Package 'BreedingSchemeLanguage'

October 22, 2018

Type Package

Title Describe and Simulate Breeding Schemes

Version 0.9.6

Date 2018-10-21

Maintainer Jean-Luc Jannink <jeanluc.work@gmail.com>

Description Users can simulate their planned breeding schemes by using functions that do standard breeding tasks. Yabe, Iwata, & Jannink (2017) <doi:10.2135/cropsci2016.06.0538>.

License GPL-3

Depends R (>= 3.0.0), snowfall, Rcpp

Imports ggplot2, lme4, Matrix, rrBLUP, stats

LinkingTo Rcpp

LazyLoad yes

VignetteBuilder knitr

Suggests knitr, rmarkdown

NeedsCompilation yes

RoxygenNote 6.1.0

Author Shiori Yabe [aut],
Jean-Luc Jannink [aut, cre],
Hiroyoshi Iwata [aut, ctb],
Liang Liming [cph] (Copyright holder of GENOME software. See ./COPYRIGHTS),
Abecasis Goncalo [cph] (Copyright holder of GENOME software. See ./COPYRIGHTS),
Nishimura Takuji [cph] (Copyright holder of Mersenne Twister code. See ./COPYRIGHTS),
Matsumoto Makoto [cph] (Copyright holder of Mersenne Twister code. See ./COPYRIGHTS)

Repository CRAN

Date/Publication 2018-10-22 14:10:03 UTC

R topics documented:

`addProgenyData` *Add progeny information to data after cross, doubledHaploid, or selfFertilize*

Description

Add progeny information to data after cross, doubledHaploid, or selfFertilize

Usage

```
addProgenyData(bsl, geno, pedigree)
```

Arguments

bsl	the list that has all the objects for one simulation
geno	the genotypes of the progeny
pedigree	the three-column pedigree of the progeny (last col: DH, outbred, self) Normally DH=-1, outbred=0, self=nGen of selfing but that is not currenly properly updated

Value

data with progeny information added

---

calcAmatrix *Calculate an additive relationship matrix*

Description

pedigreeToAmatrix returns an additive relationship matrix from a pedigree specified in three columns. The first column has to be the row number and sire and dam columns refer directly to rows of the pedigree.

Usage

```
calcAmatrix(pedColumns, aMatIn = NULL)
```

Arguments

pedColumns	A data.frame with four columns. The first column has to be the row number and sire and dam columns refer directly to rows of the pedigree. Parents of founders need to be set to 0 (ZERO). The row of a child has to be after (i.e. a higher row number) that of its parents. If an individual has one known and one unknown parent, set the unknown parent to 0. The fourth column indicates: If negative, the individual is a DH: the F1 is created, generates a gamete, which is then doubled. If positive, the number of generations an individual was self-pollinated after it's F1 ancestor was created (can be 0 if the individual is the F1).
aMatIn	A square matrix that contains the additive relationship matrix between individuals at the beginning of the pedigree. If given, the function saves time by starting calculations after those individuals This aMatIn functionality is NOT compatible with calculating A inverse

Details

pedigreeToAmatrix has some functionality useful for plants. It can handle a pedigree of doubled haploid lines. Individuals can be self-pollinated for an arbitrary number of generations.

Value

A matrix, aMat, the additive relationship matrix

---

calcGenotypicValue *calcGenotypicValue*

Description

calcGenotypicValue

Usage

```
calcGenotypicValue(geno, mapData)
```

Arguments

geno	matrix of haplotypes
mapData	map data

---

calcPhenotypicValue *calcPhenotypicValue*

Description

calcPhenotypicValue

Usage

```
calcPhenotypicValue(gv, nRep, errorVar)
```

Arguments

gv	genotypic values
nRep	how many replications of phenotypic values: used for multiple locations and/or years
errorVar	error variance

cross *Cross with random mating, or equal contributions, or randomly between two populations*

Description

Cross with random mating, or equal contributions, or randomly between two populations

