Package 'GenoPop'

November 10, 2023

Title GenoPop

Version 0.0.0.9000

Description Package to calclulate population genomics stats from vcf files imported by vcfR.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

Imports vcfR,
methods,
Rcpp,
foreach,
doParallel,
parallel,
missForest

Suggests testthat,
withr

LazyData false

LinkingTo Rcpp

R topics documented:

`calculateAlleleFreqs` *calculateAlleleFreqs*

Description

Calculate allele frequencies from a vcfR object. Will also add the slots ploidy and sep_gt, if not already present.

Usage

```
calculateAlleleFreqs(object, missing_data = "none", ...)
```

Arguments

`object`	A S4 object of class vcfR.
`missing_data`	Method to deal with missing data. Options are "remove", "impute", "none". Default is "none". In case of "remove", function needs the additional parameter threshold, which is the fraction of missing data in a variant that is still acceptable. In case of "impute" the function needs the additional parameters "method", with which the imputation method can be chosen. See `imputeMissingData`
`...`	Additional parameters for how to deal with missing data. For imputation see `imputeMissingData` and for removal see `rmMissingData`.

Value

A S4 object of the same class but with following slots added:

- allele_freqs (data frame)

Examples

```
data("real", package = "GenoPop")
vcf <- calculateAlleleFreqs(real, missing_data = "none")
vcf <- calculateAlleleFreqs(real, missing_data = "remove", threshold = 0.1)
vcf <- calculateAlleleFreqs(real, missing_data = "impute", method = "mean")
head(vcf@allele_freqs)
```

`calculatePloidyAndSepGT` *calculatePloidyAndSepGT*

Description

Calculate ploidy levels and separate the genotypes into a matrix according to the ploidy.

Usage

```
calculatePloidyAndSepGT(object)
```

Arguments

`object`	A S4 object of class vcfR.

Value

A S4 object of the same class but with following slots added:

- ploidy (integer)
- sep_gt (matrix)

Examples

```
data("real", package = "GenoPop")
vcf <- calculatePloidyAndSepGT(real)
vcf@ploidy
head(vcf@sep_gt)
```

calculateWindowedMetric *calculateWindowedMetric*

Description

Calculate one of the population genomics metrics of this package on a per window basis over a longer sequence or even whole chromsomes and genomes. Calculations are done in parallel.

Usage

```
calculateWindowedMetric(
  object,
  metricFunction,
  window_size = 1000,
  step_size = 0,
  min_var = 2,
  pop_assignments = NULL,
  write_log = FALSE,
  logfile = "logfile.txt"
)
```

Arguments

object	An S4 object of type GPvcfR.
metricFunction	One of the population genomics metrics functions included in this package.
window_size	The size of the window for which Pi is calculated. (Default = 1000)
step_size	The size of the step in between windows. (Default = 0)
min_var	Minimum number of variants that must be present in a window to calculate the metric. Default is set to 2, because many metrics break if there is only one or none variant to work with.
pop_assignments	If the metric is calculated from two populations (f.e. Fst, private alleles, etc.) then on has to provide a named vector. Elements are the population names and names are the individual name.
write_log	Logical, whether a log file of the process should be written to disk. This is adviced for imputing large data sets.
logfile	Name of the log file, if write_log is true.

Value

A data frame with four columns, the window chromosome, the window start and end postion, the number of variants in the window, and the value of the metric.

Examples

```
data("mys", package = "GenoPop")
calculateWindowedMetric(mys, Pi, window_size = 10000)
```

---

dav *The test data set called 'dav'*

Description

This is the real data set, but separated by population (mys & dav) with missing data imputed (rf) and allele frequencies calculated.

