Package ‘Ease’

November 7, 2022

Type Package
Title Simulating Explicit Population Genetics Models
Version 0.1.2
Author Ehouarn Le Faou <ehouarnlefaou@orange.fr> [aut, cre]
Maintainer Ehouarn Le Faou <ehouarnlefaou@orange.fr>
Description Implementation in a simple and efficient way of fully customisable population genetics simulations, considering multiple loci that have epistatic interactions. Specifically suited to the modelling of multilocus nucleocytoplasmic systems (with both diploid and haploid loci), it is nevertheless possible to simulate purely diploid (or purely haploid) genetic models.

Examples of models that can be simulated with Ease are numerous, for example models of genetic incompatibilities as presented by Marie-Orleach et al. (2022) <doi:10.1101/2022.07.25.501356>.

Many others are conceivable, although few are actually explored, Ease having been developed in particular to provide a solution so that these kinds of models can be simulated simply.

License MIT + file LICENSE
Encoding UTF-8
Imports methods, Rcpp (>= 1.0.7), RcppProgress (>= 0.1), stats
LinkingTo Rcpp, RcppProgress
RoxygenNote 7.2.1
Suggests rmarkdown, knitr, testthat (>= 3.0.0)
VignetteBuilder knitr
Config/testthat/edition 3
NeedsCompilation yes
Repository CRAN
Date/Publication 2022-11-07 14:40:08 UTC
R topics documented:

- alleleFreqMatGeneration ............................................. 3
- areThereHomoz .......................................................... 4
- catn ........................................................................... 4
- check_genome ................................................................ 5
- check_metapopulation ...................................................... 6
- check_mutationMatrix ...................................................... 6
- check_population .......................................................... 7
- check_selection ............................................................. 7
- extractAlleleComb ......................................................... 8
- Genome-class ................................................................. 8
- genotyping ..................................................................... 9
- getCustomOutput ............................................................ 10
- getRecords ................................................................. 11
- getResults ...................................................................... 11
- haploCrossMatrix .......................................................... 12
- haplotyping .................................................................... 13
- IDgenomeGeneration ....................................................... 13
- IDgenotypeGeneration ..................................................... 14
- IDhaplotypeGeneration ................................................... 15
- initialize_Genome-method ............................................... 15
- initialize_Metapopulation-method ..................................... 16
- initialize_MutationMatrix-method .................................... 17
- initialize_Population-method ............................................ 17
- initialize_Selection-method ............................................ 18
- is_correct_transition_matrix .......................................... 19
- is_default_matrix .......................................................... 20
- is_probability_matrix ..................................................... 20
- isAffected ...................................................................... 21
- isHaploSelectFormula ...................................................... 21
- listing .......................................................................... 22
- meiosisMatrix ................................................................. 22
- Metapopulation-class ....................................................... 23
- METAPOP_SIMULATION .................................................... 24
- mutation ....................................................................... 26
- MutationMatrix-class ...................................................... 27
- mutMatFriendly ............................................................. 28
- mutMatRates ................................................................. 28
- outFunct ....................................................................... 29
- Population-class ............................................................ 30
- print_Genome-method ....................................................... 31
- print_Metapopulation-method ......................................... 31
- print_MutationMatrix-method ........................................ 32
- print_Population-method ................................................ 32
- print_Selection-method .................................................. 33
- recombinationMatrix ...................................................... 33
- rowResultGen ............................................................... 34
alleleFreqMatGeneration

Generation of the matrix for calculating allelic frequencies

Description

Generates a matrix that allows to go from genotypic frequencies to allelic frequencies.

Usage

alleleFreqMatGeneration(genomeObj)

Arguments

genomeObj  a Genome object

Details

An allele frequency matrix is a matrix with rows equal to the number of genotypes and columns equal to the number of alleles. By multiplying a row matrix of genotype frequencies we obtain a row matrix of associated allele frequencies.

Value

A matrix for calculating allelic frequencies from genotypes frequencies.
**Author(s)**

Ehouarn Le Faou

---

**areThereHomoz**

*Are there any allelic combinations including homozygosity*

**Description**

Test if there are homozygotes in the specified allelic combinations of a selection formula

**Usage**

`areThereHomoz(formula)`

**Arguments**

- `formula` (a selection formula)

**Value**

Logical indicating if there are homozygotes

---

**catn**

*Concatenate, print and line break*

**Description**

Object output in the same way as the function `cat` but adding a line break at the end.

**Usage**

`catn(..., file = "", sep = " ", fill = FALSE, labels = NULL, append = FALSE)`

**Arguments**

- `...` (see `cat`)
- `file` (see `cat`)
- `sep` (see `cat`)
- `fill` (see `cat`)
- `labels` (see `cat`)
- `append` (see `cat`
check.genome

Details

See cat.

Value

None (invisible NULL).

Author(s)

Ehouarn Le Faou

---

check.genome  

*The validity check for the Genome class*

Description

The validity check for the Genome class

Usage

check.genome(object)

Arguments

- object: a Genome object

Value

A logical corresponding to whether the object is a correct Genome object.

Author(s)

Ehouarn Le Faou
check.metapopulation  The validity check for the Metapopulation class

Description

The validity check for the Metapopulation class

Usage

check.metapopulation(object)

Arguments

object  a Metapopulation object

Value

a boolean corresponding to whether the object is a correct Metapopulation object.

Author(s)

Ehouarn Le Faou

check.mutationMatrix  The validity check associated with the MutationMatrix class

Description

The validity check associated with the MutationMatrix class

Usage

check.mutationMatrix(object)

Arguments

object  an object of class MutationMatrix

Value

A logical corresponding to whether x is a correct MutationMatrix object.

Author(s)

Ehouarn Le Faou
The validity check for the Population class

Usage

check.population(object)

Arguments

object a Population object

Value

A boolean corresponding to whether the object is a correct Population object.

Author(s)

Ehouarn Le Faou

The validity check for the Selection class

Usage

check.selection(object)

Arguments

object Selection object

Value

A logical corresponding to whether the object is a correct Selection object.

Author(s)

Ehouarn Le Faou
extractAlleleComb  Extract the allele combination

Description
Conversion of an allelic combination defined in a selection formula into the vector listing the alleles present (alleles that must be in the homozygous state appear 2 times, 1 time for heterozygous).

Usage
extractAlleleComb(xVect)

Arguments
xVect       allelic combination extracted from a selection formula.

Value
the list of alleles that must be present in the genotype to match the input allelic combination

Author(s)
Ehouarn Le Faou

Genome-class  Genome class

Description
The Genome class allows to define all the characteristics of the genome which will be used as a basis for the construction of transition matrices from one generation to another in simulations of the model.

