Package 'EFGLmh'

January 28, 2022

Type Package

Title Functions For Working With Microhaps for EFGL

Version 0.1.0

Description Written to work with microhaps and SNPs (which are just short microhaps).
More generally, will function with codominant, diploid genotypes.
Uses ``Progeny-style" (and FishGen-style) inputs.
Performs basic manipulations, data summaries, and exporting data in
formats for other packages/programs.

Imports tibble,
dplyr,
readr,
tidyr

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.1.2

Depends R (>= 2.10)

Suggests knitr,
rmarkdown,
tidyverse,
rubias

VignetteBuilder knitr

R topics documented:

aRich . . . 2
calcAF . . . 3
calcHet . . . 3
cleanGrandma . . . 4
combineEFGLdata . . . 4
construct_EFGLdata . . . 5
createF1Hybrids . . . 5
dumpTable . . . 6
exampleData . . . 6
exportCKMRsimAF . . . 7
exportCKMRsimLG . . . 7

exportGenAlEx . . . 8
exportGenePop . . . 8
exportGrandma . . . 9
exportHierFstat . . . 10
exportPlink . . . 10
exportProgenyStyle . . . 11
exportRubias_baseline . . . 12
exportRubias_mixture . . . 12
exportSNPPIT . . . 13
exportStructure . . . 14
genoSuccess . . . 15
getInds . . . 15
getLoci . . . 16
getMeta . . . 16
getPops . . . 16
lociSuccess . . . 17
moveInds . . . 17
movePops . . . 18
numInds . . . 18
print.EFGLdata . . . 19
readInData . . . 19
removeInds . . . 20
removeLoci . . . 21
removePops . . . 21
whichLower . . . 22

Index **23**

`aRich` *calculate allelic richness of loci*

Description

calculate allelic richness of loci

Usage

```
aRich(x)
```

Arguments

`x`	an EFGLdata object

Value

a tibble giving the allelic richness of each locus

`calcAF` *calculate allele frequencies within populations Loci that are all missing genotypes wihtin a pop will be (silently) excluded form the output.*

Description

calculate allele frequencies within populations Loci that are all missing genotypes wihtin a pop will be (silently) excluded form the output.

Usage

```
calcAF(x)
```

Arguments

x	an EFGLdata object

Value

a tibble

`calcHet` *calculate expected and observed heterozygosity within populations*

Description

calculate expected and observed heterozygosity within populations

Usage

```
calcHet(x)
```

Arguments

x	an EFGLdata object

Value

a tibble

cleanGrandma *convenience function to remove loci with all fails or no variation from gRandma input*

Description

convenience function to remove loci with all fails or no variation from gRandma input

Usage

```
cleanGrandma(baseline, mixture = NULL)
```

Arguments

baseline	a gRandma baseline input
mixture	a gRandma mixture input

Value

a list with two components, one is the baseline, one is the mixture

combineEFGLdata *combine multiple EFGL objects into one*

Description

combine multiple EFGL objects into one

Usage

```
combineEFGLdata(
  ...,
  genoComb = c("intersect", "union"),
  metaComb = c("intersect", "union")
)
```

Arguments

...	multiple EFGLdata objects separated by commas
genoComb	if the objects have different loci, whether to create a new object with the intersection or union of loci. If union, genotypes for missing loci are all NA.
metaComb	if the objects have different metadata fields, whether to create a new object with the intersection or union of the fields. If union, missing fields are all NA.

construct_EFGLdata *some basic checks on EFGLdata objects*

Description

some basic checks on EFGLdata objects

Usage

```
construct_EFGLdata(x)
```

Arguments

x	an EFGLdata object

createF1Hybrids *Create F1 hybrids from two populations*

Description

Creates hybrids by sampling one allele from one pop and the other allele from the other pop based on observed allele frequencies. Missing genotypes are sampled based on the combined rate of missing genotypes or data can be simulated wiht no missing genotypes. Alleles at different loci are sampled independently. Note that if a sex marker exists in the data set, it is treated like any other marker, resulting in XX, XY, and YY genotypes in the hybrids. You may want to remove it prior to simulation using the 'removeLoci' function. Loci with all missing genotypes in one population are simulated to have all missing genotypes in the hybrids REGARDLESS of the input value for 'missingGenos'.