Usage

```
data("dav", package = "GenoPop")
```

Format

A GPvcfR object

Examples

```
## Not run:
data("dav", package = "GenoPop")
head(dav@imp_gt)

## End(Not run)
```

Dxy *Dxy*

Description

Calculate the average number of nucleotide differences per site (Dxy) between two populations. Nei & Li, 1979 (https://doi.org/10.1073/pnas.76.10.52699) Handling missing alleles at one site is equivalent to Korunes & Samuk, 2021 (https://doi.org/10.1111/1755-0998.13326), but for simplicity assuming that completely missing sites are invariant sites, which will underestimate Dxy. Otherwise this would only function with VCF files that include all monomorphic sites, which may be unpractical given common data sets. If you happen to know the number of missing sites vs the number of monomorphic sites, please use the number of monomorphic + the number of polymorphic sites as the sequence length to the the most accurate estimation of Dxy.

Usage

```
Dxy(object, pop_assignments, seq_length)
```

Arguments

object	A GPvcfR object.
pop_assignments	A named vector with elements being the population names and names being the individual names.
seq_length	Length of the sequence in number of bases, including monomorphic sites.

Value

The average number of nucleotide substitutions per site (Dxy).

Examples

```
data("mys", package = "GenoPop")
mys1 <- c("8449", "8128", "8779")
mys2 <- c("8816", "8823", "8157")
individuals <- c(mys1, mys2)
pop_names <- c(rep("pop1", length(mys1)), rep("pop2", length(mys2)))
pop_assignments <- setNames(pop_names, individuals)
Dxy(mys, pop_assignments, 265392)
```

ExpectedHeterozygosity

Expected Heterozygosity

Description

This function calculates the expected heterozygosity of a population.

Usage

```
ExpectedHeterozygosity(object)
```

Arguments

object	An S4 object of type GPvcfR. Allele frequencies must be present.

Value

Expected heterozygosity.

Examples

```
data("mys", package = "GenoPop")
ExpectedHeterozygosity(mys)
```

filterBiallelicSNPs *filterBiallelicSNPs*

Description

Filter for only biallelic SNPs in the data set.

Usage

```
filterBiallelicSNPs(object)
```

Arguments

object	A S4 object of class vcfR.

Value

A S4 object of the same class, but complex and multiallelic variantes are removed from the @fix and @gt slots.

Examples

```
data("real", package = "GenoPop")
vcf <- filterBiallelicSNPs(real)
```

FixedSites *FixedSites*

Description

Count the number of sites fixed for an alternative allele.

Usage

```
FixedSites(object)
```

Arguments

object	An S4 object of type GPvcfR. Allele frequencies must be present.

Value

The number of fixed sites.

Examples

```
data("real", package = "GenoPop")
vcf <- calculateAlleleFreqs(real, missing_data = "impute", method = "mean")
FixedSites(vcf)
```

`Fst` *Fst*

Description

Calculate the mean or weighted (by number of non missing chromsomes) fixiation index (Fst) from two populations in a list of GPvcfR objects using the method of Weir and Cockerham (1984).

Usage

```
Fst(object, pop_assignments, weighted = FALSE)
```

Arguments

`object`	A GPvcfR object.
`pop_assignments`	A named vector. Elements are the population names and names are the individual name.
`weighted`	Logical, wether weighted Fst or mean Fst is returned. (Default = FALSE (mean Fst is returned))

Value

Fst value.

Examples

```
mys1 <- c("8449", "8128", "8779")
mys2 <- c("8816", "8823", "8157")

individuals <- c(mys1, mys2)
pop_names <- c(rep("mys1", length(mys1)), rep("mys2", length(mys2)))
pop_assignments <- setNames(pop_names, individuals)

data("mys", package = "GenoPop")
Fst(mys, pop_assignments)
```

GenoPop *GenoPop*

Description

A R package to perform several population genomics analyses directly on whole genome vcf files.

Details

Most important features are:

- Reading genotype data ready to use from vcf files using the vcfR package.
- Different methods to impute or remove missing data from the genotype matrix.
- Calculation of several commonly used population genomics metrics from the genotype data.
- Window based analyses with those different metrics.
- Parallelization and optimization of heavy tasks to enable processing of whole genomes (imputation and window based analyses).
- Writing processed data back to file in vcf format.