Usage

```
cross(sEnv = NULL, nProgeny = 100, equalContribution = F,
  popID = NULL, popID2 = NULL, notWithinFam = F, pedigree = NULL)
```

Arguments

sEnv	the environment that BSL functions operate in. If NULL, the default simEnv is attempted
nProgeny	the number of progenies. Default: 100
equalContribution	if T all individuals used the same number of times as parents, if F individuals chosen at random to be parents. Default: F
popID	population ID to be crossed. Default: the last population
popID2	population ID to be crossed with popID to make hybrids. Default: second population not used
notWithinFam	if TRUE, like equalContribution, all individuals are used the same number of times as parents and self-fertilization is not allowed. In addition, half- and full-sibs are not allowed to mate. Default: F
pedigree	optional two- or three-column matrix: the first two columns are the GIDs that you want to cross, the third column is the number of progeny from that cross. NOTE: pedigree supersedes the nProgeny, equalContribution, popID, popID2, and notWithinFam parameters. You have to know what you are doing to use this parameter. Default: NULL

Value

modifies the list sims in environment sEnv by creating a progeny population as specified, with an incremented population number

See Also

defineSpecies for an example

defineCosts — *Define the costs that go into breeding Default for some costs is zero because they probably belong to fixed costs*

Description

Define the costs that go into breeding Default for some costs is zero because they probably belong to fixed costs

Usage

```
defineCosts(sEnv = NULL, phenoCost = NULL, genoCost = 0.25,
  crossCost = 1, selfCost = 1, doubHapCost = 5, predCost = 0,
  selectCost = 0, locCost = 0, yearCost = 0)
```

Arguments

sEnv	the environment that BSL functions operate in. Default is "simEnv" so use that to avoid specifying when calling functions
phenoCost	named vector: names are the plot types and contents are the cost for one plot of that type (default: Standard=1; if plotTypes have been defined in defineVariances and phenoCost is not specified, all plotTypes are given a cost of 1)
genoCost	scalar: cost to genotype one individual default (0.25)
crossCost	scalar: cost of creating a new individual from a cross (1)
selfCost	scalar: the cost of creating a selfed seed (1)
doubHapCost	scalar: the cost of creating a doubled haploid seed (5)
predCost	scalar: the cost of running the analysis to make predictions (0)
selectCost	scalar: the cost of running the analysis to do selection (0)
locCost	scalar: the cost of maintaining a location for a year (0)
yearCost	scalar: the cost of program upkeep for a year (0)

Value

modifies the list sims in environment sEnv by adding cost parameters allowing the total cost of the simulated scheme to be calculated

`defineSpecies` *Define and create species*

Description

Define and create species

Usage

```
defineSpecies(loadData = NULL, importFounderHap = NULL,
  saveDataFileName = NULL, nSim = 1, nCore = 1, nChr = 7,
  lengthChr = 150, effPopSize = 100, nMarkers = 1000, nQTL = 50,
  propDomi = 0, nEpiLoci = 0, domModel = "HetHom")
```

Arguments

`loadData`	if null create a new species (default), else the file name of previously created species like "species1_1" (Do not put "RData" extension). A path can be specified, like "simDirectory/species1_1" (in which case "simDirectory" must exist).
`importFounderHap`	if null create new founder haplotypes (default), else the file name of externally sourced founder haplotypes in hapmap format (see Manual). If founder haplotypes are loaded, the ‘nMarkers‘ parameter is superseded by the number of loci in those haplotypes.
`saveDataFileName`	if not NULL, the name to save the species data, like "species1_1". A path can be specified, like "simDirectory/species1_1" (in which case "simDirectory" must exist). Default: NULL
`nSim`	the number of simulation trials
`nCore`	simulation processed in parallel over this number of CPUs (Check computer capacity before setting above 1.)
`nChr`	the number of chromosomes
`lengthChr`	the length of each chromosome (cM. all chromosomes are the same length)
`effPopSize`	the effective population size in the base population
`nMarkers`	the number of markers, which is used especially for genomic selection
`nQTL`	the number of QTLs controlling the target trait
`propDomi`	the probability of dominant QTL among the all QTL
`nEpiLoci`	the expected number of epistatic loci for each effect
`domModel`	the dominance model: "HetHom" means homozygotes have equal effect but opposite to that of heterozygotes, "Partial": zero means ancestral dominant over derived, one means derived dominant over ancestral, any value in between means partial dominance. At the moment, functionality for the "Partial" option is only available when using ImportFounderHap.

