set.seed(999)
x <- glSim(n.ind = 10, n.snp.nonstruc = 5e2, n.snp.struc = 5e2, ploidy = 2)
position(x) <- as.vector(vapply(1:10, function(x) sort(sample(1e3, 100)), integer(100)))
chromosome(x) <- rep(1:10, each = 100)
res <- win.ia(x, window = 100L)
plot(res, type = "l")

# Converting chromosomal coordinates to tidy data
library("dplyr")
library("tidyr")
res_tidy <- res %>%
  tibble(rd = ., chromosome = names(.)) %>% # create two column data frame
  separate(chromosome, into = c("chromosome", "position")) %>% # get the position info
  mutate(position = as.integer(position)) %>% # force position as integers
  mutate(chromosome = factor(chromosome, unique(chromosome))) # force order chromosomes
res_tidy

# Plotting with ggplot2
library("ggplot2")
ggplot(res_tidy, aes(x = position, y = rd, color = chromosome)) +
  geom_line() +
```

```
  facet_wrap(~chromosome, nrow = 1) +
  ylab(expression(bar(r)[d])) +
  xlab("terminal position of sliding window") +
  labs(caption = "window size: 100bp") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1, vjust = 0.5)) +
  theme(legend.position = "top")


## End(Not run)
```

Index

∗ **amova**
poppr.amova, 98
∗ **angular**
nei.dist, 81
∗ **bootstrap**
aboot, 9
∗ **cluster**
upgma, 128
∗ **coancestry**
nei.dist, 81
∗ **datasets**
nei.dist, 81
∗ **edwards**
nei.dist, 81
∗ **missing**
info_table, 59
∗ **nei**
nei.dist, 81
∗ **ploidy**
info_table, 59
∗ **provesti**
nei.dist, 81
∗ **reynolds**
nei.dist, 81
∗ **rodgers**
nei.dist, 81
∗ **rogers**
nei.dist, 81

aboot, 9, *18*, *67*, *83*
aboot(), *5, 6*
ade4::amova(), *98, 99*, *101*
ade4::cailliez(), *99, 100*
ade4::is.euclid(), *101*
ade4::lingoes(), *99, 100*
ade4::quasieuclid(), *99, 100*
adegenet, *4*
adegenet::dist.genpop(), *11*
adegenet::genind, *4*, *45, 46*, *64*, *69*, *92, 93*, *109*, *117*
adegenet::genind(), *10*, *35*, *52*
adegenet::genlight, *4*, *9*, *69*
adegenet::genpop(), *10*
adegenet::import2genind(), *55*
adegenet::other(), *116*
adegenet::popNames(), *93*
adegenet::splitStrata(), *117*
adegenet::strata(), *93*, *100, 101*, *109*
adegenet::tab(), *93*
Aeut, 12
Aeut(), *8*
amova, *65*
amova *(*poppr.amova*)*, 98
ape::boot.phylo(), *10, 11*
ape::is.ultrametric(), *10*
ape::phylo(), *10*
as.genambig *(*bootgen2genind*)*, 17
as.genambig(), *5*, *66*, *101*
as.genambig,genind-method *(*bootgen2genind*)*, 17
as.genclone, *44*
as.genclone *(*bootgen2genind*)*, 17
as.genclone(), *5*
as.genclone,genind-method *(*bootgen2genind*)*, 17
as.phylo, *128*
as.snpclone, 13, *44*
as.snpclone,genlight-method *(*as.snpclone*)*, 13

bitwise.dist, 14, *32*, *41*, *44*, *70*, *73*
bitwise.dist(), *6*, *99*, *130*
bitwise.ia(), *129*
boot.ia, 16
boot.phylo, *19, 20*
boot::boot(), *33, 34*, *36, 37*
boot::boot.ci(), *36, 37*
boot::norm.ci(), *36, 37*
bootgen, *5*, *10, 11*
bootgen2genind, 17

bootgen2genind(), *5*, *11*
bootgen2genind,bootgen-method *(*bootgen2genind*)*, 17
bootstrap *(*aboot*)*, 9
bruvo.between *(*bruvo.dist*)*, 21
bruvo.boot, 19, *24*, *27*, *43*, *127*
bruvo.boot(), *6*, *10, 11*
bruvo.dist, *20*, 21, *27*, *43*, *57*, *74*, *127*
bruvo.dist(), *5, 6*
bruvo.msn, 24, *24*, *43*, *57*, *88*, *90*, *105*, *127*
bruvo.msn(), *7*
bruvomat, *5*

calculating allele frequencies from a round-robin approach, *114*
clonecorrect, 28
clonecorrect(), *5*, *46*, *55*, *93*, *96*, *98*, *101*, *117*