Installation Instructions for GenoPop:

Prerequisites:
Before installing GenoPop, make sure you have R installed on your system. You can download and install R from CRAN.

Installing GenoPop from GitHub:
To install the GenoPop package directly from GitHub, you will need the `devtools` package in R. If you don't have `devtools` installed, you can install it by running the following command in R:

```
install.packages("devtools")
```

Once devtools is installed, you can install GenoPop using the install_github function. Run the following commands in your R console:

```
library(devtools)
install_github("https://github.com/MiGurke/GenoPop")
```

Then load the package..

```
library(GenoPop)
```

and your ready to go!

Dependencies:
For proper compression of vcf's newly generated, you need to have tabix installed on your machine. All other dependencies are supposed to be handled by R itself.

Getting started:

To get your vcf formated data into R use the reading function of the vcfR package:

```
vcf <- read.vcfR( "example.vcf", verbose = FALSE )
```

It can read compressed and umcompressed vcf files.

From there a good point to start your analysis is to use the calculateAlleleFreqs function to just populate the most important data and information slots from you vcf file automatically.

```
example <- calculateAlleleFreqs(vcf)
```

This will give you a nicely formatted genotype matrix (example@sep_gt), calculated allele frequencies (example@allele_freqs) and some interesting stats about the amount of missing data in you data set (example@missing_data).

From there it depends on you how to continue. You can fix some issue with missing data by imputing or removing it, directly calculate some stats like Fst, Pi, and the site frequency spectrum, or even do a window based analysis. Just have a look at the available functions in the man pages.

If you need further assistance or got suggestions for this package, feel free to open an issue on the GenoPop GitHub repo or contact me in any other way.

Author(s)

Maintainer: Marie (Mick) Gurke <margurke@gmail.com> (ORCID)

`imputeMissingData` *imputeMissingData*

Description

Impute missing variants in genotype data stored in vcfR object.

Usage

```
imputeMissingData(object, method = "mean", ...)
```

Arguments

`object`	A S4 object of class vcfR.
`method`	Method used for missing data imputation. Available are "kNN", "rf", and "mean". (Default = "mean").
`...`	Additional parameters for different imputation methods. For more info look up the documentation of them:`meanImputation`, `kNNImputation`, `rfImputation`.

Value

A S4 object of the same class, but the slot imp_gt is now filled with the imputed genotype matrix.

Examples

```
data("real", package = "GenoPop")
vcf <- imputeMissingData(real, method = "mean")
```

kNNImputation *kNNImputation*

Description

kNNImputation

Usage

```
kNNImputation(
  sep_gt,
  k = 3,
  chunk_size = 1000,
  write_log = FALSE,
  logfile = "logfile.txt"
)
```

Arguments

sep_gt	A separated genotype matrix from a myvcfR object.
k	Number of nearest neighbours used for imputation, default: 3.
chunk_size	Number of variants analyzed in on batch in the parallelization. Default: 1000. Increasing this might improve accuracy, but will substantially increase running time.
write_log	Logical, whether a log file of the process should be written to disk. This is adviced for imputing large data sets.
logfile	Name of the log file, if write_log is true.

Value

A separated genotype matrix from a myvcfR object, but with imputed missing values.

Examples

```
example_matrix <- matrix(c("0", "1", ".", "1", "1", ".", "0", "0", ".", "1", "1", ".", "0", "0", "0", "0", ".", "1",
kNNImputation(example_matrix, k = 3, chunk_size = 1000)
```

knn_imputeR *knn_imputeR*

Description

This is just the wrapper function for the kNN imputation algorithm that is written in C++. Please use the function kNNImputation().

Usage

```
knn_imputeR(data, k)
```

Arguments

data	NumericMatrix, the data matrix.
k	Integer, the number of neighbors.

Value

NumericMatrix, the imputed data matrix.