Value

An environment that contains a list sims with each object of the list being one replicate to initiate a simulation

Examples

```
if (exists("simEnv")){
rm(list=names(simEnv), envir=simEnv)
rm(simEnv)
}
simEnv <- defineSpecies(nSim=2, nChr=5, lengthChr=100, effPopSize=20, nMarkers=100, nQTL=10)
initializePopulation(nInd=50) # popID 0 created
phenotype()
select(nSelect=20) # popID 1 selected out of popID 0
cross() # popID 2 created
phenotype(nRep=2)
select(nSelect=5) # popID 3 selected out of popID 2
cross() # popID 4 created
plotData()
```

---

defineVariances *Define genetic, interaction, and error variances*

Description

Define genetic, interaction, and error variances

Usage

```
defineVariances(sEnv = NULL, gVariance = 1, locCorrelations = NULL,
  gByLocVar = 1, gByYearVar = 1, fracGxEAdd = 1,
  plotTypeErrVars = c(Standard = 1))
```

Arguments

sEnv	the environment that BSL functions operate in. Default is "simEnv" so use that to avoid specifying when calling functions
gVariance	genetic variance in the initial population Default is 1
locCorrelations	matrix: genetic correlation in performance between locations default: If given, the genetic co-variance across locations is gVariance * locCorrelations. If NULL, deviations with variance gByLocVar will be added to the main genotypic effect to determine the genotypic value in each location. The expected genetic correlation between locations is then gVariance / (gVariance + gByLocVar). Default is NULL
gByLocVar	scalar: the genotype by location variance Default is 1, BUT if locCorrelations given, this parameter is not used)

gByYearVar	scalar: the genotype by year variance Default is 1
fracGxEAdd	scalar: for GxL and GxY what fraction of the effect is additive versus non-additive Default is 1
plotTypeErrVars	named vector: names are the plot types and contents are the error variances associated with them (default: Standard=1)

Value

modifies the list sims in environment sEnv by adding parameters that determine genetic, location, year, and error variances

---

DH *DH*

Description

DH

Usage

```
DH(genoParent, pos)
```

Arguments

genoParent	matrix of haplotypes
pos	position of markers/QTLs

---

doubledHaploid *Doubled haploids*

Description

Doubled haploids

Usage

```
doubledHaploid(sEnv = NULL, nProgeny = 100, popID = NULL)
```

Arguments

sEnv	the environment that BSL functions operate in. Default is "simEnv" so use that to avoid specifying when calling functions
nProgeny	the number of progeny
popID	population ID to be devided by meiosis and doubled (default: the latest population)

Value

modifies the list sims in environment sEnv by creating a doubled haploid progeny population as specified, with an incremented population number

---

`genotype` *Genotype markers*

Description

Genotype markers

Usage

```
genotype(sEnv = NULL, popID = NULL)
```

Arguments

`sEnv`	the environment that BSL functions operate in. Default is "simEnv" so use that to avoid specifying when calling functions
`popID`	population ID to be genotyped (default: all populations)

Value

modifies the list sims in environment sEnv by indicating that genotypes of individuals in popID are available for breeding tasks

---

`getCoalescentSim` *getCoalescentSim*

Description

getCoalescentSim

Usage

```
getCoalescentSim(nPopsSamples = NULL, effPopSize = 100, nChr = 7,
  nPiecesPerChr = 15000, recBTpieces = 1e-04, nMrkOrMut = 100,
  minMAF = 0.01, seed = as.integer(Sys.time()), tree = 0)
```

