

Package ‘EFGLmh’

June 17, 2020

Type Package

Title Functions For Working With Microhaps for EFGL

Version 0.1.0

Maintainer Thomas Delomas <thomas.delomas@idfg.idaho.gov>

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

License MIT + LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.1.0

Depends R (>= 2.10)

Suggests knitr,
rmarkdown,
tidyverse

VignetteBuilder knitr

R topics documented:

aRich	2
calcHet	2
cleanGrandma	3
combineEFGLdata	3
construct_EFGLdata	4
dumpTable	4
exportGenAlEx	5
exportGenePop	5
exportGrandma	6
exportRubias_baseline	6
exportRubias_mixture	7
exportSNPPIT	7

genoSuccess	8
getInds	9
getLoci	9
getMeta	9
getPops	10
lociSuccess	10
moveInds	11
movePops	11
numInds	12
print.EFGLdata	12
readInData	13
removeInds	14
removeLoci	14
removePops	15
whichLower	15

Index	16
--------------	-----------

aRich	<i>calculate allelic richness of loci</i>
-------	---

Description

calculate allelic richness of loci

Usage

aRich(x)

Arguments

x an EFGLdata object

Value

a tibble giving the allelic richness of each locus

calcHet	<i>calculate expected and observed heterozygosity within populations</i>
---------	--

Description

calculate expected and observed heterozygosity within populations

Usage

calcHet(x)

Arguments

x an EFGLdata object

Value

a tibble

cleanGrandma	<i>convenience function to remove loci with all fails or no variation from gRandma input</i>
--------------	--

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	<i>combine multiple EFGL objects into one</i>
-----------------	---

Description

combine multiple EFGL objects into one

Usage

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

Arguments

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.
...	multiple EFGLdata objects separated by commas

construct_EFGLdata	<i>some basic checks on EFGLdata objects</i>
--------------------	--

Description

some basic checks on EFGLdata objects

Usage

```
construct_EFGLdata(x)
```

Arguments

x	an EFGLdata object
---	--------------------

dumpTable	<i>wrapper for write table with commonly used options - carried over from IDFGEN</i>
-----------	--

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

exportGenAIEx	<i>write a GenAIEx input file</i>
---------------	-----------------------------------

Description

write a GenAIEx input file

Usage

```
exportGenAIEx(x, filename, pops = NULL, loci = NULL, title = "")
```

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
return	nothing, just writes a file

exportGenePop	<i>write a genepop input file</i>
---------------	-----------------------------------

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
return	nothing, just writes a file

exportGrandma	<i>export a gRandma baseline or mixture</i>
---------------	---

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

exportRubias_baseline	<i>export a rubias baseline</i>
-----------------------	---------------------------------

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	<i>export a rubias mixture</i>
----------------------	--------------------------------

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	<i>write a SNPPIT input file. Will warn about skipping loci with > 2 alleles.</i>
--------------	--

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
return	nothing, just writes a file

genoSuccess	<i>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)</i>
-------------	---

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	<i>get a vector of individuals present</i>
---------	--

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	<i>get a vector of loci names present</i>
---------	---

Description

get a vector of loci names present

Usage

```
getLoci(x)
```

Arguments

x	an EFGLdata object
---	--------------------

getMeta	<i>get a vector of metadata column names present</i>
---------	--

Description

get a vector of metadata column names present

Usage

```
getMeta(x)
```

Arguments

x	an EFGLdata object
---	--------------------

getPops	<i>get a vector of populations (pedigrees) present</i>
---------	--

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	<i>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)</i>
-------------	--

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	<i>combine individuals into one population AND REMOVE the previous entry for those individuals</i>
----------	--

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	<i>combine populations into one AND REMOVE the old populations</i>
----------	--

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	<i>get the number of individuals present in each pop</i>
---------	--

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	<i>print method for EFGLdata</i>
----------------	----------------------------------

Description

print method for EFGLdata

Usage

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

Arguments

x	an EFGLdata object
---	--------------------

readInData	<i>read in data from a Progeny-style output file or matrix/dataframe/tibble and create an EFGLdata object</i>
------------	---

Description

read in data from a Progeny-style output file or matrix/dataframe/tibble and 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 IDF-GEN 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	<i>remove individuals from an EFGLdata object</i>
------------	---

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	<i>remove loci from an EFGLdata object</i>
------------	--

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	<i>remove pops from an EFGLdata object</i>
------------	--

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	<i>Identify which individual out of duplicate pairs has the lower genotyping success</i>
------------	--

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

aRich, [2](#)

calcHet, [2](#)
cleanGrandma, [3](#)
combineEFGLdata, [3](#)
construct_EFGLdata, [4](#)

dumpTable, [4](#)

exportGenAlEx, [5](#)
exportGenePop, [5](#)
exportGrandma, [6](#)
exportRubias_baseline, [6](#)
exportRubias_mixture, [7](#)
exportSNPPIT, [7](#)

genoSuccess, [8](#)
getInds, [9](#)
getLoci, [9](#)
getMeta, [9](#)
getPops, [10](#)

lociSuccess, [10](#)

moveInds, [11](#)
movePops, [11](#)

numInds, [12](#)

print.EFGLdata, [12](#)

readInData, [13](#)
removeInds, [14](#)
removeLoci, [14](#)
removePops, [15](#)

whichLower, [15](#)