correcting rare alleles, *84, 85*, *110*, *120, 121*
cutoff_predictor, 30, *41*, *74*
cutoff_predictor(), *6*

delete_edges, *90*
diss.dist, 32, *41*, *57*, *73, 74*, *82, 83*
diss.dist(), *6*, *10, 11*, *15*, *101*
dist, *23*
dist(), *11*, *14*, *98*
dist.genpop, *81, 82*
diversity_boot, 33
diversity_boot(), *6*, *36, 37*, *39*
diversity_ci, 35
diversity_ci(), *6*, *34*, *39*, *94*, *96*
diversity_stats, 38, *70, 71*
diversity_stats(), *6*, *34*, *36, 37*, *93*, *96*

edwards.dist *(*nei.dist*)*, 81
edwards.dist(), *6*, *11*

file.choose(), *49*
file.copy(), *45*
file.path(), *49*
filter_stats, *31*, 40, *74*
filter_stats(), *6*
fix_replen, *21*, *23, 24*, 42, *127*
fix_replen(), *5, 6*
formula, *98*
formula(), *109*

gen-class, *5*
genambig, *5*
genclone, *4, 5*, *15*, *18, 19*, *21, 22*, *25*, *27*, *29*, *40*, *42*, *45–47*, *60, 61*, *63, 64*, *67*, *69*, *73, 74*, *77–79*, *81*, *87*, *92, 93*, *98*, *100*, *103, 104*, *106*, *109*, *117–119*, *125–127*
genclone *(*genclone-class*)*, 43
genclone(), *35*, *52*
genclone-class, *9*, 43, *116, 117*
genclone2genind *(*bootgen2genind*)*, 17
genclone2genind(), *5*
genclone2genind,genclone-method *(*bootgen2genind*)*, 17
genind, *5*, *15*, *18, 19*, *21, 22*, *25*, *29*, *32*, *40*, *42*, *44*, *47*, *60, 61*, *63*, *65*, *67*, *73*, *81, 82*, *87*, *98*, *100*, *103*, *106*, *118, 119*, *125–127*
genind-class, *9*, *116, 117*
genind2genalex, 45
genind2genalex(), *5*, *117*
genlight, *5*, *13–15*, *40*, *44*, *63*, *65*, *87*, *98*, *100*, *103*, *106*, *129, 130*
genotype_curve, 47
genotype_curve(), *7*
genpop, *118, 119*
genpop-class, *9*
getfile, 49
getfile(), *5*
ggplot2::ggplot, *70*
gray, *26*, *50*, *88*, *103*
greycurve, *26, 27*, 50, *88*, *90*, *103*, *105*
greycurve(), *7*

hclust, *128*
hist, *40*

ia, *32*, 51
ia(), *7*, *17*, *52*, *94*, *96*, *130*
imsn, 56
imsn(), *7*
incomp, 58
incomp(), *7*
info_table, 59
info_table(), *7*
informloci, 61
informloci(), *5*
is.clone *(*is.snpclone*)*, 62
is.genclone *(*is.snpclone*)*, 62
is.snpclone, 62

isPoly, *61*

jack.ia *(*ia*)*, 51
jack.ia(), *53*, *55*

layout.auto, *88*, *90*
legend, *90*
list.files(), *49*
loci, *47*, *48*
locus_table, 63
locus_table(), *7*

make_haplotypes, 65
make_haplotypes(), *5*, *100*, *101*
make_haplotypes,ANY-method *(*make_haplotypes*)*, 65
make_haplotypes,genclone-method *(*make_haplotypes*)*, 65
make_haplotypes,genind-method *(*make_haplotypes*)*, 65
make_haplotypes,genlight-method *(*make_haplotypes*)*, 65
make_haplotypes,snpclone-method *(*make_haplotypes*)*, 65
matrix, *9*
minimum.spanning.tree, *26*, *104*
missingno, *20*, *57*, 66, *73*, *105*
missingno(), *5*, *10*, *52*, *55*, *93*, *96*, *99*, *101*
MLG, *5*, *13*, *44*, *77*
mlg, 68
mlg(), *6*, *94*, *96*
mlg.crosspop(), *6*
mlg.filter, *26*, *31*, *40*, *41*, *44*, *48*, *70*, *71*, 72, *103*, *104*
mlg.filter(), *6*, *99*, *100*
mlg.filter,genclone-method *(*mlg.filter*)*, 72
mlg.filter,genind-method *(*mlg.filter*)*, 72
mlg.filter,genlight-method *(*mlg.filter*)*, 72
mlg.filter,snpclone-method *(*mlg.filter*)*, 72
mlg.filter<- *(*mlg.filter*)*, 72
mlg.filter<-,genclone-method *(*mlg.filter*)*, 72
mlg.filter<-,genind-method *(*mlg.filter*)*, 72
mlg.filter<-,genlight-method *(*mlg.filter*)*, 72
mlg.filter<-,snpclone-method *(*mlg.filter*)*, 72