Usage

```
createF1Hybrids(x, pop1, pop2, newName, n = 50, missingGenos = TRUE)
```

Arguments

x	an EFGLdata object
pop1	one parent population (must be in 'x')
pop2	the other parent population (must be in 'x')
newName	a string giving the name of the population to put hybrids into. This MUST be a new pop.
n	the number of hybrids to simulate
missingGenos	TRUE to simulate missing genotypes, FALSE to not

Value

an EFGLdata object with all the populations in 'x', plus the new simulated hybrid population

dumpTable *wrapper for write table with commonly used options - carried over from IDFGEN*

Description

wrapper for write table with commonly used options - carried over from IDFGEN

Usage

```
dumpTable(x, filename, row.names = FALSE, sep = "\t")
```

Arguments

x	object to write out
filename	filename to write out as
row.names	passed to write.table
sep	passed to write.table

Value

nothing, just writes a file

exampleData *An example input dataset used in the vignette*

Description

An example input dataset used in the vignette

Usage

```
exampleData
```

Format

a tibble

exportCKMRsimAF *return a CKMRsim allele frequency input tibble.*

Description

No chromosome and position information is used. All populations specified are combined into one allele frequency output. Any loci with only missing genotypes are omitted from the output. The output should then be run through the CKMRsim function `reindex_markers()`.

Usage

```
exportCKMRsimAF(x, pops = NULL, loci = NULL)
```

Arguments

x	an EFGLdata object
pops	a vector of pops to include. If not specified, all pops are used.
loci	a vector of loci to include. If not specified, all loci are used.

Value

a tibble

exportCKMRsimLG *Return a long genotype tibble for input to CKMRsim for relationship inference.*

Description

Return a long genotype tibble for input to CKMRsim for relationship inference.

Usage

```
exportCKMRsimLG(x, pops = NULL, loci = NULL)
```

Arguments

x	an EFGLdata object
pops	a vector of pops to include. If not specified, all pops are used.
loci	a vector of loci to include. If not specified, all loci are used.

Value

a tibble

exportGenAlEx *write a GenAlEx input file*

Description

write a GenAlEx input file

Usage

```
exportGenAlEx(
  x,
  filename,
  pops = NULL,
  loci = NULL,
  title = "",
  useNames = TRUE
)
```

Arguments

x	an EFGLdata object
filename	the name of the file to write
pops	a vector of pops to include. If not specified, all pops are used.
loci	a vector of loci to include. If not specified, all loci are used.
title	a string to use as the "title" row
useNames	TRUE to use samples names, FALSE to replace with unique numerical identifiers

Value

nothing, just writes a file

exportGenePop *write a genepop input file*

Description

write a genepop input file

Usage

```
exportGenePop(
  x,
  filename,
  header = "genePop file",
  pops = NULL,
  loci = NULL,
  useIndNames = FALSE
)
```

Arguments

x	an EFGLdata object
filename	the name of the file to write
header	a string to use as the header line of the genepop file
pops	a vector of pops to include. If not specified, all pops are used.
loci	a vector of loci to include. If not specified, all loci are used.
useIndNames	TRUE to use individual names as sample identifiers. Otherwise, population names are used

Value

nothing, just writes a file

---

exportGrandma *export a gRandma baseline or mixture*

Description

export a gRandma baseline or mixture

Usage

```
exportGrandma(x, pops = NULL, loci = NULL, baseline = TRUE)
```

Arguments

x	an EFGLdata object
pops	a vector of pops to include in the baseline. If not specified, all pops are used.
loci	a vector of loci to use. If not specified, all loci are used.
baseline	TRUE to make a baseline input, FALSE to make a mixture input.

Value

a tibble

exportHierFstat *export a hierfstat input dataframe*

Description

export a hierfstat input dataframe

Usage

```
exportHierFstat(x, pops = NULL, loci = NULL)
```

Arguments

x	an EFGLdata object
pops	a vector of pops to include. If not specified, all pops are used.
loci	a vector of loci to use. If not specified, all loci are used.

Value

a dataframe coded to be used as input for hierfstat

exportPlink *export a Plink PED file and optionally a (not very useful except as a template) MAP file*

Description

Only biallelic markers are used, exports as two column per call. This format can be used as the input for "Admixture".