Details
A genome includes the list of all possible haplotypes and genotypes resulting from the combination of the alleles defined in input. As the Ease package was originally built for population genetics simulations including both diploid and haploid loci, it is necessary that both types of loci are defined. Despite this, the user can define only diploid or only haploid loci if they wish. If no diploid locus is defined, one is automatically generated with only one allele, thus not influencing the simulation. The same applies if no haploid locus is defined.

Each locus is described by a vector of factors which are the names of the possible alleles at that locus. All diploid (resp. haploid) loci thus defined are grouped in a list, called listDipLoci (resp. listHapLoci). Therefore, a Genome class object has two lists of loci defined in this way, one for diploid loci, one for haploid loci. The alleles and loci (diploid and haploid) must all have different names so that no ambiguity can persist.
If several are defined, the order of the diploid loci in the list is not trivial. The rates of two-to-one combinations between them must indeed be defined by the vector recRate. For example, if three diploid loci are defined, recRate must be of length 2, the first of its values defining the recombination rate between the first and second loci, the second of its values the recombination rate between the second and third loci. For example, if we want to define two groups of two loci that are linked to each other but are on two different chromosomes, we can define a recRate = c(0.1, 0.5, 0.1). The first two loci are thus relatively linked (recombination rate of 0.1), as are the last two loci. On the other hand, the recombination rate of 0.5 between the second and third loci ensures that the two groups are independent.

Slots

- listHapLoci: a list of haploid loci
- listDipLoci: a list of diploid loci
- recRate: a two-by-two recombination rate vector
- nbHL: the number of haploid loci
- nbDL: the number of diploid loci
- listLoci: the list of all loci
- haplotypesHL: haplotypes of haploid loci only
- haplotypesDL: haplotypes of diploid loci only
- haplotypes: haplotypes of all loci
- alleles: the vector of all the alleles
- nbAlleles: the number of alleles
- nbHaplo: the number of haplotypes
- IDhaplotypes: IDs of haplotypes
- genotypes: the list of genotypes
- nbGeno: the number of genotypes
- IDgenotypes: IDs of genotypes
- IDgenome: ID of the genome

Author(s)

Ehouarn Le Faou

Description

Generation of genotypes associated with a Genome object.
Usage

getCustomOutput(genomeObj)

Arguments

genomeObj a Genome object.

Details

The output genotypes are described as a list of three matrices. A genotype consists of two diploid haplotypes (first two matrices) and one haploid haplotype (third matrix), which are read at the same row number on all three matrices.

Value

A list of matrices describing genotypes in rows.

Author(s)

Ehouarn Le Faou

description

Getting the custom output

Usage

getCustomOutput(metapop)

Arguments

metapop a Metapopulation objects

Value

The list generated through the custom result function, if at least it was specified during the simulation of the Metapopulation.

Author(s)

Ehouarn Le Faou
**getRecords**  

*Getting the simulation results*

**Description**

Getting the simulation results

**Usage**

```r
getRecords(metapop)
```

**Arguments**

- `metapop`  
  a Metapopulation objects

**Value**

A list where each item is associated with a simulation. Each of these elements consists of a list of data.frames, one per population. These data.frames consist of the same columns as the results (see `getResults` documentation), except that they do not include the stop conditions.

**Author(s)**

Ehouarn Le Faou

---

**getResults**  

*Getting the simulation results*

**Description**

Getting the simulation results

**Usage**

```r
getResults(metapop)
```

**Arguments**

- `metapop`  
  a Metapopulation objects

**Value**

A data.frame where each line corresponds to a simulation. The results include:
- the last generation reached (the threshold or the generation that first verified at least one of the stopping conditions)
- the final population size
- the genotype frequencies
- allelic frequencies
- the reason(s) for the stop (either the threshold was reached, i.e. `unstopped` or the stop condition(s) that was (were) reached, in the form of boolean values
- Average fitness (individual, gamete production and gametic)
Author(s)

Ehouarn Le Faou

Description

Generation of the haplotype crossing matrix associated to a Genome object.

Usage

haploCrossMatrix(genomeObj)

Arguments

genoMrhoObj a Genome object

Details

A crossover matrix is a square matrix of size equal to the number of haplotypes. It describes for each combination of two gametic haplotypes the genotype index resulting from their syngamy. In the general case it is not a symmetrical matrix (it is if a single haploid locus with a single allele is defined), because the transmission of haploid loci is only maternal, therefore non-symmetrical as is the transmission of diploid loci. It is therefore necessary to enter the haplotype frequencies of male gametes in the columns and the haplotype frequencies of female gametes in the rows during the calculations (this is done in the simulations).

Value

An haplotype crossing matrix.

Author(s)

Ehouarn Le Faou
haplotyping

**Description**

Generation of haplotypes associated with a Genome object.

**Usage**

```r
haplotyping(genomeObj)
```

**Arguments**

- `genomeObj`: a Genome object

**Details**

The generated haplotypes are output as a list of three enumeration in the form of matrices of alleles (each row corresponding to an haplotype, each column to a locus). The first enumeration corresponds to haplotypes considering only haploid loci, the second only diploid loci and the third all loci (with two matrices, 1 for haploid loci, 1 for diploid loci).

**Value**

A list of matrices describing haplotypes in rows.

**Author(s)**

Ehouarn Le Faou

---

**IDgenomeGeneration**

**Genome identifier**

**Description**

Generation of the input genome ID, i.e. the concatenation in string form of the names of the loci and alleles constituting this genome.

**Usage**

```r
IDgenomeGeneration(listLoci, alleles)
```

**Arguments**

- `listLoci`: the list of all loci
- `alleles`: the vector of all the alleles
**Value**

The genome ID as a character string.

**Author(s)**

Ehouarn Le Faou

---

**IDgenotypeGeneration**  *Genotype identifier*

---

**Description**

Generation of the input genotype ID, i.e. the concatenation in string form of the names of the alleles constituting these haplotypes (the two from the diploid genome and the one from the haploid genome).

**Usage**

`IDgenotypeGeneration(dl1, dl2, hl = NULL)`

**Arguments**

- `dl1` the first diploid haplotype as a character (or factors) vector.
- `dl2` the second diploid haplotype as a character (or factors) vector.
- `hl` the haploid haplotype as a character (or factors) vector.

**Value**

The genotype ID as a character string.

**Author(s)**

Ehouarn Le Faou
IDhaplotypeGeneration | Haplotype identifier

Description
Generation of the input haplotype ID, i.e. the concatenation in string form of the names of the alleles constituting this haplotype.

Usage
IDhaplotypeGeneration(dl, hl)

Arguments
- dl: the diploid haplotype as a character (or factors) vector.
- hl: the haploid haplotype as a character (or factors) vector.

Value
The haplotype ID as a character string.

