Package ‘AlphaSimR’

August 18, 2024

Type Package

Title Breeding Program Simulations

Version 1.6.0

Date 2024-08-15

Description The successor to the 'AlphaSim' software for breeding program simulation [Faux et al. (2016) <doi:10.3835/plantgenome2016.02.0013>]. Used for stochastic simulations of breeding programs to the level of DNA sequence for every individual. Contained is a wide range of functions for modeling common tasks in a breeding program, such as selection and crossing. These functions allow for constructing simulations of highly complex plant and animal breeding programs via scripting in the R software environment. Such simulations can be used to evaluate overall breeding program performance and conduct research into breeding program design, such as implementation of genomic selection. Included is the 'Markovian Coalescent Simulator' ('MaCS') for fast simulation of biallelic sequences according to a population demographic history [Chen et al. (2009) <doi:10.1101/gr.083634.108>].

License MIT + file LICENSE

URL `https://github.com/gaynorr/AlphaSimR`,
`https://gaynorr.github.io/AlphaSimR/`,
`https://www.edx.org/learn/animal-breeding/the-university-of-edinburgh-breeding-programme-modelling-with-alphasimr`

Encoding UTF-8

Depends R (>= 4.0.0), methods, R6

Imports Rcpp (>= 0.12.7), Rdpack

RdMacros Rdpack

LinkingTo Rcpp, RcppArmadillo (>= 0.7.500.0.0), BH

RoxygenNote 7.3.2

Suggests knitr, rmarkdown, testthat

VignetteBuilder knitr

NeedsCompilation yes

Author Chris Gaynor [aut, cre] (<https://orcid.org/0000-0003-0558-6656>),
Gregor Gorjanc [ctb] (<https://orcid.org/0000-0001-8008-2787>),
John Hickey [ctb] (<https://orcid.org/0000-0001-5675-3974>),
Daniel Money [ctb] (<https://orcid.org/0000-0001-5151-3648>),
David Wilson [ctb],
Thiago Oliveira [ctb] (<https://orcid.org/0000-0002-4555-2584>),
Audrey Martin [ctb] (<https://orcid.org/0000-0003-2235-0098>),
Philip Greenspoon [ctb] (<https://orcid.org/0000-0001-6284-7248>)

Maintainer Chris Gaynor <gaynor.robert@hotmail.com>

Repository CRAN

Date/Publication 2024-08-17 22:20:13 UTC

Contents

`.newPop` *Create new population (internal)*

Description

Creates a new `Pop-class` from an object of of the Pop superclass.

Usage

```
.newPop(
  rawPop,
  id = NULL,
  mother = NULL,
  father = NULL,
  iMother = NULL,
  iFather = NULL,
  isDH = NULL,
  femaleParentPop = NULL,
  maleParentPop = NULL,
  hist = NULL,
  simParam = NULL,
  ...
)
```

Arguments

`rawPop`	an object of the pop superclass
`id`	optional id for new individuals
`mother`	optional id for mothers
`father`	optional id for fathers
`iMother`	optional internal id for mothers
`iFather`	optional internal id for fathers
`isDH`	optional indicator for DH/inbred individuals
`femaleParentPop`	optional population of female parents
`maleParentPop`	optional population of male parents
`hist`	optional recombination history
`simParam`	an object of `SimParam`
`...`	additional arguments passed to the finalizePop function in simParam

Value

Returns an object of `Pop-class`

`aa` *Additive-by-additive epistatic deviations*

Description

Returns additive-by-additive epistatic deviations for all traits

Usage

```
aa(pop, simParam = NULL)
```

Arguments

`pop`	an object of `Pop-class`
`simParam`	an object of `SimParam`

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
SP$addTraitAD(10, meanDD=0.5)
SP$setVarE(h2=0.5)


#Create population
pop = newPop(founderPop, simParam=SP)
aa(pop, simParam=SP)
```

`addSegSite` *Add segregating site to MapPop*

Description

This function allows for adding a new segregating site with user supplied genotypes to a MapPop. The position of the site is set using a genetic map position.

Usage

```
addSegSite(mapPop, siteName, chr, mapPos, haplo)
```

Arguments

`mapPop`	an object of `MapPop-class`
`siteName`	name to give the segregating site
`chr`	which chromosome to add the site
`mapPos`	genetic map position of site in Morgans
`haplo`	haplotypes for the site

Value

an object of `MapPop-class`

Examples

```
# Creates a populations of 10 outbred individuals
# Their genome consists of 1 chromosome and 2 segregating sites
founderPop = quickHaplo(nInd=10,nChr=1,segSites=2)

# Add a locus a the 0.5 Morgan map position
haplo = matrix(sample(x=0:1, size=20, replace=TRUE), ncol=1)

founderPop2 = addSegSite(founderPop, siteName="x", chr=1, mapPos=0.5, haplo=haplo)

pullSegSiteHaplo(founderPop2)
```

`attrition` *Lose individuals at random*

Description

Samples individuals at random to remove from the population. The user supplies a probability for the individuals to be removed from the population.

Usage

```
attrition(pop, p)
```

Arguments

`pop`	an object of `Pop-class`
`p`	the expected proportion of individuals that will be lost to attrition.

Value

an object of `Pop-class`

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=100, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)


#Create population
pop = newPop(founderPop, simParam=SP)

#Lose an expected 5% of individuals
pop = attrition(pop, p=0.05)
```

---

bv *Breeding value*

Description

Returns breeding values for all traits

Usage

```
bv(pop, simParam = NULL)
```

Arguments

`pop`	an object of `Pop-class`
`simParam`	an object of `SimParam`

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
SP$addTraitAD(10, meanDD=0.5)
SP$setVarE(h2=0.5)


#Create population
pop = newPop(founderPop, simParam=SP)
bv(pop, simParam=SP)
```

`calcGCA` *Calculate GCA*

Description

Calculate general combining ability of test crosses. Intended for output from hybridCross using the "testcross" option, but will work for any population.

Usage