Examples

```
example_matrix <- matrix(c(0, 1, NA, 1, 1, NA, 0, 0, NA, 1, 1, NA, 0, 0, 0, 0, NA, 1, 0, 0, NA, 1, 1, NA, 0), nrow = 5,
knn_imputeR(example_matrix, k = 3)
```

meanImputation *meanImputation*

Description

Use the mean over each variant or individual to replace missing data with that mean. Mean is rounded to keep genotype integers, so this just corresponds to the most often occuring genotyp.

Usage

```
meanImputation(sep_gt, mode = "variant")
```

Arguments

sep_gt	separated genotype matrix from myvcfR object.
mode	Means are calculated either yb "variant" or by "individual".

Value

Matrix with imputed missing data.

Examples

```
example_matrix <- matrix(c("0", "1", ".", "1", "1", ".", "0", "0", ".", "1", "1", ".", "0", "0", "0", "0", ".", "1",
meanImputation(example_matrix, mode ="variant")
meanImputation(example_matrix, mode ="individual")
```

`mys` *The test data set called 'mys'*

Description

This is the real data set, but separated by population (mys & dav) with missing data imputed (rf) and allele frequencies calculated.

Usage

```
data("mys", package = "GenoPop")
```

Format

A GPvcfR object

Examples

```
## Not run:
data("mys", package = "GenoPop")
head(mys@imp_gt)

## End(Not run)
```

`ObservedHeterozygosity` *Observed Heterozygosity*

Description

This function calculates the observed heterozygosity in a population.

Usage

```
ObservedHeterozygosity(object)
```

Arguments

`object`	An S4 object of type GPvcfR.

Value

Observed heterozygosity.

Examples

```
data("mys", package = "GenoPop")
ObservedHeterozygosity(mys)
```

`OneDimSFS` *OneDimSFS*

Description

Calculate a one dimensional site frequency spectrum from an GPvcfR object.

Usage

```
OneDimSFS(object, folded = FALSE)
```

Arguments

`object`	A S4 object of type GPvcfR. Allele frequencies and genotype matrix must be present.
`folded`	Logical, deciding if folded (TRUE) or unfolded (FALSE) SFS is returned. For the unfolded it is assumed that the genotype "0" represents the ancestral state in the data. (Default is unfolded (FALSE).)

Value

Site frequency spectrum as a named vector

Examples

```
data("mys", package = "GenoPop")
OneDimSFS(mys, folded = FALSE)
```

`Pi` *Pi*

Description

Calculate the average number of nucleotide differences per site between two sequences. Nei & Li, 1979 (https://doi.org/10.1073/pnas.76.10.5269) The formula used for this is equivalent to the one used in vcftools –window-pi (https://vcftools.sourceforge.net/man_latest.html). Handling missing alleles at one site is equivalent to Korunes & Samuk, 2021 (https://doi.org/10.1111/1755-0998.13326), but for simplicity assuming that completely missing sites are invariant sites, which will underestimate Pi. Otherwise this would only function with VCF files that include all monomorphic sites, which may be unpractical given common data sets. If you happen to know the number of missing sites vs the number of monomorphic sites, please use the number of monomorphic + the number of polymorphic sites as the sequence length to the the most accurate estimation of Pi.

Usage

```
Pi(object, seq_length)
```

Arguments

`object`	An S4 object of type GPvcfR.
`seq_length`	Length of the sequence in number of bases. Must be provided to accurately work with all monomorphic sites, including those monomorphic for the reference, which are generally not included in a vcf.

Value

The nucleotide diversity (pi) per window in data frame.

Examples

```
data("mys", package = "GenoPop")
Pi(mys, 265392)
```

`PolymorphicSites` *PolymorphicSites*

Description

Count the number of polymorphic sites in the data set (aka. sites not fixed for the alternative allele).

Usage

```
PolymorphicSites(object)
```

Arguments

object	An S4 object of type GPvcfR. Allele frequencies must be present.

Value

The number of polymorphic sites.

Examples

```
data("real", package = "GenoPop")
vcf <- calculateAlleleFreqs(real, missing_data = "impute", method = "mean")
PolymorphicSites(vcf)
```

PrivateAlleles *PrivateAlleles*

Description

Function to calculate the number of private alleles in two populations.