Author(s)
Ehouarn Le Faou

initialize, Genome-method
Initialize method for the Genome class

Description
Initialize method for the Genome class

Usage
## S4 method for signature 'Genome'
initialize(.Object, listHapLoci, listDipLoci, recRate)

Arguments
- .Object: a Genome object
- listHapLoci: a list of haploid loci
- listDipLoci: a list of diploid loci
- recRate: a two-by-two recombination rate vector
**initialize,Metapopulation-method**

*Initialize method for the Metapopulation class*

---

**Description**

Initialize method for the Metapopulation class

**Usage**

```r
## S4 method for signature 'Metapopulation'
initialize(.Object, populations, migMat)
```

**Arguments**

- `.Object`: a Metapopulation object
- `populations`: list of Population object(s)
- `migMat`: migration matrix

**Value**

a Metapopulation object

**Author(s)**

Ehouarn Le Faou
initialize,MutationMatrix-method

*Initialize method for the MutationMatrix class*

### Description

Initialize method for the MutationMatrix class

### Usage

```r
## S4 method for signature 'MutationMatrix'
initialize(.Object, genomeObj, mutHapLoci, mutDipLoci)
```

### Arguments

- `.Object`: a MutationMatrix object
- `genomeObj`: a Genome object
- `mutHapLoci`: a list of haploid locus by locus allelic mutation matrices.
- `mutDipLoci`: a list of diploid locus by locus allelic mutation matrices.

### Value

A MutationMatrix object

### Author(s)

Ehouarn Le Faou

initialize,Population-method

*Initialize method for the Population class*

### Description

Initialize method for the Population class

### Usage

```r
## S4 method for signature 'Population'
initialize(
  .Object, name, size, dioecy, selfRate,
)```
initialize, Selection-method