```
calcGCA(pop, use = "pheno")
```

Arguments

`pop`	an object of `Pop-class` or `HybridPop-class`
`use`	tabulate either genetic values "gv", estimated breeding values "ebv", or phenotypes "pheno"

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10, inbred=TRUE)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitA(10)

#Create population
pop = newPop(founderPop, simParam=SP)

#Make crosses for full diallele
pop2 = hybridCross(pop, pop, simParam=SP)
GCA = calcGCA(pop2, use="gv")
```

`cChr` *Combine MapPop chromosomes*

Description

Merges the chromosomes of multiple `MapPop-class` or `NamedMapPop-class` objects. Each MapPop must have the same number of chromosomes

Usage

```
cChr(...)
```

Arguments

`...`	`MapPop-class` or `NamedMapPop-class` objects to be combined

Value

Returns an object of `MapPop-class`

Examples

```
pop1 = quickHaplo(nInd=10, nChr=1, segSites=10)
pop2 = quickHaplo(nInd=10, nChr=1, segSites=10)

combinedPop = cChr(pop1, pop2)
```

`dd` *Dominance deviations*

Description

Returns dominance deviations for all traits

Usage

```
dd(pop, simParam = NULL)
```

Arguments

`pop`	an object of `Pop-class`
`simParam`	an object of `SimParam`

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
SP$addTraitAD(10, meanDD=0.5)
SP$setVarE(h2=0.5)


#Create population
pop = newPop(founderPop, simParam=SP)
dd(pop, simParam=SP)
```

`doubleGenome` *Double the ploidy of individuals*

Description

Creates new individuals with twice the ploidy. This function was created to model the formation of tetraploid potatoes from diploid potatoes. This function will work on any population.

Usage

```
doubleGenome(pop, keepParents = TRUE, simParam = NULL)
```

Arguments

`pop`	an object of 'Pop' superclass
`keepParents`	should previous parents be used for mother and father.
`simParam`	an object of 'SimParam' class

Value

Returns an object of `Pop-class`

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=2, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)


#Create population
pop = newPop(founderPop, simParam=SP)

#Create individuals with doubled ploidy
pop2 = doubleGenome(pop, simParam=SP)
```

`ebv` *Estimated breeding value*

Description

A wrapper for accessing the ebv slot

Usage

```
ebv(pop)
```

Arguments

`pop`	a `Pop-class` or similar object

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitAD(10, meanDD=0.5)
SP$setVarE(h2=0.5)

#Create population
pop = newPop(founderPop, simParam=SP)
pop@ebv = matrix(rnorm(pop@nInd), nrow=pop@nInd, ncol=1)
ebv(pop)
```

`editGenome` *Edit genome*

Description

Edits selected loci of selected individuals to a homozygous state for either the 1 or 0 allele. The gv slot is recalculated to reflect the any changes due to editing, but other slots remain the same.

Usage

```
editGenome(pop, ind, chr, segSites, allele, simParam = NULL)
```

Arguments

`pop`	an object of `Pop-class`
`ind`	a vector of individuals to edit
`chr`	a vector of chromosomes to edit. Length must match length of segSites.
`segSites`	a vector of segregating sites to edit. Length must match length of chr.
`allele`	either 0 or 1 for desired allele
`simParam`	an object of `SimParam`

Value

Returns an object of `Pop-class`

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=2, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitA(10)

#Create population
pop = newPop(founderPop, simParam=SP)

#Change individual 1 to homozygous for the 1 allele
#at locus 1, chromosome 1
pop2 = editGenome(pop, ind=1, chr=1, segSites=1,
                  allele=1, simParam=SP)
```

`editGenomeTopQtl` *Edit genome - the top QTL*

Description

Edits the top QTL (with the largest additive effect) to a homozygous state for the allele increasing. Only nonfixed QTL are edited The gv slot is recalculated to reflect the any changes due to editing, but other slots remain the same.

Usage

```
editGenomeTopQtl(pop, ind, nQtl, trait = 1, increase = TRUE, simParam = NULL)
```

Arguments

`pop`	an object of `Pop-class`
`ind`	a vector of individuals to edit
`nQtl`	number of QTL to edit
`trait`	which trait effects should guide selection of the top QTL
`increase`	should the trait value be increased or decreased
`simParam`	an object of `SimParam`

Value

Returns an object of `Pop-class`

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=2, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitA(10)

#Create population
pop = newPop(founderPop, simParam=SP)

#Change up to 10 loci for individual 1
pop2 = editGenomeTopQtl(pop, ind=1, nQtl=10, simParam=SP)
```

`fastRRBLUP` *Fast RR-BLUP*

Description

Solves an RR-BLUP model for genomic predictions given known variance components. This implementation is meant as a fast and low memory alternative to `RRBLUP` or `RRBLUP2`. Unlike the those functions, the fastRRBLUP does not fit fixed effects (other than the intercept) or account for unequal replication.

Usage

```
fastRRBLUP(
  pop,
  traits = 1,
  use = "pheno",
  snpChip = 1,
  useQtl = FALSE,
  maxIter = 1000,
  Vu = NULL,
  Ve = NULL,
  simParam = NULL,
  ...
)
```

Arguments

`pop`	a `Pop-class` to serve as the training population
`traits`	an integer indicating the trait to model, a trait name, or a function of the traits returning a single value. Only univariate models are supported.
`use`	train model using phenotypes "pheno", genetic values "gv", estimated breeding values "ebv", breeding values "bv", or randomly "rand"

`snpChip`	an integer indicating which SNP chip genotype to use
`useQtl`	should QTL genotypes be used instead of a SNP chip. If TRUE, snpChip specifies which trait's QTL to use, and thus these QTL may not match the QTL underlying the phenotype supplied in traits.
`maxIter`	maximum number of iterations.
`Vu`	marker effect variance. If value is NULL, a reasonable value is chosen automatically.
`Ve`	error variance. If value is NULL, a reasonable value is chosen automatically.
`simParam`	an object of `SimParam`
`...`	additional arguments if using a function for traits

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=20)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitA(10)
SP$setVarE(h2=0.5)
SP$addSnpChip(10)

#Create population
pop = newPop(founderPop, simParam=SP)

#Run GS model and set EBV
ans = fastRRBLUP(pop, simParam=SP)
pop = setEBV(pop, ans, simParam=SP)

#Evaluate accuracy
cor(gv(pop), ebv(pop))
```