Usage

```
exportPlink(
  x,
  filename,
  pops = NULL,
  loci = NULL,
  FID = "Ind",
  IID = "Ind",
  pa = NULL,
  ma = NULL,
  sex = NULL,
  pheno = NULL,
  map = NULL
)
```

Arguments

x	an EFGLdata object
filename	the name of the file to write
pops	a vector of pops to include. If not specified, all pops are used.
loci	a vector of loci to use. If not specified, all loci are used.
FID	Metadata column name to use as the family ID
IID	Metadata column name to use as the within-family ID
pa	Metadata column name to use as the within-family ID of the father. If NULL, all are listed as unknown (0).
ma	Metadata column name to use as the within-family ID of the mother. If NULL, all are listed as unknown (0).
sex	Metadata column to use as the sex. This should be coded as "M" for male, "F" for female, and any other values are treated as Unknown. If NULL, all are listed as unknown (0).
pheno	Metadata column to use as the phenotype. This should be coded as "1" for control, "2" for case, and "-9" or "0" for Unknown. If NULL, all are listed as unknown (0).
map	filename to write a MAP file. If NULL, no MAP file is written. This just writes a dummy MAP file with the loci names in order and all given the same chromosome code and positions of "0". If you need a valid MAP file, you will need to edit this.

Value

nothing, just writes a file

---

exportProgenyStyle *export a "Progeny-style" export file for later reading into EFGLmh*

Description

Columns in order are Pop, Ind, metadata, genotypes (2-column per call) Missing genotypes are "0" for SNPs (biallelic or nonvariable with alleles represented by 1 character) and "000" for others. If a locus is all missing, it is treated as a SNP. Pop column is called "Pedigree" and Ind column is called "Individual.Name".

Usage

```
exportProgenyStyle(x, filename, pops = NULL, loci = NULL, metadata = NULL)
```

Arguments

x	an EFGLdata object
filename	the name of the file to write
pops	a vector of pops to include. If not specified, all pops are used.
loci	a vector of loci to use. If not specified, all loci are used.
metadata	a vector of metadata fields to include. If not specified, all fields are used.

Value

nothing, just writes a file

`exportRubias_baseline` *export a rubias baseline*

Description

export a rubias baseline

Usage

```
exportRubias_baseline(
  x,
  pops = NULL,
  repunit = NULL,
  collection = NULL,
  loci = NULL
)
```

Arguments

`x`	an EFGLdata object
`pops`	a vector of pops to include in the baseline. If not specified, all pops are used.
`repunit`	the column name of the metadata variable designating repunit. This can be Pop to use the population name. If not specified, NA is used for all samples.
`collection`	the column name of the metadata variable designating collection. This can be Pop to use the population name. If not specified, NA is used for all samples.
`loci`	a vector of loci to use. If not specified, all loci are used.

Value

a tibble

`exportRubias_mixture` *export a rubias mixture*

Description

export a rubias mixture

Usage

```
exportRubias_mixture(x, pops = NULL, collection = NULL, loci = NULL)
```

Arguments

x	an EFGLdata object
pops	a vector of pops to include in the baseline. If not specified, all pops are used.
collection	the column name of the metadata variable designating collection. This can be Pop to use the population name. If not specified, NA is used for all samples. For mixtures, this variable indicates what samples come from the same "stratum" - to be analyzed together.
loci	a vector of loci to use. If not specified, all loci are used.

Value

a tibble

---

exportSNPPIT *write a SNPPIT input file. Will warn about skipping loci with > 2 alleles.*

Description

write a SNPPIT input file. Will warn about skipping loci with > 2 alleles.

Usage

```
exportSNPPIT(
  x,
  filename,
  baseline,
  mixture,
  loci = NULL,
  errorRate = 0.005,
  POPCOLUMN_SEX = NULL,
  POPCOLUMN_REPRO_YEARS = NULL,
  POPCOLUMN_SPAWN_GROUP = NULL,
  OFFSPRINGCOLUMN_BORN_YEAR = NULL,
  OFFSRPINGCOLUMN_SAMPLE_YEAR = NULL,
  OFFSPRINGCOLUMN_AGE_AT_SAMPLING = NULL
)
```

Arguments

x	an EFGLdata object
filename	the name of the file to write
baseline	a vector of pops to use as the baseline (potential parents).
mixture	a vector of pops to use as the mixture (potential offspring).
loci	a vector of loci to include. If not specified, all loci are used.
errorRate	per allele error rate for all loci
POPCOLUMN_SEX	metadata column with sex info (coded as M, F, and ?)