Usage

```
PrivateAlleles(object, pop_assignments)
```

Arguments

object	A GPvcfR object.
pop_assignments	A named vector. Elements are the population names and names are the individual name.

Value

A list containing the number of private alleles for each population.

Examples

```
mys1 <- c("8449", "8128", "8779")
mys2 <- c("8816", "8823", "8157")

individuals <- c(mys1, mys2)
pop_names <- c(rep("mys1", length(mys1)), rep("mys2", length(mys2)))
pop_assignments <- setNames(pop_names, individuals)

data("mys", package = "GenoPop")
PrivateAlleles(mys, pop_assignments)
```

real *The test data set called 'real'*

Description

These are first couple of thousand lines of a vcf file from real world data. It contains genotype information of 16 individuals of two bat species.

Usage

```
data("real", package = "GenoPop")
```

Format

A vcfR object.

Examples

```
## Not run:
data("real", package = "GenoPop")
head(vcf@gt)

## End(Not run)
```

rfImputation *rfImputation*

Description

rfImputation

Usage

```
rfImputation(
  sep_gt,
  maxiter = 10,
  ntree = 100,
  chunk_size = 1000,
  write_log = FALSE,
  logfile = "logfile.txt"
)
```

Arguments

sep_gt	A separated genotype matrix from a myvcfR object.
maxiter	The number of improvement iterations the random forest algorithm (missForest) runs.
ntree	The number of decision trees in the random forest.
chunk_size	Number of variants analyzed in on batch in the parallelization. Default: 1000. Increasing this might improve accuracy, but will substantially increase running time.
write_log	Logical, whether a log file of the process should be written to disk. This is adviced for imputing large data sets.
logfile	Name of the log file, if write_log is true.

Value

A separated genotype matrix from a myvcfR object, but with imputed missing values.

Examples

```
example_matrix <- matrix(c("0", "1", ".", "1", "1", ".", "0", "0", ".", "1", "1", ".", "0", "0", "0", "0", ".", "1",
rfImputation(example_matrix, maxiter = 10, ntree = 100, chunk_size = 1000)
```

---

rmMissingData *rmMissingData*

Description

Remove variants from vcfR object with to much missing data.

Usage

```
rmMissingData(object, threshold = 0.1)
```

Arguments

object	A S4 object of class vcfR.
threshold	Fraction of missing individuals per variant that is still accepted. Default: 0.1

Value

A S4 object of the same class, but without variants that did not meet the missingness threshold. In addition, the slot "missing_data" will be added to the object. It contains a list with information about the removed variants and the missingness per variant and individual.

Examples

```
data("real", package = "GenoPop")
vcf <- rmMissingData(real, 0.1)
```

SegregatingSites *SegregatingSites*

Description

Count the number of segregating sites in the data set.

Usage

```
SegregatingSites(object)
```

Arguments

object	An S4 object of type GPvcfR. Allele frequencies must be present.

Value

The number of segregating sites.

Examples

```
data("real", package = "GenoPop")
vcf <- calculateAlleleFreqs(real, missing_data = "impute", method = "mean")
SegregatingSites(vcf)
```

---

separateByPopulations *separateByPopulations*

Description

separates a vcfR object into new objects per population. Needs to be done prior to calculating allele frequencies.

Usage

```
separateByPopulations(object, pop_assignments, rm_ref_alleles = TRUE)
```

Arguments

object	A S4 object of class vcfR or GPvcfR.
pop_assignments	A named vector. Elements are the population names and names are the individual name.
rm_ref_alleles	Logical, wether variants that only have the reference allele should be removed from the respective subpopulations object. (Default = TRUE)

Value

A list containing one vcfR object per population.