`genicVarA` *Additive genic variance*

Description

Returns additive genic variance for all traits

Usage

```
genicVarA(pop, simParam = NULL)
```

Arguments

`pop`	an object of `Pop-class`
`simParam`	an object of `SimParam`

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
SP$addTraitAD(10, meanDD=0.5)
SP$setVarE(h2=0.5)


#Create population
pop = newPop(founderPop, simParam=SP)
genicVarA(pop, simParam=SP)
```

`genicVarAA` *Additive-by-additive genic variance*

Description

Returns additive-by-additive epistatic genic variance for all traits

Usage

```
genicVarAA(pop, simParam = NULL)
```

Arguments

`pop`	an object of `Pop-class`
`simParam`	an object of `SimParam`

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
SP$addTraitAD(10, meanDD=0.5)
SP$setVarE(h2=0.5)


#Create population
```

```
pop = newPop(founderPop, simParam=SP)
genicVarAA(pop, simParam=SP)
```

---

genicVarD *Dominance genic variance*

Description

Returns dominance genic variance for all traits

Usage

```
genicVarD(pop, simParam = NULL)
```

Arguments

`pop`	an object of `Pop-class`
`simParam`	an object of `SimParam`

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
SP$addTraitAD(10, meanDD=0.5)
SP$setVarE(h2=0.5)


#Create population
pop = newPop(founderPop, simParam=SP)
genicVarD(pop, simParam=SP)
```

---

genicVarG *Total genic variance*

Description

Returns total genic variance for all traits

Usage

```
genicVarG(pop, simParam = NULL)
```

Arguments

`pop`	an object of `Pop-class`
`simParam`	an object of `SimParam`

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
SP$addTraitAD(10, meanDD=0.5)
SP$setVarE(h2=0.5)


#Create population
pop = newPop(founderPop, simParam=SP)
genicVarG(pop, simParam=SP)
```

`genParam` *Sumarize genetic parameters*

Description

Calculates genetic and genic additive and dominance variances for an object of `Pop-class`

Usage

```
genParam(pop, simParam = NULL)
```

Arguments

`pop`	an object of `Pop-class`
`simParam`	an object of `SimParam`

Value

varA an nTrait by nTrait matrix of additive genetic variances

varD an nTrait by nTrait matrix of dominance genetic variances

varAA an nTrait by nTrait matrix of additive-by-additive genetic variances

varG an nTrait by nTrait matrix of total genetic variances

genicVarA an nTrait vector of additive genic variances

genicVarD an nTrait vector of dominance genic variances

genicVarAA an nTrait vector of additive-by-additive genic variances

genicVarG an nTrait vector of total genic variances

covA_HW an nTrait vector of additive covariances due to non-random mating

covD_HW an nTrait vector of dominance covariances due to non-random mating

covAA_HW an nTrait vector of additive-by-additive covariances due to non-random mating

covG_HW an nTrait vector of total genic covariances due to non-random mating

covA_L an nTrait vector of additive covariances due to linkage disequilibrium

covD_L an nTrait vector of dominance covariances due to linkage disequilibrium

covAA_L an nTrait vector of additive-by-additive covariances due to linkage disequilibrium

covAD_L an nTrait vector of additive by dominance covariances due to linkage disequilibrium

covAAA_L an nTrait vector of additive by additive-by-additive covariances due to linkage disequilibrium

covDAA_L an nTrait vector of dominance by additive-by-additive covariances due to linkage disequilibrium

covG_L an nTrait vector of total genic covariances due to linkage disequilibrium

mu an nTrait vector of trait means

mu_HW an nTrait vector of expected trait means under random mating

gv a matrix of genetic values with dimensions nInd by nTraits

bv a matrix of breeding values with dimensions nInd by nTraits

dd a matrix of dominance deviations with dimensions nInd by nTraits

aa a matrix of additive-by-additive epistatic deviations with dimensions nInd by nTraits

gv_mu an nTrait vector of intercepts with dimensions nInd by nTraits

gv_a a matrix of additive genetic values with dimensions nInd by nTraits

gv_d a matrix of dominance genetic values with dimensions nInd by nTraits

gv_aa a matrix of additive-by-additive genetic values with dimensions nInd by nTraits

alpha a list of average allele subsitution effects with length nTraits

alpha_HW a list of average allele subsitution effects at Hardy-Weinberg equilibrium with length nTraits

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
SP$addTraitAD(10, meanDD=0.5)
SP$setVarE(h2=0.5)


#Create population
pop = newPop(founderPop, simParam=SP)
ans = genParam(pop, simParam=SP)
```

`getGenMap` *Get genetic map*

Description

Retrieves the genetic map for all loci.

Usage

```
getGenMap(object = NULL, sex = "A")
```

Arguments

`object`	where to retrieve the genetic map. Can be an object of `SimParam` or `MapPop-class`. If NULL, the function will look for a SimParam object called "SP" in your global environment.
`sex`	determines which sex specific map is returned. Options are "A" for average map, "F" for female map, and "M" for male map. All options are equivalent if not using sex specific maps or using pulling from a MapPop.

Value

Returns a data.frame with:

id Unique identifier for locus

chr Chromosome containing the locus

pos Genetic map position

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
getGenMap(founderPop)
```

`getNumThreads` *Number of available threads*

Description

Gets the number of available threads by calling the OpenMP function `omp_get_max_threads()`

Usage

```
getNumThreads()
```

Value

integer

Examples

```
getNumThreads()
```

`getPed` *Get pedigree*

Description

Returns the population's pedigree as stored in the id, mother and father slots. NULL is returned if the input population lacks the required.

Usage

```
getPed(pop)
```

Arguments

`pop`	a population

Examples

```
# Create a founder population
founderPop = quickHaplo(2,1,2)

# Set simulation parameters
SP = SimParam$new(founderPop)

# Create a population
pop = newPop(founderPop, simParam=SP)
```

```
# Get the pedigree
getPed(pop)

# Returns NULL when a population lacks a pedigree
getPed(founderPop)
```

`getQtlMap` *Get QTL genetic map*

Description

Retrieves the genetic map for the QTL of a given trait.