```r
initialize, Selection-method
demography, growthRate, initGenoFreq, genomeObj, initPopSize, selectionObj, mutMatrixObj
)
```

**Arguments**

- `.Object` a Population object
- `name` the name of the population.
- `size` the size of the population.
- `dioecy` logical indicating whether the population is dioecious or not (hermaphrodite).
- `selfRate` the selfing rate of the population.
- `demography` logical indicating whether the population has stochastic demography (this does not include migration), i.e. non-constant size and potentially population growth or decay, depending on the situation it is in.
- `growthRate` growth rate of the population.
- `initGenoFreq` A row matrix of the size of the genotype number describing the initial allele frequencies common to all simulations.
- `genomeObj` a Genome object.
- `initPopSize` initial population size, knowing that if the demography is extinct, the initial population size will automatically be set equal to the population size.
- `selectionObj` a Selection object.
- `mutMatrixObj` a MutationMatrix object.

**Value**

a Population object

**Author(s)**

Ehouarn Le Faou

---

**initialize, Selection-method**

*Initialize method for the Selection class*

**Description**

Initialize method for the Selection class
is.correct.transition.matrix

Usage

## S4 method for signature 'Selection'
initialize(.Object, genomeObj)

Arguments

,Object a Selection object
genomeObj a Genome object

Value

A Selection object

Author(s)

Ehouarn Le Faou

---

is.correct.transition.matrix

*Test if a matrix is a correct transition matrix*

Description

Test if a matrix is a correct transition matrix

Usage

is.correct.transition.matrix(x, type, name)

Arguments

x a matrix.
type type of the matrix (mutation matrix ? recombination matrix ?)
name the name of the matrix.

Value

A logical corresponding to whether x is a correct transition matrix, i.e. a square matrix with dimensions greater than 0 and whose rows sum to 1.

Author(s)

Ehouarn Le Faou
is.default.matrix   Test if a matrix is a default matrix

Description
Test if a matrix is a default matrix

Usage
is.default.matrix(x)

Arguments
x  a matrix.

Value
A logical corresponding to whether x is a default matrix (matrix of dimension 0x0).

Author(s)
Ehouarn Le Faou

is.probability.matrix   Test if a matrix is of probability

Description
Test if a matrix is of probability

Usage
is.probability.matrix(x)

Arguments
x  a matrix.

Value
A logical corresponding to whether x is a probability matrix (sum of rows equal to 1).

Author(s)
Ehouarn Le Faou
isAffected  Is this haplo/geno-type affected?

Description
Determination for a given genotype or haplotype whether it contains the allelic combination under selection

Usage
isAffected(refDNAtype, selDNAtype)

Arguments
refDNAtype  the reference allelic combination (that of a genotype or a haplotype
selDNAtype  the selected allelic combination

Value
a logic indicating whether the reference genotype or haplotype is affected by the allelic combination under selection

Author(s)
Ehouarn Le Faou

isHaploSelectFormula  Are there any allelic combinations including homozygosity

Description
Test if there are homozygotes in the specified allelic combinations of a list of selection formulas

Usage
isHaploSelectFormula(selectFormula)

Arguments
selectFormula  a list of selection formula

Value
logical indicating if there are homozygotes
Listing

Listing for display

Description
Listing from the elements of a vector by producing a string, with comma separation between each element and the word "and" between the last two elements.

Usage
listing(vect)

Arguments
vect a vector of any class.

Value
A listing of the elements of the input vector as a string.

Author(s)
Ehouarn Le Faou

meiosisMatrix

Meiosis matrix generation

Description
Generation of the meiosis matrix associated to a Genome object.

Usage
meiosisMatrix(genomeObj)

Arguments
genomeObj a Genome object

Details
A meiosis matrix is a matrix where the number of rows is equal to the number of genotypes and the number of columns to the number of haplotypes. It is a matrix that allows to pass from parental genotypes to gametic haplotypes by meiosis. It is a probability matrix in that the sum of the values in each row is equal to 1. For a given genotype, the row associated with it describes the probabilistic proportions that lead by meiosis to the production of the other genotypes (and of itself if there are no mutations).
Meta-population-class

Value

A meiosis matrix (probability matrix that associates to each genotype in a row the probability of producing each of the possible haplotypes).

Author(s)

Ehouarn Le Faou

---

Description

The class `Meta-population` is used to centralise the information relating to the populations that we want to simulate, as well as to define the migration conditions between them if there are several. This class is thus defined by a list of objects `Population` and a migration matrix.

Slots

- `populations` list of objects `Population`
- `nbPop` number of populations
- `names` names of the populations
- `migMat` migration matrix between population (if there is more than one)
- `sizes` sizes of the populations
- `dioecies` sexual systems of the populations (they must all be the same)
- `selfRates` selfing rates of the populations
- `demographies` demography parameter of the populations
- `growthRates` growth rates of the populations
- `initPopSizes` initial population sizes of the populations
- `initGenoFreqs` initial genotypic frequencies of the populations
- `genome` a `Genome` object
- `genomeIDs` ID of the `Genome` object
- `mutMat` a `MutationMatrix` object
- `selection` a `Selection` object
- `recMat` recombination matrix
- `meiosisMat` a meiosis matrix
- `haploCrossMat` an haplotype crossing matrix
- `haploCrossMatNamed` an haplotype crossing matrix with names of genotypes instead of their indices
- `gametogenesisMat` a gametogenesis matrix
alleleFreqMat a matrix for calculating allelic frequencies
rawOutputSimul raw output of the simulation function, its refinement is done directly afterwards
in the simulate method
stopCondition list of stop conditions for the simulation (if required)
IDstopCondition names of stop conditions. They are given an arbitrary name if none is given by
the user.
results data.frame.
records list.
customOutput list.

Author(s)
Ehouarn Le Faou

Description
Simulation of a metapopulation

Usage
METAPOP_SIMULATION(nbPop, ids, migMat, nsim, verbose, recording, recordGenGap, drift,
nbHaplo, nbGeno, idGeno, nbAlleles, idAlleles, nbLoci, initGenoFreq, meiosisMat,
gametogenesisMat, popSize, threshold, dioecy, selfRate,
stopCondition,
IDstopCondition,
haploCrossMat,
alleleFreqMat,
gamFit,
indFit,
gamProdFit,
demography,
growthRate,
initPopSize,
nameOutFunct
)

Arguments

nbPop number of populations in the metapopulation
ids population IDs
migMat migration matrix
nsim number of simulations
verbose boolean determining if the progress of the simulations should be displayed or not (useful in case of many simulations)
recording a boolean indicating whether to record all mutations, i.e. to record allelic and genotypic frequencies along the simulations
recordGenGap the number of generations between two records during simulation, if the record parameter is TRUE. Whatever the value of this parameter, both the first and the last generation will be included in the record
drift a boolean indicating whether genetic drift should be considered (i.e. whether deterministic simulations are performed or not)
nbHaplo number of haplotypes
nbGeno number of genotypes
idGeno genotypes ID
nbAlleles number of alleles for each loci
idAlleles alleles ID
nbLoci number of loci
initGenoFreq list of initial genotype frequencies in the populations
meiosisMat meiosis matrix
gametogenesisMat gametogenesis matrix
popSize list population sizes
threshold threshold for simulations
dioecy whether the population(s) is dioecious or not (hermaphroditism)
selfRate list of the selfing rate in populations (only for hermaphroditic population)
mutation

Description
Utility function to easily generate a mutation matrix (see setMutationMatrix).

Usage
mutation(from, to, rate)

Arguments
- from: name of the original allele
- to: name of the mutant allele
- rate: rate at which the mutation occurs

Details
Mutation occurs from one allele to another at a specific rate. Please take care to define alleles as traits, that these alleles are present in the genome you are using and that the alleles are associated with the same locus.

Value
A standardised list of input parameters that will be used by the function setMutationMatrix to generate the mutation matrix.
### Example with two loci, each with two alleles ###

# Definition of the genome
DL <- list(dl = c("A", "a"))
HL <- list(hl = c("B", "b"))
geno <- setGenome(listHapLoci = HL, listDipLoci = DL)

# The mutation function allows each transition from one allele to 
# another to be defined individually, to produce the mutation matrix
# as follows:
mutMatrixObj <- setMutationMatrix(geno,
  mutations = list(
    mutation(from = "A", to = "a", rate = 0.1),
    mutation(from = "B", to = "b", rate = 0.1)
  )
)

---

**MutationMatrix-class**  
**Mutation matrix**

**Description**

A mutation matrix is used to simulate mutations that affect loci. An object of the class MutationMatrix does not only contain a (haplotypic) mutation matrix. It also contains the attributes necessary for the construction and easy-to-read display of this matrix.

**Details**

The mutation matrix itself is a square matrix of size equal to the number of haplotypes. It is a probability matrix in that the sum of the values in each row is equal to 1. For a given haplotype, the row associated with it describes the probabilistic proportions that lead by mutation of this haplotype to the production of the other haplotypes (and of itself if there are no mutations).

**Slots**

- mutHapLoci a list of haploid locus by locus allelic mutation matrices.
- mutDipLoci a list of diploid locus by locus allelic mutation matrices.
- mutLoci a list concatenating mutHapLoci and mutDipLoci
- nbALDL a vector of the number(s) of alleles at each haploid locus
- nbALHL a vector of the number(s) of alleles at each diploid locus
- mutationMatrix the haplotypic mutation matrix
- nbHaplo the number of haplotypes
- nbDL the number of diploid loci
- nbHL the number of haploid loci
- haplotypes the enumeration of haplotypes
- IDgenome ID of the associated genome
**Author(s)**
Ehouarn Le Faou

---

### `mutMatFriendly`

*Individual mutation definition to allelic mutation matrices*

**Description**
Translation of the list of individually defined mutations into allelic mutation matrices which are then used to generate the genotypic mutation matrix.

**Usage**
```
mutMatFriendly(genomeObj, mutations)
```

**Arguments**
- `genomeObj` a Genome object
- `mutations` list of mutations defined individually with the function `mutation`

**Value**
A list of the two list of allelic mutation matrices, for haploid and diploid loci respectively.

**Author(s)**
Ehouarn Le Faou

---

### `mutMatRates`

*Mutation matrix from rates*

**Description**
Generation of a mutation matrix from the allele enumeration vector of a loci and the forward and backward mutation rates.

**Usage**
```
mutMatRates(alleles, forwardMut, backwardMut)
```

**Arguments**
- `alleles` allele enumeration vector of a locus
- `forwardMut` forward mutation rate
- `backwardMut` backward mutation rate
Details

See MutationMatrix for more details on mutation matrices.

Value

An allelic mutation matrix (probability matrix which associates to each allele in a row the probability of mutating or not to the other alleles of the locus in question).

Author(s)

Ehouarn Le Faou

Description

Allow to produce a custom output for a simulation.

Usage

outFunct(pop)

Arguments

- pop: list of some characteristics of the population:
  - customOutput: list of all previous savings
  - gen: generation
  - freqGeno: list of genotypic frequency matrices (matrix 1 x # genotypes). The list is constructed as follows: if the population is hermaphroditic it has only one element "ind", if the population is dioecious it has three elements, "female", "male" and "ind" which correspond respectively to the genotypic frequencies of the females, the males and the average of the two (assuming a sex ratio of 50:50).
  - freqHaplo: list of genotypic frequency matrices (matrix 1 x # haplotypes). The list is constructed in the same way as for genotypic frequencies (see above).
  - freqAlleles: list of allelic frequency matrices (matrix 1 x # alleles). The list is constructed in the same way as for genotypic frequencies (see above).

Details

This function is called each generation in each population of a simulation and systematically returns a list with the first element being a logic that indicates whether something should be saved. If so, the second element of this list will be saved.

By default the save nothing function, but it can be changed by the user as an argument in the simulate method of the Metapopulation class.

Author(s)

Ehouarn Le Faou
Population-class  

Description

The Population class allows for the collection of the parameters necessary to characterise a biological population. It is an essentially useful class in that no method associated with the population class can simulate its dynamics. To do this, it is necessary to use the Metapopulation class, which takes as input a list of populations (from one). The Population class is also used to check that each of these parameters is compatible with each other.

Details

Thus to build an object of class Ease, it is necessary to have defined an object Genome, as well as an object MutationMatrix and an object Selection (even if it is neutral, see setSelectNeutral).

Slots

- name  the name of the population.
- size  the size of the population.
- dioecy  logical indicating whether the population is dioecious or not (hermaphrodite).
- selfRate  the selfing rate of the population
- demography  logical indicating whether the population has stochastic demography (this does not include migration), i.e. non-constant size and potentially population growth or decay, depending on the situation it is in.
- growthRate  growth rate of the population.
- initGenoFreq  A row matrix of the size of the genotype number describing the initial allele frequencies common to all simulations
- genome  a Genome object
- initPopSize  initial population size, knowing that if the demography is extinct, the initial population size will automatically be set equal to the population size.
- selection  a Selection object
- mutMat  a MutationMatrix object

Author(s)

Ehouarn Le Faou
**Description**

Print method for the Genome class

**Usage**