Examples

```
mys <- c("8449", "8128", "8779", "8816", "8823", "8157")
dav <- c("8213", "8241", "8232", "8224", "10165", "8221", "8813", "8825", "8182", "8187")

individuals <- c(mys, dav)
pop_names <- c(rep("mys", length(mys)), rep("dav", length(dav)))
pop_assignments <- setNames(pop_names, individuals)

data("real", package = "GenoPop")
vcfs <- separateByPopulations(real, pop_assignments)
```

---

sim *The test data set called 'sim'*

Description

These are first couple of thousand lines of a vcf file from a simulated data set that was created by msprime. It also contains genotype information of 16 individuals.

Usage

```
data("sim", package = "GenoPop")
```

Format

A vcfR object.

Examples

```
## Not run:
data("sim", package = "GenoPop")
head(vcf@gt)

## End(Not run)
```

SingeltonSites *SingeltonSites*

Description

Count the number of singelton sites in the data set. These are sites where a minor allele occurs only once in the sample.

Usage

```
SingeltonSites(object)
```

Arguments

object	An S4 object of type GPvcfR. Allele frequencies must be present.

Value

The number of singelton sites.

Examples

```
data("real", package = "GenoPop")
vcf <- calculateAlleleFreqs(real, missing_data = "impute", method = "mean")
SingeltonSites(vcf)
```

TajimasD *Tajima's D*

Description

Calculate Tajima's D statistic for a given dataset, a measure for neutrality.

Usage

```
TajimasD(object, seq_length)
```

Arguments

object	A S4 object of type GPvcfR. Allele frequencies and genotype matrix must be present.
seq_length	Length of the sequence in number of bases. Must be provided to accurately work with all monomorphic sites, including those monomorphic for the reference, which are generally not included in a vcf.

Value

Tajima's D value.

Examples

```
data("mys", package = "GenoPop")
TajimasD(mys, 265392)
```

TwoDimSFS *TwoDimSFS*

Description

Calculate a two-dimensional site frequency spectrum from a list of two GPvcfR objects.

Usage

```
TwoDimSFS(object, pop_assignments, folded = FALSE)
```

Arguments

object	A GPvcfR object.
pop_assignments	A named vector. Elements are the population names and names are the individual name.
folded	Logical, deciding if folded (TRUE) or unfolded (FALSE) SFS is returned. (Default is unfolded (FALSE).)

Value

Two-dimensional site frequency spectrum as a matrix

Examples

```
mys1 <- c("8449", "8128", "8779")
mys2 <- c("8816", "8823", "8157")

individuals <- c(mys1, mys2)
pop_names <- c(rep("mys1", length(mys1)), rep("mys2", length(mys2)))
pop_assignments <- setNames(pop_names, individuals)

data("mys", package = "GenoPop")

TwoDimSFS(mys, pop_assignments, folded = TRUE)
```

WattersonsTheta *WattersonsTheta*

Description

Calculate Watterson's thea, a measure for neutrality, from an GPvcfR object. The metric will be normalized by the sequence length to make it comparable between data sets.

Usage

```
WattersonsTheta(object, seq_length)
```

Arguments

object	A S4 object of type GPvcfR. Allele frequencies and genotype matrix must be present.
seq_length	The length of the sequence in the data set.

Value

Watterson's theta value.

Examples

```
data("mys", package = "GenoPop")
WattersonsTheta(mys, 265392)
```

writeVCF *writeVCF*

Description

Writes a new VCF file to disk using the imputed or separated genotypes from a GPvcfR object.

Usage

```
writeVCF(object, file_path, use_imputed = TRUE, bgzip = FALSE)
```

Arguments

object	A GPvcfR object containing the VCF data.
file_path	Path to the output VCF file.
use_imputed	Logical, indicating whether to use the imputed genotypes if available. If FALSE or imputed genotypes are not available, separated genotypes will be used.
bgzip	Logical, indicating whether to bgzip the output file (requires tabix to be installed).

Value

Invisible NULL. The function is called for its side effect of writing a file.

Examples

```
data("mys", package = "GenoPop")
writeVcf(mys, "output.vcf", use_imputed = TRUE)
```

Index