Usage

```
getQtlMap(trait = 1, sex = "A", simParam = NULL)
```

Arguments

`trait`	an integer for the
`sex`	determines which sex specific map is returned. Options are "A" for average map, "F" for female map, and "M" for male map. All options are equivalent if not using sex specific maps.
`simParam`	an object of `SimParam`

Value

Returns a data.frame with:

id Unique identifier for the QTL

chr Chromosome containing the QTL

site Segregating site on the chromosome

pos Genetic map position

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitA(5)

#Pull SNP map
getQtlMap(trait=1, simParam=SP)
```

`getSnpMap` *Get SNP genetic map*

Description

Retrieves the genetic map for a given SNP chip.

Usage

```
getSnpMap(snpChip = 1, sex = "A", simParam = NULL)
```

Arguments

`snpChip`	an integer. Indicates which SNP chip's map to retrieve.
`sex`	determines which sex specific map is returned. Options are "A" for average map, "F" for female map, and "M" for male map. All options are equivalent if not using sex specific maps.
`simParam`	an object of `SimParam`

Value

Returns a data.frame with:

id Unique identifier for the SNP

chr Chromosome containing the SNP

site Segregating site on the chromosome

pos Genetic map position

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addSnpChip(5)

#Pull SNP map
getSnpMap(snpChip=1, simParam=SP)
```

`gv` *Genetic value*

Description

A wrapper for accessing the gv slot

Usage

```
gv(pop)
```

Arguments

`pop`	a `Pop-class` or similar object

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
SP$addTraitAD(10, meanDD=0.5)
SP$setVarE(h2=0.5)


#Create population
pop = newPop(founderPop, simParam=SP)
gv(pop)
```

`hybridCross` *Hybrid crossing*

Description

A convenient function for hybrid plant breeding simulations. Allows for easy specification of a test cross scheme and/or creation of an object of `HybridPop-class`. Note that the `HybridPop-class` should only be used if the parents were created using the `makeDH` function or `newPop` using inbred founders. The id for new individuals is [mother_id]_[father_id]

Usage

```
hybridCross(
  females,
  males,
  crossPlan = "testcross",
  returnHybridPop = FALSE,
  simParam = NULL
)
```

Arguments

`females`	female population, an object of `Pop-class`
`males`	male population, an object of `Pop-class`
`crossPlan`	either "testcross" for all possible combinations or a matrix with two columns for designed crosses
`returnHybridPop`	should results be returned as `HybridPop-class`. If false returns results as `Pop-class`. Population must be fully inbred if TRUE.
`simParam`	an object of `SimParam`

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=2, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)


#Create population
pop = newPop(founderPop, simParam=SP)

#Make crosses for full diallele
pop2 = hybridCross(pop, pop, simParam=SP)
```

`HybridPop-class` *Hybrid population*

Description

A lightweight version of `Pop-class` for hybrid lines. Memory is saved by not storing genotypic data.

Usage

```
## S4 method for signature 'HybridPop'
x[i]

## S4 method for signature 'HybridPop'
c(x, ...)

isHybridPop(x)
```

Arguments

`x`	a 'HybridPop'
`i`	index of individuals
`...`	additional 'HybridPop' objects

Methods (by generic)

- `[`: Extract HybridPop using index or id
- `c(HybridPop)`: Combine multiple HybridPops

Functions

- `isHybridPop()`: Test if object is of a HybridPop class

Slots

`nInd` number of individuals

`id` an individual's identifier

`mother` the identifier of the individual's mother

`father` the identifier of the individual's father

`nTraits` number of traits

`gv` matrix of genetic values. When using GxE traits, gv reflects gv when p=0.5. Dimensions are nInd by nTraits.

`pheno` matrix of phenotypic values. Dimensions are nInd by nTraits.

`gxe` list containing GxE slopes for GxE traits

`importGenMap` *Import genetic map*

Description

Formats a genetic map stored in a data.frame to AlphaSimR's internal format. Map positions must be in Morgans.

Usage

```
importGenMap(genMap)
```

Arguments

`genMap`	genetic map as a data.frame. The first three columns must be: marker name, chromosome, and map position (Morgans). Marker name and chromosome are coerced using as.character.

Value

a list of named vectors

Examples

```
genMap = data.frame(markerName=letters[1:5],
                    chromosome=c(1,1,1,2,2),
                    position=c(0,0.5,1,0.15,0.4))

asrMap = importGenMap(genMap=genMap)

str(asrMap)
```

`importHaplo` *Import haplotypes*

Description

Formats haplotype in a matrix format to an AlphaSimR population that can be used to initialize a simulation. This function serves as wrapper for `newMapPop` that utilizes a more user friendly input format.

Usage

```
importHaplo(haplo, genMap, ploidy = 2L, ped = NULL)
```

Arguments

`haplo`	a matrix of haplotypes
`genMap`	genetic map as a data.frame. The first three columns must be: marker name, chromosome, and map position (Morgans). Marker name and chromosome are coerced using as.character. See `importGenMap`
`ploidy`	ploidy level of the organism
`ped`	an optional pedigree for the supplied genotypes. See details.

Details

The optional pedigree can be a data.frame, matrix or a vector. If the object is a data.frame or matrix, the first three columns must include information in the following order: id, mother, and father. All values are coerced using as.character. If the object is a vector, it is assumed to only include the id. In this case, the mother and father will be set to "0" for all individuals.

Value

a `MapPop-class` if ped is NULL, otherwise a `NamedMapPop-class`

Examples

```
haplo = rbind(c(1,1,0,1,0),
              c(1,1,0,1,0),
              c(0,1,1,0,0),
              c(0,1,1,0,0))
colnames(haplo) = letters[1:5]

genMap = data.frame(markerName=letters[1:5],
                    chromosome=c(1,1,1,2,2),
                    position=c(0,0.5,1,0.15,0.4))

ped = data.frame(id=c("a","b"),
                 mother=c(0,0),
                 father=c(0,0))

founderPop = importHaplo(haplo=haplo,
                         genMap=genMap,
                         ploidy=2L,
                         ped=ped)
```

`importInbredGeno` *Import inbred, diploid genotypes*

Description

Formats the genotypes from inbred, diploid lines to an AlphaSimR population that can be used to initialize a simulation. An attempt is made to automatically detect 0,1,2 or -1,0,1 genotype coding. Heterozygotes or probabilistic genotypes are allowed, but will be coerced to the nearest homozygote. Pedigree information is optional and when provided will be passed to the population for easier identification in the simulation.