```r
## S4 method for signature 'Genome'
print(x, ...)
```

**Arguments**

- `x`: a Genome object
- `...`: Ignored.

**Value**

No return value, only a display.

**Author(s)**

Ehouarn Le Faou

---

**Description**

Print method for the Metapopulation class

**Usage**

```r
## S4 method for signature 'Metapopulation'
print(x, ...)
```

**Arguments**

- `x`: a Metapopulation object
- `...`: Ignored.

**Author(s)**

Ehouarn Le Faou
print,MutationMatrix-method

*Print method for the MutationMatrix class*

**Description**

Print method for the MutationMatrix class

**Usage**

```r
## S4 method for signature 'MutationMatrix'
print(x, ...)
```

**Arguments**

- `x` a MutationMatrix object
- `...` there are no more parameters.

**Value**

No return value, only a display.

**Author(s)**

Ehouarn Le Faou

print,Population-method

*Print method for the Population class*

**Description**

Print method for the Population class

**Usage**

```r
## S4 method for signature 'Population'
print(x, ...)
```

**Arguments**

- `x` a Population object
- `...` the other parameter is `frame`, which is a logic indicating whether the frame surrounding the display of the population characteristics should be displayed or not.
print,Selection-method

Description
Print method for the Selection class

Usage
```r
## S4 method for signature 'Selection'
print(x, ...)
```

Arguments
- `x`: a Selection object
- `...`: there are no more parameters.

Value
No return value, only a display.

Author(s)
Ehouarn Le Faou

---

recombinationMatrix

Recombination matrix generation

Description
Generation of the recombination matrix associated to a Genome object.

Usage
```r
recombinationMatrix(genomeObj)
```

Arguments
- `genomeObj`: a Genome object
Details

A recombination matrix is a square matrix of size equal to the number of genotypes. It is a probability matrix in that the sum of the values in each row is equal to 1. For a given genotype, the row associated with it describes the probabilistic proportions that lead by recombination between diploid loci to the production of the other genotypes (and of itself if there are no mutations).

Value

A recombination matrix (probability matrix which associates to each genotype in a row the probability of recombining or not and of becoming another genotype or remaining the same).

Author(s)

Ehouarn Le Faou

---

**rowResultGen**

*Processing a result (or record) list*

---

Description

Processing a result (or record) list

Usage

`rowResultGen(x)`

Arguments

- **x**
  - list of result or record

Value

Merges the column names of the matrices making up the list with the names of the matrices, then merges the matrices together.

Author(s)

Ehouarn Le Faou
selectFormIntoVect  
**Conversion of selection formulas**

**Description**
Conversion of a list of selection formulas into a genotypic (or haplotypic) fitness vector associated with a Genome object.

**Usage**
```
selectFormIntoVect(selectFormula, genomeObj, haplo = FALSE)
```

**Arguments**
- `selectFormula`: a list of selection formulas
- `genomeObj`: a Genome object
- `haplo`: logical indicating whether the selection should apply to haplotypes (in the case of gametic selection for example)

**Value**
a vector of fitness values

**Author(s)**
Ehouarn Le Faou

selectInputTreatment  
**Treatment of selection formulas**

**Description**
Determines whether an entry for the selection is a list of selection formulas or just a vector. If it is a list of formulas, turns them into a vector. If it is a vector, does nothing.

**Usage**
```
selectInputTreatment(selectInput, genomeObj, haplo = FALSE)
```

**Arguments**
- `selectInput`: a selection input
- `genomeObj`: a Genome object
- `haplo`: logical indicating whether the selection should apply to haplotypes (in the case of gametic selection for example)
Value

a vector of fitness values

Author(s)

Ehouarn Le Faou

Summary

Selection-class

Class used to generate objects that manage the selection in the simulations.

Details

An object of type Selection is an object which describes the set of fitnesses which will be taken into account in the simulations. The selection according to these fitnesses can be applied at three levels: at the level of the individual, at the level of the production of gametes and at the level of the gametes themselves. Selection is therefore genotypic in the first two cases (each genotype is associated with a fitness value) and haplotypic in the third (each haplotype is associated with a fitness value).

Slots

genome a Genome object
IDhaplotypes IDs of haplotypes
IDgenotypes IDs of genotypes
IDgenome ID of the associated genome
nbHaplo the number of haplotypes
nbGeno the number of genotypes
gamFit the list of gametes’ fitness
indFit the list of individuals’ fitness
gamProdFit the list of gamete production fitness
sOnInds a logical indicating whether a selection on individuals has been configured by the user
sOnGams a logical indicating whether a selection on gametes has been configured by the user
sOnGamsProd a logical indicating whether a selection on gamete production has been configured by the user

Author(s)

Ehouarn Le Faou
selection.form.treatment

Treatment of a selection formula

Description
Conversion of the factors of a selection formula into the list of corresponding allelic combinations

Usage
selection.form.treatment(factors, genomeObj = NULL, checking = FALSE)

Arguments
- factors: formula factors (right-hand members)
- genomeObj: a Genome object for the test (see checking parameter)
- checking: logical indicating whether a test verifying the compatibility of input factors with the genome

Value
A list of vectors enumerating the allelic combinations that correspond to the factors

Author(s)
Ehouarn Le Faou

setGenome

Setting the genome

Description
Generation of a genome class object from the list of haploid loci and diploid loci. Each loci is defined by a factor vector that enumerates its alleles.

Usage
setGenome(listHapLoci = list(), listDipLoci = list(), recRate = numeric())

Arguments
- listHapLoci: a list of haploid loci
- listDipLoci: a list of diploid loci
- recRate: a two-by-two recombination rate vector
Details

A genome includes the list of all possible haplotypes and genotypes resulting from the combination of the alleles defined in input. As the Ease package was originally built for population genetics simulations including both diploid and haploid loci, it is necessary that both types of loci are defined. Despite this, the user can define only diploid or only haploid loci if they wish. If no diploid locus is defined, one is automatically generated with only one allele, thus not influencing the simulation. The same applies if no haploid locus is defined.

Each locus is described by a vector of factors which are the names of the possible alleles at that locus. All diploid (resp. haploid) loci thus defined are grouped in a list, called `listDipLoci` (resp. `listHapLoci`). Therefore, a Genome class object has two lists of loci defined in this way, one for diploid loci, one for haploid loci. The alleles and loci (diploid and haploid) must all have different names so that no ambiguity can persist.

If several are defined, the order of the diploid loci in the list is not trivial. The rates of two-to-one combinations between them must indeed be defined by the vector `recRate`. For example, if three diploid loci are defined, `recRate` must be of length 2, the first of its values defining the recombination rate between the first and second loci, the second of its values the recombination rate between the second and third loci. For example, if we want to define two groups of two loci that are linked to each other but are on two different chromosomes, we can define a `recRate = c(0.1, 0.5, 0.1)`.

Value

- a Genome object

Author(s)

Ehouarn Le Faou

Examples