mlg.id(), *6*
mlg.table, *77–79*, *106*
mlg.table(), *4*, *6*, *33*, *35*, *36*
mlg.vector(), *6*
mll, *44*, *70*, *71*, *74*, 76, *78*, *79*
mll(), *6*
mll,genclone-method *(*mll*)*, 76
mll,genind-method *(*mll*)*, 76
mll,genlight-method *(*mll*)*, 76
mll,snpclone-method *(*mll*)*, 76
mll.custom, *25*, *48*, *71*, *77*, 78, *79*, *88*, *103*
mll.custom(), *6*
mll.custom,genclone-method *(*mll.custom*)*, 78
mll.custom,snpclone-method *(*mll.custom*)*, 78
mll.custom<- *(*mll.custom*)*, 78
mll.custom<-,genclone-method *(*mll.custom*)*, 78
mll.custom<-,snpclone-method *(*mll.custom*)*, 78
mll.levels *(*mll.custom*)*, 78
mll.levels(), *6*
mll.levels,genclone-method *(*mll.custom*)*, 78
mll.levels,snpclone-method *(*mll.custom*)*, 78
mll.levels<- *(*mll.custom*)*, 78
mll.levels<-,genclone-method *(*mll.custom*)*, 78
mll.levels<-,snpclone-method *(*mll.custom*)*, 78
mll.reset, 79
mll.reset(), *6*
mll.reset,genclone-method *(*mll.reset*)*, 79
mll.reset,snpclone-method *(*mll.reset*)*, 79
mll<- *(*mll*)*, 76
mll<-,genclone-method *(*mll*)*, 76
mll<-,snpclone-method *(*mll*)*, 76
monpop, 80
monpop(), *8*
msn.bruvo *(*bruvo.msn*)*, 24

msn.poppr (poppr.msn), 102
mst, 27

nancycats, 20, 27, 105
nei.dist, 57, 67, 81
nei.dist(), 6, 9, 11
nj, 20, 105
nmll, 111
nmll (mll), 76
nmll(), 6
nmll,genclone-method (mll), 76
nmll,genind-method (mll), 76
nmll,genlight-method (mll), 76
nmll,snpclone-method (mll), 76
nodelabels, 20, 105

old2new_genclone, 83
old_partial_clone (partial_clone), 84
old_Pinf (Pinf), 86
other slot, 66

pair.ia (ia), 51
pair.ia(), 7, 17, 52, 54
palette, 90
partial_clone, 84
partial_clone(), 8
pegas::amova(), 66, 98, 101
pgen, 84, 111, 112, 114, 115, 121, 123
pgen(), 7
Pinf, 86
Pinf(), 8
plot.igraph, 89, 90
plot_poppr_msn, 26, 27, 57, 87, 104, 105
plot_poppr_msn(), 7
polysat::deSilvaFreq(), 101
poppr, 67, 92
poppr(), 7, 34, 37, 39, 55, 65
poppr-package, 3
poppr.all, 97
poppr.all(), 7, 49, 96
poppr.amova, 67, 83, 98
poppr.amova(), 7, 66
poppr.msn, 27, 57, 88, 90, 102
poppr.msn(), 7
poppr_has_parallel, 105
popsub, 57, 71, 106
popsub(), 5, 64
Pram, 107
Pram(), 8

prevosti.dist, 32
prevosti.dist (nei.dist), 81
prevosti.dist(), 6, 10, 11, 15
private_alleles, 108
private_alleles(), 7
provesti.dist, 32
provesti.dist (nei.dist), 81
psex, 52, 85, 110, 114, 115, 121, 123
psex(), 7, 17

rare_allele_correction, 7, 85, 112, 114, 121
read.genalex, 18, 116
read.genalex(), 5, 46, 55, 92
recode_polyploids, 118
recode_polyploids(), 5, 117
regex(), 49
resample.ia (ia), 51
resample.ia(), 6, 53, 55
reynolds.dist (nei.dist), 81
reynolds.dist(), 6, 11
rogers.dist, 73
rogers.dist (nei.dist), 81
rogers.dist(), 6, 11
round, 42
rraf, 84, 85, 110–112, 114, 115, 120, 123
rraf(), 7
rrmlg, 85, 112, 115, 120, 121, 122
rrmlg(), 7

samp.ia, 124
samp.ia(), 7, 15, 55, 130
sessionInfo, 57
setPop, 44
shufflepop, 125
shufflepop(), 5, 52, 93
snpclone, 4, 5, 9, 13–15, 29, 40, 63, 69, 73, 74, 77–79, 87, 98, 100, 103, 106, 129, 130
snpclone (genclone-class), 43
snpclone(), 35
snpclone-class (genclone-class), 43
splitStrata, 18
stats::rmultinom(), 34
strata, 44, 65, 84, 110, 120

tab, 20, 67, 105
test_replen, 24, 43, 127
test_replen(), 5

topo.colors, *25*, *103*

upgma, *19, 20*, *105*, 128

vegan::diversity(), *39*, *65*, *71*
vegan::rarefy(), *93*

win.ia, 129
win.ia(), *7*, *15*, *55*