Usage

```
importInbredGeno(geno, genMap, ped = NULL)
```

Arguments

`geno`	a matrix of genotypes
`genMap`	genetic map as a data.frame. The first three columns must be: marker name, chromosome, and map position (Morgans). Marker name and chromosome are coerced using as.character. See importGenMap
`ped`	an optional pedigree for the supplied genotypes. See details.

Details

The optional pedigree can be a data.frame, matrix or a vector. If the object is a data.frame or matrix, the first three columns must include information in the following order: id, mother, and father. All values are coerced using as.character. If the object is a vector, it is assumed to only include the id. In this case, the mother and father will be set to "0" for all individuals.

Value

a `MapPop-class` if ped is NULL, otherwise a `NamedMapPop-class`

Examples

```
geno = rbind(c(2,2,0,2,0),
             c(0,2,2,0,0))
colnames(geno) = letters[1:5]

genMap = data.frame(markerName=letters[1:5],
                    chromosome=c(1,1,1,2,2),
                    position=c(0,0.5,1,0.15,0.4))

ped = data.frame(id=c("a","b"),
                 mother=c(0,0),
                 father=c(0,0))

founderPop = importInbredGeno(geno=geno,
                              genMap=genMap,
                              ped=ped)
```

`isFemale` *Test if individuals of a population are female or male*

Description

Test if individuals of a population are female or male

Usage

```
isFemale(x)

isMale(x)
```

Arguments

x	Pop-class

Value

logical

Functions

- `isMale()`: Test if individuals of a population are female or male

Examples

```
founderGenomes <- quickHaplo(nInd = 3, nChr = 1, segSites = 100)
SP <- SimParam$new(founderGenomes)
SP$setSexes(sexes = "yes_sys")
pop <- newPop(founderGenomes)

isFemale(pop)
isMale(pop)

pop[isFemale(pop)]
pop[isFemale(pop)]@sex
```

---

isPop *Test if object is of a Population class*

Description

Utilify function to test if object is of a Population class

Usage

```
isPop(x)
```

Arguments

x	Pop-class

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=2, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
SP$addTraitA(10)

#Create population
pop = newPop(founderPop, simParam=SP)
isPop(pop)
isPop(SP)
```

LociMap-class *Loci metadata*

Description

used for both SNPs and QTLs

Slots

`nLoci`	total number of loci
`lociPerChr`	number of loci per chromosome
`lociLoc`	physical position of loci
`name`	optional name for LociMap object

`makeCross` *Make designed crosses*

Description

Makes crosses within a population using a user supplied crossing plan.

Usage

```
makeCross(pop, crossPlan, nProgeny = 1, simParam = NULL)
```

Arguments

`pop`	an object of `Pop-class`
`crossPlan`	a matrix with two column representing female and male parents. Either integers for the position in population or character strings for the IDs.
`nProgeny`	number of progeny per cross
`simParam`	an object of `SimParam`

Value

Returns an object of `Pop-class`

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)


#Create population
pop = newPop(founderPop, simParam=SP)

#Cross individual 1 with individual 10
crossPlan = matrix(c(1,10), nrow=1, ncol=2)
pop2 = makeCross(pop, crossPlan, simParam=SP)
```

`makeCross2` *Make designed crosses*

Description

Makes crosses between two populations using a user supplied crossing plan.

Usage

```
makeCross2(females, males, crossPlan, nProgeny = 1, simParam = NULL)
```

Arguments

`females`	an object of `Pop-class` for female parents.
`males`	an object of `Pop-class` for male parents.
`crossPlan`	a matrix with two column representing female and male parents. Either integers for the position in population or character strings for the IDs.
`nProgeny`	number of progeny per cross
`simParam`	an object of `SimParam`

Value

Returns an object of `Pop-class`

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)


#Create population
pop = newPop(founderPop, simParam=SP)

#Cross individual 1 with individual 10
crossPlan = matrix(c(1,10), nrow=1, ncol=2)
pop2 = makeCross2(pop, pop, crossPlan, simParam=SP)
```

`makeDH` *Generates DH lines*

Description

Creates DH lines from each individual in a population. Only works with diploid individuals. For polyploids, use `reduceGenome` and `doubleGenome`.

Usage

```
makeDH(pop, nDH = 1, useFemale = TRUE, keepParents = TRUE, simParam = NULL)
```

Arguments

`pop`	an object of 'Pop' superclass
`nDH`	total number of DH lines per individual
`useFemale`	should female recombination rates be used.
`keepParents`	should previous parents be used for mother and father.
`simParam`	an object of 'SimParam' class

Value

Returns an object of `Pop-class`

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=2, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)


#Create population
pop = newPop(founderPop, simParam=SP)

#Create 1 DH for each individual
pop2 = makeDH(pop, simParam=SP)
```

`MapPop-class` *Raw population with genetic map*

Description

Extends `RawPop-class` to add a genetic map. This is the first object created in a simulation. It is used for creating initial populations and setting traits in the `SimParam`.

Usage

```
## S4 method for signature 'MapPop'
x[i]

## S4 method for signature 'MapPop'
c(x, ...)

isMapPop(x)
```

Arguments

`x`	a ’MapPop’ object
`i`	index of individuals
`...`	additional ’MapPop’ objects

Methods (by generic)

- `[`: Extract MapPop by index
- `c(MapPop)`: Combine multiple MapPops

Functions

- `isMapPop()`: Test if object is of a MapPop class

Slots

`genMap` list of chromosome genetic maps

`centromere` vector of centromere positions

`inbred` indicates whether the individuals are fully inbred

`meanEBV` *Mean estimated breeding values*

Description

Returns the mean estimated breeding values for all traits

Usage

```
meanEBV(pop)
```

Arguments

`pop`	an object of `Pop-class` or `HybridPop-class`

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
SP$addTraitA(10)
trtH2 = 0.5
SP$setVarE(h2=trtH2)


#Create population
pop = newPop(founderPop, simParam=SP)
pop@ebv = trtH2 * (pop@pheno - meanP(pop)) #ind performance based EBV
meanEBV(pop)
```

`meanG` *Mean genetic values*

Description

Returns the mean genetic values for all traits

Usage