```r
DL <- list(dl = c("A", "a"))
HL <- list(hl = c("B", "b"))
genomeObj <- setGenome(listHapLoci = HL, listDipLoci = DL)
```

Description

A metapopulation is a set of population(s) (from 1) that are simulated with potential migration between them. Only genotypes can migrate, i.e. adult individuals.
setMetapopulation

Usage

setMetapopulation(populations, migMat = matrix(1))

Arguments

populations a list of Population objects
migMat a migration matrix

Details

The construction of a Metapopulation object requires only two arguments (one optional). The first is a population(s) list, defined from the population class. The second is a migration matrix, which connects the populations together. This matrix is a probability matrix (square with the sum of the rows equal to 1, whose size is equal to the number of populations) where each value corresponds to the proportion of individuals (genotypes) that disperse from their source population (row) to their target population (column).

Value

a Metapopulation object

Author(s)

Ehouarn Le Faou

Examples

# Definition of a population in its simplest form:
DL <- list(dl = c("A", "a"))
HL <- list(hl = c("B", "b"))
mutations <- list(
  mutation(from = "A", to = "a", rate = 1e-3),
  mutation(from = "B", to = "b", rate = 1e-3)
)
genomeObj <- setGenome(listHapLoci = HL, listDipLoci = DL)
pop <- setPopulation(
  name = "A",
  size = 1000,
  dioecy = TRUE,
  genomeObj = genomeObj,
  selectionObj = setSelectNeutral(genomeObj),
  mutMatrixObj = setMutationMatrix(genomeObj, mutations = mutations)
)
metapop <- setMetapopulation(populations = list(pop))
metapop <- simulate(metapop, nsim = 10, seed = 123)
# Other examples available in the documentation of the package
setMutationMatrix

Setting the mutation matrix

Description

Generation of the mutation matrix associated with the genome given as input. A mutation matrix is used to simulate mutations that affect loci. An object of the class MutationMatrix does not only contain a (genotypic) mutation matrix. It also contains the attributes necessary for the construction and easy-to-read display of this matrix. The mutation matrix itself is a square matrix of size equal to the number of genotypes. It is a probability matrix in that the sum of the values in each row is equal to 1. For a given genotype, the row associated with it describes the probabilistic proportions that lead by mutation of this genotype to the production of the other genotypes (and of itself if there are no mutations).

Usage

setMutationMatrix(genomeObj, ...)

Arguments

- genomeObj: a Genome object
- ...: see details.

Details

There are three ways to define the mutation matrix associated with a Genome class object.

1) By giving two lists of allelic mutation matrices mutHapLoci and mutDipLoci, for haploid and diploid loci respectively. Each of these lists contains as many matrices as there are loci. These matrices are transition matrices (squares, with the sum of the rows equal to 1) of size equal to the number of alleles at the locus concerned.

2) By giving a forward and a backward allelic mutation rate (forwardMut and backwardMut respectively). The generated mutation matrices will thus be defined with the same rates for all loci. A forward mutation rate means that the transition from one allele to another is done in the order in which they were defined when the Genome class object was created, and in the other direction for the backward rate.

3) By giving a list of mutations generated through the mutation function.

Value

- a MutationMatrix object

Author(s)

Ehouarn Le Faou
Examples

### Example with two loci, each with two alleles ###

```r
# Definition of the genome
DL <- list(dl = c("A", "a"))
HL <- list(hl = c("B", "b"))
genomeObj <- setGenome(listHapLoci = HL, listDipLoci = DL)

# Three ways to define the same mutation matrix associated with the
genome defined above:

# 1) Mutation matrix from matrices
mutHapLoci <- list(matrix(c(0.99, 0.01, 0.01, 0.99), 2))
mutDipLoci <- list(matrix(c(0.99, 0.01, 0.01, 0.99), 2))
# One can then define the MutationMatrix class object:
setMutationMatrix(genomeObj,
    mutHapLoci = mutHapLoci,
    mutDipLoci = mutDipLoci
)

# 2) Mutation matrix from mutation rates
mutMatrixObj <- setMutationMatrix(genomeObj, forwardMut = 0.1)
# or by adding a backward mutation rate:
mutMatrixObj <- setMutationMatrix(genomeObj,
    forwardMut = 1e-3,
    backwardMut = 1e-4
)

# 3) Mutation matrix from single mutation definition
mutMatrixObj <- setMutationMatrix(genomeObj,
    mutations = list(
        mutation(from = "A", to = "a", rate = 0.1),
        mutation(from = "B", to = "b", rate = 0.1)
    )
)
```

---

**setPopulation**

**Setting a population**

**Description**

Generation of a population by providing all the necessary ingredients for its definition, including a genome, a mutation matrix and a selection regime.

**Usage**