POPCOLUMN_REPRO_YEARS
: metadata column with repro years

POPCOLUMN_SPAWN_GROUP
: metadata column with spawn group

OFFSPRINGCOLUMN_BORN_YEAR
: metadata column with birth year

OFFSRPINGCOLUMN_SAMPLE_YEAR
: metadata column with sample year

OFFSPRINGCOLUMN_AGE_AT_SAMPLING
: metadata column with age at sampling

Value

nothing, just writes a file

`exportStructure` *write a Structure input file.*

Description

Columns are Ind, Pop (converted to integers), then genotypes. Missing alleles are coded as -9

Usage

```
exportStructure(x, filename, pops = NULL, loci = NULL)
```

Arguments

`x`	an EFGLdata object
`filename`	the name of the file to write
`pops`	a vector of pops to include. If not specified, all pops are used.
`loci`	a vector of loci to include. If not specified, all loci are used.

Value

nothing, just writes a file

`genoSuccess` *calculate genotyping success of individuals (uses only allele 1 for each genotype - assumes if allele 1 is (is not) NA, so is (is not) allele 2)*

Description

calculate genotyping success of individuals (uses only allele 1 for each genotype - assumes if allele 1 is (is not) NA, so is (is not) allele 2)

Usage

```
genoSuccess(x, loci = NULL)
```

Arguments

`x`	an EFGLdata object
`loci`	a vector of loci to include. If not specified, all loci are used.

Value

a tibble giving the genotyping success of each individual as a proportion and number of missing genotypes

`getInds` *get a vector of individuals present*

Description

get a vector of individuals present

Usage

```
getInds(x, pops = NULL)
```

Arguments

`x`	an EFGLdata object
`pops`	a vector of pops that you want individual names for. If not specified, names for all pops are returned

Value

a vector of the Individual names present

getLoci *get a vector of loci names present*

Description

get a vector of loci names present

Usage

```
getLoci(x)
```

Arguments

x	an EFGLdata object

getMeta *get a vector of metadata column names present*

Description

get a vector of metadata column names present

Usage

```
getMeta(x)
```

Arguments

x	an EFGLdata object

getPops *get a vector of populations (pedigrees) present*

Description

get a vector of populations (pedigrees) present

Usage

```
getPops(x)
```

Arguments

x	an EFGLdata object

Value

a vector of the unique population names present

`lociSuccess` *calculate genotyping success of loci (uses only allele 1 for each genotype - assumes if allele 1 is (is not) NA, so is (is not) allele 2)*

Description

calculate genotyping success of loci (uses only allele 1 for each genotype - assumes if allele 1 is (is not) NA, so is (is not) allele 2)

Usage

```
lociSuccess(x)
```

Arguments

`x`	an EFGLdata object

Value

a tibble giving the genotyping success of each locus as a proportion

`moveInds` *combine individuals into one population AND REMOVE the previous entry for those individuals*

Description

combine individuals into one population AND REMOVE the previous entry for those individuals

Usage

```
moveInds(x, inds, newName)
```

Arguments

`x`	an EFGLdata object
`inds`	a vector of individuals to put in the new pop
`newName`	a string giving the name of population to add the individuals too. This can be a new pop or an existing pop (a warning is issued if existing).

Value

an EFGLdata object

movePops *combine populations into one AND REMOVE the old populations*

Description

combine populations into one AND REMOVE the old populations

Usage

```
movePops(x, pops, newName)
```

Arguments

x	an EFGLdata object
pops	a vector of populations to combine
newName	a string giving the name of the population to combine pops into. This can be a new pop or an existing pop (a warning is issued if existing).