```
meanG(pop)
```

Arguments

`pop`	an object of `Pop-class` or `HybridPop-class`

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
SP$addTraitA(10)
SP$setVarE(h2=0.5)


#Create population
pop = newPop(founderPop, simParam=SP)
meanG(pop)
```

`meanP` *Mean phenotypic values*

Description

Returns the mean phenotypic values for all traits

Usage

```
meanP(pop)
```

Arguments

`pop`	an object of `Pop-class` or `HybridPop-class`

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
SP$addTraitA(10)
SP$setVarE(h2=0.5)


#Create population
pop = newPop(founderPop, simParam=SP)
meanP(pop)
```

`mergeGenome` *Combine genomes of individuals*

Description

This function is designed to model the pairing of gametes. The male and female individuals are treated as gametes, so the ploidy of newly created individuals will be the sum of it parents.

Usage

```
mergeGenome(females, males, crossPlan, simParam = NULL)
```

Arguments

`females`	an object of `Pop-class` for female parents.
`males`	an object of `Pop-class` for male parents.
`crossPlan`	a matrix with two column representing female and male parents. Either integers for the position in population or character strings for the IDs.
`simParam`	an object of `SimParam`

Value

Returns an object of `Pop-class`

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)


#Create population
pop = newPop(founderPop, simParam=SP)

#Cross individual 1 with individual 10
crossPlan = matrix(c(1,10), nrow=1, ncol=2)
pop2 = mergeGenome(pop, pop, crossPlan, simParam=SP)
```

`mergePops` *Merge list of populations*

Description

Rapidly merges a list of populations into a single population

Usage

```
mergePops(popList)
```

Arguments

`popList`	a list containing `Pop-class` elements or a `MultiPop-class`

Value

Returns a `Pop-class`

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)


#Create a list of populations and merge list
pop = newPop(founderPop, simParam=SP)
pop@misc$tmp = rnorm(n=10)
pop@misc$tmp2 = rnorm(n=10)

popList = list(pop, pop)
pop2 = mergePops(popList)
```

`MultiPop-class` *Multi-Population*

Description

The mega-population represents a population of populations. It is designed to behave like a list of populations.

Usage

```
## S4 method for signature 'MultiPop'
x[i]

## S4 method for signature 'MultiPop'
x[[i]]

## S4 method for signature 'MultiPop'
c(x, ...)

## S4 method for signature 'MultiPop'
length(x)

isMultiPop(x)
```

Arguments

`x`	a 'MultiPop' object
`i`	index of populations or mega-populations
`...`	additional 'MultiPop' or 'Pop' objects

Methods (by generic)

- `[`: Extract MultiPop by index
- `[[`: Extract Pop by index
- `c(MultiPop)`: Combine multiple MultiPops
- `length(MultiPop)`: Number of pops in MultiPop

Functions

- `isMultiPop()`: Test if object is of a MultiPop class

Slots

`pops` list of `Pop-class` and/or `MultiPop-class`

`mutate` *Add Random Mutations*

Description

Adds random mutations to individuals in a population. Note that any existing phenotypes or EBVs are kept. Thus, the user will need to run `setPheno` and/or `setEBV` to generate new phenotypes or EBVs that reflect changes introduced by the new mutations.

Usage

```
mutate(pop, mutRate = 2.5e-08, returnPos = FALSE, simParam = NULL)
```

Arguments

`pop`	an object of `Pop-class`
`mutRate`	rate of new mutations
`returnPos`	should the positions of mutations be returned
`simParam`	an object of `SimParam`

Value

an object of `Pop-class` if returnPos=FALSE or a list containing a `Pop-class` and a data.frame containing the postions of mutations if returnPos=TRUE

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=2, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitA(10)

#Create population
pop = newPop(founderPop, simParam=SP)

#Introduce mutations
pop = mutate(pop, simParam=SP)
```

`NamedMapPop-class` *Raw population with genetic map and id*

Description

Extends `MapPop-class` to add id, mother and father.

Usage

```
## S4 method for signature 'NamedMapPop'
x[i]

## S4 method for signature 'NamedMapPop'
c(x, ...)

isNamedMapPop(x)
```

Arguments

`x`	a 'NamedMapPop' object
`i`	index of individuals
`...`	additional 'NamedMapPop' objects

Methods (by generic)

- `[`: Extract NamedMapPop by index
- `c(NamedMapPop)`: Combine multiple NamedMapPops

Functions

- `isNamedMapPop()`: Test if object is a NamedMapPop class

Slots

`id` an individual's identifier

`mother` the identifier of the individual's mother

`father` the identifier of the individual's father

`newEmptyPop` *Creates an empty population*

Description

Creates an empty `Pop-class` object with user defined ploidy and other parameters taken from simParam.

Usage

```
newEmptyPop(ploidy = 2L, simParam = NULL)
```

Arguments

`ploidy`	the ploidy of the population
`simParam`	an object of `SimParam`

Value

Returns an object of `Pop-class` with zero individuals

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=2, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
SP$addTraitA(10)

#Create empty population
pop = newEmptyPop(simParam=SP)
isPop(pop)
```

`newMapPop` *New MapPop*

Description

Creates a new `MapPop-class` from user supplied genetic maps and haplotypes.

Usage

```
newMapPop(genMap, haplotypes, inbred = FALSE, ploidy = 2L)
```

Arguments

`genMap`	a list of genetic maps
`haplotypes`	a list of matrices or data.frames that can be coerced to matrices. See details.
`inbred`	are individuals fully inbred
`ploidy`	ploidy level of the organism

Details

Each item of genMap must be a vector of ordered genetic lengths in Morgans. The first value must be zero. The length of the vector determines the number of segregating sites on the chromosome.

Each item of haplotypes must be coercible to a matrix. The columns of this matrix correspond to segregating sites. The number of rows must match the number of individuals times the ploidy if using inbred=FALSE. If using inbred=TRUE, the number of rows must equal the number of individuals. The haplotypes can be stored as numeric, integer or raw. The underlying C++ function will use raw.

Value

an object of `MapPop-class`

Examples