Arguments

nPopsSamples	-pop[2] (GENOME)
effPopSize	-N (GENOME)
nChr	-c (GENOME)
nPiecesPerChr	-pieces (GENOME)
recBTpieces	-rec (GENOME)
nMrkOrMut	parameter to calculate -s (GENOME)
minMAF	-maf (GENOME)
seed	-seed (GENOME)
tree	-tree (GENOME)

---

initializePopulation *Create a founder population*

Description

Create a founder population

Usage

```
initializePopulation(sEnv = NULL, nInd = 100,
  founderHapsInDiploids = F)
```

Arguments

sEnv	the environment that BSL functions operate in. If NULL, the default simEnv is attempted
nInd	population size (default:100)
founderHapsInDiploids	set to TRUE if you have uploaded phased diploid data in defineSpecies

Value

modifies the list sims in environment sEnv by creating a founder population

See Also

defineSpecies for an example

makeDHs *makeDHs*

Description

makeDHs

Usage

```
makeDHs(popSize, geno, pos)
```

Arguments

popSize	the number of DH individuals to return
geno	matrix of haplotypes
pos	position of markers/QTLs

makeGamete *makeGamete*

Description

makeGamete

Usage

```
makeGamete(geno, pos)
```

Arguments

geno	matrix of haplotypes
pos	position of markers/QTLs

makeMap *create map and QTL effects*

Description

create map and QTL effects

Usage

```
makeMap(map, nLoci, nMarkers, nQTL, propDomi, interactionMean,
  qtlInfo = NULL)
```

Arguments

map	map information (Chromosome and Position)
nLoci	the number of markers and QTL
nMarkers	the number of markers, which is used especially for genomic selection
nQTL	the number of QTLs controlling the target trait
propDomi	the probability of dominant QTL among the all QTL
interactionMean	the expected number of epistatic loci for each effect
qtlInfo	possible to give the function all of the qtl information

Value

map data including which loci are primary QTL and which are modifying loci, which have dominance effects, the effect sizes

makeProgenies *makeProgenies*

Description

makeProgenies

Usage

```
makeProgenies(parents, geno, pos)
```

Arguments

parents	ID of haplotypes that the parents harbor
geno	matrix of haplotypes
pos	position of markers/QTLs

makeProgeny *makeProgeny*

Description

makeProgeny

Usage

```
makeProgeny(genoMat, genoPat, pos)
```

Arguments

genoMat	matrix of maternal haplotype
genoPat	matrix of paternal haplotype
pos	position of markers/QTLs

makeSelfs *makeSelfs*

Description

makeSelfs

Usage

```
makeSelfs(popSize, geno, pos)
```

Arguments

popSize	the number of selfed individuals to return
geno	matrix of haplotypes
pos	position of markers/QTLs

`outputResults` *Save the results*

Description

Save the results

Usage

```
outputResults(sEnv = NULL, summarize = T, saveDataFileName = NULL)
```

Arguments

`sEnv`	the environment that BSL functions operate in. Default is "simEnv" so use that to avoid specifying when calling functions
`summarize`	If TRUE a matrix with breeding cycles in rows and replications in columns, with the first set of columns being cycle means and the second set of columns cycle variances. If FALSE a list as long as the number of replications, with each list element containing a list of all simulation objects.
`saveDataFileName`	NULL or string of the file name to save the simulated data, like "result1_1". A path can be specified, like "simDirectory/result1_1" (in which case "simDirectory" must exist). Default: NULL.

Value

If saveDataFileName is NULL, data is return as an object, else data is saved as a ".rds" file. To examine the data, use the readRDS function.