Value

an EFGLdata object

numInds *get the number of individuals present in each pop*

Description

get the number of individuals present in each pop

Usage

```
numInds(x, pops = NULL)
```

Arguments

x	an EFGLdata object
pops	a vector of pops that you want individual names for. If not specified, numbers for all pops are returned

Value

a named vector with the number of individuals in each pop

`print.EFGLdata` *print method for EFGLdata*

Description

print method for EFGLdata

Usage

```
## S3 method for class 'EFGLdata'
print(x, ...)
```

Arguments

`x`	an EFGLdata object
`...`	ignored

`readInData` *read in data from a Progeny-style output file or matrix/dataframe/tibble adn create an EFGLdata object*

Description

read in data from a Progeny-style output file or matrix/dataframe/tibble adn create an EFGLdata object

Usage

```
readInData(
  input,
  genotypeStart = NULL,
  pedigreeColumn = 1,
  nameColumn = 2,
  convertNames = TRUE,
  convertMetaDataNames = TRUE,
  missingAlleles = c("0", "00", "000"),
  guess_max = 10000
)
```

Arguments

`input`	Either a character string to the tab-separated input file with a header row or a matrix/dataframe/tibble. Structure of the input: one row per individual. One column giving pedigree (population) names, one column giving individual names, optional additional metadata columns, genotype columns. Pedigree and individual name columns can be anywhere, if specified. Genotype columns MUST be consecutive and be the right most columns. Genotypes are given as two columns per call (diploidy assumed).

genotypeStart	The column number that genotypes start at. If not specified, the first column with a column name ending in ".A1", ".a1", "-A1", or "-a1" is chosen.
pedigreeColumn	The column number that contains pedigree (population) names.
nameColumn	The column number that contains individual names. These MUST be unique.
convertNames	TRUE to convert genotype and pedigree names in the same way that IDFGEN does (remove special characters from both and remove "." from genotype names).
convertMetaDataNames	TRUE to remove special characters and spaces from metadata column names. This makes accessing them easier.
missingAlleles	a vector of values (not NA) to treat as missing alleles. They will be converted to NA.
guess_max	If input is a character string, this is the maximum number of lines to use when guessing input data types. Making this smaller results in quicker loading, making it larger can fix some parsing errors

Value

An EFGLdata object, which is just a list with two elements. The first element is a tibble with genotype data, the second is a tibble with metadata

---

removeInds *remove individuals from an EFGLdata object*

Description

remove individuals from an EFGLdata object

Usage

```
removeInds(x, inds)
```

Arguments

x	an EFGLdata object
inds	a vector of individuals to remove

Value

an EFGLdata object

`removeLoci` *remove loci from an EFGLdata object*

Description

remove loci from an EFGLdata object

Usage

```
removeLoci(x, lociRemove)
```

Arguments

`x`	an EFGLdata object
`lociRemove`	a vector of loci names to remove

`removePops` *remove pops from an EFGLdata object*

Description

remove pops from an EFGLdata object

Usage

```
removePops(x, pops)
```

Arguments

`x`	an EFGLdata object
`pops`	a vector of pops to remove

Value

an EFGLdata object

`whichLower` *Identify which individual out of duplicate pairs has the lower genotyping success*

Description

Identify which individual out of duplicate pairs has the lower genotyping success

Usage

```
whichLower(dupTable, geno_success)
```

Arguments

`dupTable`	the output of close_matching_samples (from rubias)
`geno_success`	the output of genoSuccess

Value

a vector of unique individual names representing the individuals with lower genotyping success from each pair

Index

∗ **datasets**
    exampleData, 6

aRich, 2

calcAF, 3
calcHet, 3
cleanGrandma, 4
combineEFGLdata, 4
construct_EFGLdata, 5
createF1Hybrids, 5

dumpTable, 6

exampleData, 6
exportCKMRsimAF, 7
exportCKMRsimLG, 7
exportGenAlEx, 8
exportGenePop, 8
exportGrandma, 9
exportHierFstat, 10
exportPlink, 10
exportProgenyStyle, 11
exportRubias_baseline, 12
exportRubias_mixture, 12
exportSNPPIT, 13
exportStructure, 14

genoSuccess, 15
getInds, 15
getLoci, 16
getMeta, 16
getPops, 16

lociSuccess, 17

moveInds, 17
movePops, 18

numInds, 18

print.EFGLdata, 19

readInData, 19
removeInds, 20
removeLoci, 21
removePops, 21

whichLower, 22