```
# Create genetic map for two chromosomes, each 1 Morgan long
# Each chromosome contains 11 equally spaced segregating sites
genMap = list(seq(0,1,length.out=11),
              seq(0,1,length.out=11))

# Create haplotypes for 10 outbred individuals
chr1 = sample(x=0:1,size=20*11,replace=TRUE)
chr1 = matrix(chr1,nrow=20,ncol=11)
chr2 = sample(x=0:1,size=20*11,replace=TRUE)
chr2 = matrix(chr2,nrow=20,ncol=11)
haplotypes = list(chr1,chr2)

founderPop = newMapPop(genMap=genMap, haplotypes=haplotypes)
```

`newMultiPop` *Create new Multi Population*

Description

Creates a new `MultiPop-class` from one or more `Pop-class` and/or `MultiPop-class` objects.

Usage

```
newMultiPop(...)
```

Arguments

`...`	one or more `Pop-class` and/or `MultiPop-class` objects.

Value

Returns an object of `MultiPop-class`

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=2, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
SP$addTraitA(10)

#Create population
pop = newPop(founderPop, simParam=SP)
megaPop = newMultiPop(pop=pop)
isMultiPop(megaPop)
```

`newPop` *Create new population*

Description

Creates an initial `Pop-class` from an object of `MapPop-class` or `NamedMapPop-class`. The function is intended for use with output from functions such as `runMacs`, `newMapPop`, or `quickHaplo`.

Usage

```
newPop(rawPop, simParam = NULL, ...)
```

Arguments

`rawPop`	an object of `MapPop-class` or `NamedMapPop-class`
`simParam`	an object of `SimParam`
`...`	additional arguments used internally

Details

Note that `newPop` takes genomes from the `rawPop` and uses them without recombination! Hence, if you call `newPop(rawPop = founderGenomes)` twice, you will get two sets of individuals with different id but the same genomes. To get genetically different sets of individuals you can subset the `rawPop` input, say first half for one set and the second half for the other set.

Value

Returns an object of `Pop-class`

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=2, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
SP$addTraitA(10)

#Create population
pop = newPop(founderPop, simParam=SP)
isPop(pop)

#Misc
pop@misc$tmp1 = rnorm(n=2)
pop@misc$tmp2 = rnorm(n=2)

#MiscPop
pop@miscPop$tmp1 = sum(pop@misc$tmp1)
pop@miscPop$tmp2 = sum(pop@misc$tmp2)
```

`nInd` *Number of individuals*

Description

A wrapper for accessing the nInd slot

Usage

```
nInd(pop)
```

Arguments

pop	a `Pop-class` or similar object

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitAD(10, meanDD=0.5)
SP$setVarE(h2=0.5)

#Create population
pop = newPop(founderPop, simParam=SP)
nInd(pop)
```

`pedigreeCross` *Pedigree cross*

Description

Creates a `Pop-class` from a generic pedigree and a set of founder individuals.

The way in which the user supplied pedigree is used depends on the value of matchID. If matchID is TRUE, the IDs in the user supplied pedigree are matched against founderNames. If matchID is FALSE, founder individuals in the user supplied pedigree are randomly sampled from founderPop.

Usage

```
pedigreeCross(
  founderPop,
  id,
  mother,
  father,
  matchID = FALSE,
  maxCycle = 100,
  DH = NULL,
  nSelf = NULL,
  useFemale = TRUE,
  simParam = NULL
)
```

Arguments

`founderPop`	a `Pop-class`
`id`	a vector of unique identifiers for individuals in the pedigree. The values of these IDs are seperate from the IDs in the founderPop if matchID=FALSE.
`mother`	a vector of identifiers for the mothers of individuals in the pedigree. Must match one of the elements in the id vector or they will be treated as unknown.
`father`	a vector of identifiers for the fathers of individuals in the pedigree. Must match one of the elements in the id vector or they will be treated as unknown.
`matchID`	indicates if the IDs in founderPop should be matched to the id argument. See details.
`maxCycle`	the maximum number of loops to make over the pedigree to sort it.
`DH`	an optional vector indicating if an individual should be made a doubled haploid.
`nSelf`	an optional vector indicating how many generations an individual should be selfed.
`useFemale`	If creating DH lines, should female recombination rates be used. This parameter has no effect if, recombRatio=1.
`simParam`	an object of 'SimParam' class

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=2, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)


#Create population
pop = newPop(founderPop, simParam=SP)

#Pedigree for a biparental cross with 7 generations of selfing
id = 1:10
```

```
mother = c(0,0,1,3:9)
father = c(0,0,2,3:9)
pop2 = pedigreeCross(pop, id, mother, father, simParam=SP)
```

`pheno` *Phenotype*

Description

A wrapper for accessing the pheno slot

Usage

```
pheno(pop)
```

Arguments

`pop`	a `Pop-class` or similar object

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
SP$addTraitAD(10, meanDD=0.5)
SP$setVarE(h2=0.5)


#Create population
pop = newPop(founderPop, simParam=SP)
pheno(pop)
```

`Pop-class` *Population*

Description

Extends `RawPop-class` to add sex, genetic values, phenotypes, and pedigrees.

Usage

```
## S4 method for signature 'Pop'
x[i]

## S4 method for signature 'Pop'
c(x, ...)

## S4 method for signature 'Pop'
show(object)

## S4 method for signature 'Pop'
length(x)
```

Arguments

`x`	a 'Pop' object
`i`	index of individuals
`...`	additional 'Pop' objects
`object`	a 'Pop' object

Methods (by generic)

- `[`: Extract Pop by index or id
- `c(Pop)`: Combine multiple Pops
- `show(Pop)`: Show population summary
- `length(Pop)`: Number of individuals in Pop (the same as nInd())

Slots

`id` an individual's identifier

`iid` an individual's internal identifier

`mother` the identifier of the individual's mother

`father` the identifier of the individual's father

`sex` sex of individuals: "M" for males, "F" for females, and "H" for hermaphrodites

`nTraits` number of traits

`gv` matrix of genetic values. When using GxE traits, gv reflects gv when p=0.5. Dimensions are nInd by nTraits.

`pheno` matrix of phenotypic values. Dimensions are nInd by nTraits.