```r
setPopulation(
    name,
```
setPopulation

size,
dioecy,
genomeObj,
mutMatrixObj,
selectionObj,
selfRate = 0,
demography = F,
growthRate = 0,
initPopSize = NULL,
initGenoFreq = NULL
}

Arguments

name the name of the population
size the population size
dioecy logical indicating whether the simulated population is dioecious or hermaphroditic
genomeObj a Genome object
mutMatrixObj a MutationMatrix object
selectionObj a Selection object
selfRate the selfing rate
demography a logic indicating whether the population should have a demography (stochasticity in the number of individuals present in the population + logistic growth with carrying capacity equal to the size parameter)
growthRate a Genome object
initPopSize the initial size of the population. It is necessarily equal to size if the population has no demography.
initGenoFreq a vector of the size of the genotype number describing the initial allele frequencies common to all simulations

Details

A population is defined strictly by a name, a size, a sexual system (dioecy or hermaphrodite), and the three objects defined previously: genome, mutation matrix and selection. In addition to that, it is possible to define - a selfing rate (by default equal to 0) - a vector of initial genotypic frequencies - a demography

Two demographic regimes are possible: no demography, i.e. a fixed population size, or demography, i.e. a population where the size fluctuates stochastically. The boolean argument ‘demography’ is used to define whether there should be stochasticity. For a fixed population size, it is therefore sufficient to define that ‘demography = FALSE’ (default) and to set the desired population size with the ‘popSize’ parameter.

For a fluctuating demography, ‘demography’ must be ‘TRUE’ and three other parameters are then needed: the initial population size (‘initPopSize’), the population growth rate (‘growthRate’) and the carrying capacity of the population (the population size, ‘popSize’).

It is also possible to avoid defining a population size altogether, by setting off the genetic drift (‘drift’ parameter). This will allow the model to be simulated deterministically.
**setSelectNeutral**

**Value**

a Population object

**Author(s)**

Ehouarn Le Faou

**Examples**

```r
# Definition of a population in its simplest form:
DL <- list(dl = c("A", "a"))
HL <- list(hl = c("B", "b"))
mutations <- list(
  mutation(from = "A", to = "a", rate = 1e-3),
  mutation(from = "B", to = "b", rate = 1e-3)
)
genoobj <- setGenome(listHapLoci = HL, listDipLoci = DL)
pop <- setPopulation(
  name = "A",
  size = 1000,
  dioecy = TRUE,
  genomeObj = genomeobj,
  selectionObj = setSelectNeutral(genomeObj),
  mutMatrixObj = setMutationMatrix(genomeObj, mutations = mutations)
)
```

---

**setSelectNeutral**  Setting the selection

**Description**

Generation of a neutral class Selection object. It can be used as a basis for adding selection layers with the setSelectOnInds, setSelectOnGametes or setSelectOnGametesProd functions, or if the model is neutral.

**Usage**

```r
setSelectNeutral(genomeObj)
```

**Arguments**

genomeObj a Genome object
Details

An object of type Selection is an object which describes the set of fitnesses which will be taken into account in the simulations. The selection according to these fitnesses can be applied at three levels: at the level of the individual, at the level of the production of gametes and at the level of the gametes themselves. Selection is therefore genotypic in the first two cases (each genotype is associated with a fitness value) and haplotypic in the third (each haplotype is associated with a fitness value).

Value

a Selection object

Author(s)

Ehouarn Le Faou

Examples

### Example with two loci, each with two alleles ###
```r
# Definition of the diploid locus
DL <- list(dl = c("A", "a"))
# Definition of the haploid locus
HL <- list(hl = c("b", "B"))
# Definition of the object of Genome class
genomeObj <- setGenome(listHapLoci = HL, listDipLoci = DL)
genomeObj
```

### Exemple with more diploid loci ###
```r
# Definition of the diploid loci
DL <- list(
  dl1 = c("A", "a"),
  dl2 = c("B", "b"),
  dl3 = c("C", "c")
)
# Definition of the haploid locus
HL <- list(hl = c("D", "d"))
# Definition of the object of Genome class, with in addition the necessary
definition of recombination rates between loci:
genomeObj <- setGenome(
  listHapLoci = HL, listDipLoci = DL,
  recRate = c(0.1, 0.5)
)
# Here we have a 0.1 recombination rate between dl1 and dl2 and a 0.5
# recombination rate between dl2 and dl3. It is as if dl1 and dl2 were linked,
# for example on the same chromosome, and that dl2 (and dl1 by consequence)
# and dl3 were independent, for example on different chromosomes.
genomeObj
```
setSelectOnGametes

Setting the selection on gametes

Description

Generation of an object of the Selection class which defines a selection among the individuals either by adding this type of selection to an already existing SelectionObj object (parameter selectionObj) or by creating one.

Usage

setSelectOnGametes(
  genomeObj = NULL,
  gamFit = c(),
  femaleFit = c(),
  maleFit = c(),
  selectionObj = NULL
)

Arguments

genomeObj a Genome object
gamFit an haplotypic fitness vector for all individuals
femaleFit an haplotypic fitness vector for females only
maleFit an haplotypic fitness vector for males only
selectionObj a Selection object (in the case where the selection on individuals is overlaid on an existing Selection object)

Value

a Selection object

Author(s)

Ehouarn Le Faou

Examples

DL <- list(dl = c("A", "a"))
HL <- list(hl = c("B", "b"))
genomeObj <- setGenome(listHapLoci = HL, listDipLoci = DL)
selectionObj <- setSelectOnGametes(
  genomeObj = genomeObj,
  gamFit = c(1, 1, 0.5, 0)
)
setSelectOnGametesProd

Setting the selection on gamete production

Description

Generation of an object of the Selection class which defines a selection on the gamete production either by adding this type of selection to an already existing SelectionObj object (parameter selectionObj) or by creating one.

Usage

```r
setSelectOnGametesProd(
  genomeObj = NULL,
  indProdFit = c(),
  femProdFit = c(),
  maleProdFit = c(),
  selectionObj = NULL
)
```

Arguments

- **genomeObj** a Genome object
- **indProdFit** a genotypic fitness vector for all individuals
- **femProdFit** a genotypic fitness vector for females only
- **maleProdFit** a genotypic fitness vector for males only
- **selectionObj** a Selection object (in the case where the selection on individuals is overlaid on an existing Selection object)

Value

a Selection object

Author(s)

Ehouarn Le Faou

Examples

```r
DL <- list(dl = c("A", "a"))
HL <- list(hl = c("B", "b"))
genomeObj <- setGenome(listHapLoci = HL, listDipLoci = DL)
selectionObj <- setSelectOnGametesProd(
  genomeObj = genomeObj,
  indProdFit = c(1, 1, 1, 1, 0.5, 0)
)
```
setSelectOnInds

Setting the selection on individuals

Description

Generation of an object of the Selection class which defines a selection among the individuals either by adding this type of selection to an already existing SelectionObj object (parameter selectionObj) or by creating one.

Usage