`pedigreeMate` *pedigreeMate*

Description

pedigreeMate

Usage

```
pedigreeMate(parents, geno, pos)
```

Arguments

`parents`	two- or three-column matrix: first two columns index the parents you want to cross, if there is a third column, it is the number of progeny from that pair of parents
`geno`	matrix of haplotypes
`pos`	position of markers/QTLs

phasedHapMap2mat — *Transform a data.frame with a hapmap data in it into a marker dosage and map list*

Description

Transform a data.frame with a hapmap data in it into a marker dosage and map list

Usage

```
phasedHapMap2mat(hm)
```

Arguments

hm	The data.frame that has HapMap data in it

Value

list with markers and map objects

phenotype — *Evaluate the phenotypic value*

Description

Evaluate the phenotypic value

Usage

```
phenotype(sEnv = NULL, plotType = "Standard", nRep = 1,
  popID = NULL, locations = 1, years = NULL)
```

Arguments

sEnv	the environment that BSL functions operate in. Default is "simEnv" so use that to avoid specifying when calling functions
plotType	links to previously defined error variance and cost (default: "Standard")
nRep	scalar: the number of replications per trial (i.e., year x loc combination) (default: 1)
popID	population ID to be evaluated (default: the latest population)
locations	integer vector of the locations where phenotyping occurs (e.g., c(1, 3) to phenotype at locations 1 and 3. Default: 1, phenotype at the first location)
years	integer vector of the years when phenotyping occurs (e.g., 1:2 to phenotype during the first two years of the breeding scheme. Default: the last year among previous phenotyping. NOTE: thus, to phenotype in a new [the next] year, specify the next year number [e.g., if past phenotyping was in years 1 & 2, specify 3]).

Value

modifies the list sims in environment sEnv by generating phenotypes for the specified popID, locations, and years.

See Also

defineSpecies for an example

---

plotData *Plot the results*

Description

Plot the results

Usage

```
plotData(sEnv = NULL, ymax = NULL, add = FALSE,
  addDataFileName = NULL, popID = NULL, suppress = F)
```

Arguments

sEnv	the environment that BSL functions operate in. Default is "simEnv" so use that to avoid specifying when calling functions
ymax	the maximum value of y-axis (default: the maximun value in the data)
add	if TRUE a new result will be added to an existing plot, using data loaded from addDataFileName. if FALSE a new plot will be drawn (default)
addDataFileName	if not NULL, the name to save the summarized data for the next simulation, like "plot1_1". A path can be specified, like "simDirectory/plot1_1" (in which case "simDirectory" must exist). Default: NULL
popID	list of vectors of the population IDs you want plotted
suppress	if TRUE, this call is just to assemble the data to be plotted in a later call to plotData (default: FALSE)

Value

A matrix with the plot data

See Also

defineSpecies for an example

predGameteMeanVar *predGameteMeanVar*

Description

predGameteMeanVar

Usage

```
predGameteMeanVar(geno, pos, locEff)
```

Arguments

geno	2 x nLoc matrix of haplotypes, coded -1 and 1
pos	position of markers/QTLs
locEff	effects of each locus

predictValue *Genomic prediction*

Description

Genomic prediction

Usage

```
predictValue(sEnv = NULL, popID = NULL, trainingPopID = NULL,
  locations = NULL, years = NULL, sharingInfo = NULL)
```

Arguments

sEnv	the environment that BSL functions operate in. If NULL, the default simEnv is attempted
popID	population ID to be predicted (default: the latest population)
trainingPopID	population ID to be used for training a prediction model (default: all populations with phenotype data). NOTE: if sharingInfo="none" this parameter has no effect.
locations	data from which locations should be used (default: all locations)
years	data from which years should be used (default: all years)
sharingInfo	one of "none", "markers", "pedigree". If none, genotypic values are assumed IID. If markers or pedigree, a genomic or pedigree relationship matrix is constructed

Value

modifies the list sims in environment sEnv by calculating predicted values as specified and changing the default selection criterion to use them

---

`randomMate` *randomMate*

Description

randomMate

Usage

```
randomMate(popSize, geno, pos)
```

Arguments

`popSize`	the number of progeny to return
`geno`	matrix of haplotypes
`pos`	position of markers/QTLs