```r
setSelectOnInds(
  genomeObj = NULL,
  indFit = c(),
  femaleFit = c(),
  maleFit = c(),
  selectionObj = NULL
)
```

Arguments

- `genomeObj` a Genome object
- `indFit` a genotypic fitness vector for all individuals (whether or not they are hermaphrodite)
- `femaleFit` a genotypic fitness vector for females only (only if the population is dioecious)
- `maleFit` a genotypic fitness vector for males only (only if the population is dioecious)
- `selectionObj` a Selection object (in the case where the selection on individuals is overlaid on an existing Selection object)

Value

a Selection object

Author(s)

Ehouarn Le Faou

Examples

```r
DL <- list(dl = c("A", "a"))
HL <- list(hl = c("B", "b"))
genomeObj <- setGenome(listHapLoci = HL, listDipLoci = DL)
selectionObj <- setSelectOnInds(
  genomeObj = genomeObj,
  indFit = c(1, 1, 1, 1, 0.5, 0)
)
```
Description
Show method for the Genome class

Usage
## S4 method for signature 'Genome'
show(object)

Arguments
object a Genome object

Value
No return value, only a display.

Author(s)
Ehouarn Le Faou

Description
Show method for the Metapopulation class

Usage
## S4 method for signature 'Metapopulation'
show(object)

Arguments
object a Metapopulation object

Author(s)
Ehouarn Le Faou
### show,MutationMatrix-method

*Show method for the MutationMatrix class*

#### Description

Show method for the MutationMatrix class

#### Usage

```r
## S4 method for signature 'MutationMatrix'
show(object)
```

#### Arguments

- `object` a MutationMatrix object

#### Value

No return value, only a display.

#### Author(s)

Ehouarn Le Faou

---

### show,Population-method

*Show method for the Population class*

#### Description

Show method for the Population class

#### Usage

```r
## S4 method for signature 'Population'
show(object)
```

#### Arguments

- `object` a Population object

#### Author(s)

Ehouarn Le Faou
### show, Selection-method

*Show method for the Selection class*

**Description**

Show method for the Selection class

**Usage**

```r
## S4 method for signature 'Selection'
show(object)
```

**Arguments**

- `object`: a Selection object

**Value**

No return value, only a display.

**Author(s)**

Ehouarn Le Faou

---

### simulate, Metapopulation-method

*Simulate method for the Metapopulation class*

**Description**

Performing simulations of an Metapopulation object. The returned object is the same Metapopulation object completed with the results and records if they have been activated.

**Usage**

```r
## S4 method for signature 'Metapopulation'
simulate(
  object,
  nsim = 1,
  seed = NULL,
  threshold = 500,
  includefreqGeno = TRUE,
  recording = FALSE,
  recordGenGap = 1,
  drift = TRUE,
  includeParams = TRUE,
)```
simulate.Metapopulation-method

```
includeFitness = TRUE,
verbose = FALSE,
stopCondition = list(),
nameOutFunct = "outFunct"
)
```

Arguments

- `object`: a `Metapopulation` object
- `nsim`: the number of simulation to perform
- `seed`: the RNG seed to be fixed (allows exact reproduction of results)
- `threshold`: maximum duration of a simulation (in generations)
- `includeFreqGeno`: a logical indicating whether to include genotype frequencies in the results
- `recording`: a logical indicating whether to record all mutations, i.e. to record allelic and genotypic frequencies along the simulations
- `recordGenGap`: the number of generations between two records during simulation, if the record parameter is `TRUE`. Whatever the value of this parameter, both the first and the last generation will be included in the record
- `drift`: a logical indicating whether genetic drift should be considered (i.e. whether deterministic simulations are performed or not)
- `includeParams`: a logical indicating whether the parameters should be included in the result `data.frame` (can be useful when compiling multiple result tables)
- `includeFitness`: a logical indicating whether the mean fitness should be included in the result `data.frame` (can be useful when compiling multiple result tables)
- `verbose`: logical determining if the progress of the simulations should be displayed or not (useful in case of many simulations)
- `stopCondition`: list of vectors that each describe the allele(s) that must be fixed to define a stop condition. Each of these vectors will therefore be associated with a stop condition
- `nameOutFunct`: name of the custom output function. This function is called each generation in each population of a simulation and systematically returns a list with the first element being a logic that indicates whether something should be saved. If so, the second element of this list will be saved. If the `customOutFunct` parameter is null (default), there will be no custom output.

Value

An `Metapopulation` object from which we can now extract the results (or the records if `recording` = `TRUE`) with the `getResults` and `getRecords` functions.

Author(s)

Ehouarn Le Faou
whichHomoz

Which alleles are homozygous in the input?

Description
Determine which alleles are at least once input as homozygous in the formula.

Usage
whichHomoz(formula)

Arguments
formula a selection formula

Value
the enumeration of alleles that appear at least once homozygous

Author(s)
Ehouarn Le Faou
Index

alleleFreqMatGeneration, 3
areThereHomoz, 4

cat, 4, 5
catn, 4
check.genome, 5
check.metapopulation, 6
check.mutationMatrix, 6
check.population, 7
check.selection, 7

evaluateAlleleComb, 8

Genome-class, 8
genotyping, 9

getCustomOutput, 10
getRecords, 11
getResults, 11, 11

haploCrossMatrix, 12
haplotyping, 13

IDgenomeGeneration, 13
IDgenotypeGeneration, 14
IDhaplotypeGeneration, 15
initialize, Genome-method, 15
initialize, Metapopulation-method, 16
initialize, MutationMatrix-method, 17
initialize, Population-method, 17
initialize, Selection-method, 18
is.correct.transition.matrix, 19
is.default.matrix, 20
is.probability.matrix, 20
isAffected, 21
isHaploSelectFormula, 21

listing, 22

meiosisMatrix, 22
MTAPPO_SIMULATION, 24
Metapopulation-class, 23

mutation, 26, 28, 40
MutationMatrix-class, 27
mutMatFriendly, 28
mutMatRates, 28

outFunct, 29

Population-class, 30
print, Genome-method, 31
print, Metapopulation-method, 31
print, MutationMatrix-method, 32
print, Population-method, 32
print, Selection-method, 33

recombinationMatrix, 33
rowResultGen, 34

selectFormIntoVect, 35
selectInputTreatment, 35
Selection-class, 36
selection.form.treatment, 37
setGenome, 37
setMetapopulation, 38
setMutationMatrix, 26, 40
setPopulation, 41
setSelectNeutral, 30, 43
setSelectOnGametes, 45
setSelectOnGametesProd, 46
setSelectOnInds, 47
show, Genome-method, 48
show, Metapopulation-method, 48
show, MutationMatrix-method, 49
show, Population-method, 49
show, Selection-method, 50
simulate, Metapopulation-method, 50

whichHomoz, 52