`randomMateAll` *randomMateAll*

Description

randomMateAll

Usage

```
randomMateAll(popSize, geno, pos)
```

Arguments

`popSize`	the number of progeny to return
`geno`	matrix of haplotypes
`pos`	position of markers/QTLs

randomMateNoFam *randomMateNoFam*

Description

randomMateNoFam

Usage

```
randomMateNoFam(popSize, geno, pos, genoRec)
```

Arguments

popSize	the number of progeny to return
geno	matrix of haplotypes
pos	position of markers/QTLs
genoRec	pedigree records of the individuals represented by geno

select *Select individuals*

Description

Select individuals

Usage

```
select(sEnv = NULL, nSelect = 40, popID = NULL, random = F,
  type = "Mass")
```

Arguments

sEnv	the environment that BSL functions operate in. If NULL, the default simEnv is attempted
nSelect	the number of selected individuals (default: 40)
popID	population ID to be selected (default: When random=T, the last population. When random=F, it is the last evaluated population)
random	assuming random selection or selection according to estimated value (T: random selection, F: selection for high value)
type	"WithinFamily" or "Mass" (default: Mass). If Mass, all individuals are ranked against each other and the highest nSelect are taken. If WithinFamily, individuals are ranked within paternal half-sib (if population was randomly mated) or full-sib (if population from selfFertilize or doubledHaploid) families. If random=T, then mass or within-family selection is random. NOTE: breeding scheme budget may not be correct for within-family selection because then nSelect is per-family but the number of families varies across simulations.

Value

modifies the list sims in environment sEnv by selecting individuals of the specified popID with the default selection criterion and giving those individuals a new popID

See Also

`defineSpecies` for an example

`selfFertilize` *Self-fertilize*

Description

Self-fertilize

Usage

```
selfFertilize(sEnv = NULL, nProgeny = 100, popID = NULL)
```

Arguments

`sEnv`	the environment that BSL functions operate in. Default is "simEnv" so use that to avoid specifying when calling functions
`nProgeny`	the number of progeny
`popID`	population ID to be self-fertilized (default: the latest population)

Value

modifies the list sims in environment sEnv by creating a selfed progeny population as specified, with an incremented population number

---

`simHapMap` *Generate a data.frame with a hapmap data in it to test phasedHapMap2mat*

Description

Generate a data.frame with a hapmap data in it to test phasedHapMap2mat

Usage

```
simHapMap(nInd = 200, nMrk = 1050, nChr = 7, lenChr = 150,
  maf = stats::runif(nMrk), nCharCode = 2, nQTL = NULL,
  propDomi = NULL, interactionMean = NULL, varEffects = NULL)
```

Arguments

nInd	The number of individuals with marker data
nMrk	The number of markers
nChr	The number of chromosomes the species has
lenChr	The length of the chromosomes (all equal) in cM
maf	The desired minor allele frequency of each marker
nCharCode	Whether the genotype codes should be one or two characters
nQTL	If you want to put QTL information in the hapmap
propDomi	proportion of QTL that act dominantly
interactionMean	average number of loci interacting with QTL: Poisson
varEffects	variance of the QTL effects

Value

A data.frame with hapmap data in it

---

testParameterOptimality

Function to return the optimality of a parameter vector for a breeding scheme given a simulation environment

---

Description

Function to return the optimality of a parameter vector for a breeding scheme given a simulation environment

Usage

```
testParameterOptimality(sEnv = NULL, schemeFileName, parmList,
  objectiveFunc, budget = 1000)
```

Arguments

sEnv	the environment that BSL functions operate in. This environment should be fresh from defineSpecies, defineVariances, and defineCosts. If NULL, the default simEnv is attempted
schemeFileName	source file that holds the script of the breeding scheme. The scheme should return a value to be maximized in a variable called objective. If you want to use a non-default simulation environment, you need to plan for that in the schemeFile by giving sEnv as the first parameter in all of the BSL functions
parmList	(preferably) named list with values of parameters characterizing the breeding scheme
objectiveFunc	a function that can be applied to a BSL simulation environment to return a value showing the realized performance of the breeding scheme
budget	the maximum budget that is allowed for the breeding scheme

Value

Two-object list: `objective` is the objective function value of the breeding scheme given the simulation environment and the parameter vector and `totalCost` is the budget used by the breeding scheme. If the parameter vector causes the scheme to exceed the given budget, `objective`=NA

Index