`ebv` matrix of estimated breeding values. Dimensions are nInd rows and a variable number of columns.

`gxe` list containing GxE slopes for GxE traits

`fixEff` a fixed effect relating to the phenotype. Used by genomic selection models but otherwise ignored.

`misc` a list whose elements correspond to additional miscellaneous nodes with the items for individuals in the population (see example in `newPop`). This list is normally empty and exists solely as an open slot available for uses to store extra information about individuals.

`miscPop` a list of any length containing optional meta data for the population (see example in `newPop`). This list is empty unless information is supplied by the user. Note that the list is emptied every time the population is subsetted or combined because the meta data for old population might not be valid anymore.

See Also

`newPop`, `newEmptyPop`, `resetPop`

`popVar` *Population variance*

Description

Calculates the population variance matrix as opposed to the sample variance matrix calculated by `var`. i.e. divides by n instead of n-1

Usage

```
popVar(X)
```

Arguments

`X`	an n by m matrix

Value

an m by m variance-covariance matrix

`pullIbdHaplo` *Pull IBD haplotypes*

Description

Retrieves IBD haplotype data

Usage

```
pullIbdHaplo(pop, chr = NULL, snpChip = NULL, simParam = NULL)
```

Arguments

`pop`	an object of `Pop-class`
`chr`	a vector of chromosomes to retrieve. If NULL, all chromosomes are retrieved.
`snpChip`	an integer indicating which SNP array loci are to be retrieved. If NULL, all sites are retrieved.
`simParam`	an object of `SimParam`

Value

Returns a matrix of IBD haplotypes.

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=15)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitA(10)
SP$addSnpChip(5)
SP$setTrackRec(TRUE)

#Create population
pop = newPop(founderPop, simParam=SP)
pullIbdHaplo(pop, simParam=SP)
```

`pullMarkerGeno` *Pull marker genotypes*

Description

Retrieves genotype data for user specified loci

Usage

```
pullMarkerGeno(pop, markers, asRaw = FALSE, simParam = NULL)
```

Arguments

`pop`	an object of `RawPop-class` or `MapPop-class`
`markers`	a character vector. Indicates the names of the loci to be retrieved.
`asRaw`	return in raw (byte) format
`simParam`	an object of `SimParam`, not used if pop is `MapPop-class`

Value

Returns a matrix of genotypes.

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=15)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitA(10)
SP$addSnpChip(5)

#Create population
pop = newPop(founderPop, simParam=SP)

#Pull genotype data for first two markers on chromosome one.
#Marker name is consistent with default naming in AlphaSimR.
pullMarkerGeno(pop, markers=c("1_1","1_2"), simParam=SP)
```

`pullMarkerHaplo` *Pull marker haplotypes*

Description

Retrieves haplotype data for user specified loci

Usage

```
pullMarkerHaplo(pop, markers, haplo = "all", asRaw = FALSE, simParam = NULL)
```

Arguments

`pop`	an object of `RawPop-class` or `MapPop-class`
`markers`	a character vector. Indicates the names of the loci to be retrieved
`haplo`	either "all" for all haplotypes or an integer for a single set of haplotypes. Use a value of 1 for female haplotypes and a value of 2 for male haplotypes in diploids.
`asRaw`	return in raw (byte) format
`simParam`	an object of `SimParam`, not used if pop is `MapPop-class`

Value

Returns a matrix of genotypes.

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=15)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitA(10)
SP$addSnpChip(5)
SP$setTrackRec(TRUE)

#Create population
pop = newPop(founderPop, simParam=SP)

#Pull haplotype data for first two markers on chromosome one.
#Marker name is consistent with default naming in AlphaSimR.
pullMarkerHaplo(pop, markers=c("1_1","1_2"), simParam=SP)
```

`pullQtlGeno` *Pull QTL genotypes*

Description

Retrieves QTL genotype data

Usage

```
pullQtlGeno(pop, trait = 1, chr = NULL, asRaw = FALSE, simParam = NULL)
```

Arguments

`pop`	an object of `Pop-class`
`trait`	an integer. Indicates which trait's QTL genotypes to retrieve.
`chr`	a vector of chromosomes to retrieve. If NULL, all chromosome are retrieved.
`asRaw`	return in raw (byte) format
`simParam`	an object of `SimParam`

Value

Returns a matrix of QTL genotypes.

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=15)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitA(10)
SP$addSnpChip(5)

#Create population
pop = newPop(founderPop, simParam=SP)
pullQtlGeno(pop, simParam=SP)
```

`pullQtlHaplo` *Pull QTL haplotypes*

Description

Retrieves QTL haplotype data

Usage

```
pullQtlHaplo(
  pop,
  trait = 1,
  haplo = "all",
  chr = NULL,
  asRaw = FALSE,
  simParam = NULL
)
```

Arguments

`pop`	an object of `Pop-class`
`trait`	an integer. Indicates which trait's QTL haplotypes to retrieve.
`haplo`	either "all" for all haplotypes or an integer for a single set of haplotypes. Use a value of 1 for female haplotypes and a value of 2 for male haplotypes in diploids.
`chr`	a vector of chromosomes to retrieve. If NULL, all chromosome are retrieved.
`asRaw`	return in raw (byte) format
`simParam`	an object of `SimParam`

Value

Returns a matrix of QTL haplotypes.

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=15)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitA(10)
SP$addSnpChip(5)

#Create population
pop = newPop(founderPop, simParam=SP)
pullQtlHaplo(pop, simParam=SP)
```

`pullSegSiteGeno` *Pull segregating site genotypes*

Description

Retrieves genotype data for all segregating sites

Usage

```
pullSegSiteGeno(pop, chr = NULL, asRaw = FALSE, simParam = NULL)
```

Arguments

`pop`	an object of `RawPop-class` or `MapPop-class`
`chr`	a vector of chromosomes to retrieve. If NULL, all chromosome are retrieved.
`asRaw`	return in raw (byte) format
`simParam`	an object of `SimParam`, not used if pop is `MapPop-class`

Value

Returns a matrix of genotypes

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=15)

#Set simulation parameters
SP = SimParam$new(founderPop)
SP$addTraitA(10)
SP$addSnpChip(5)

#Create population
```

```
pop = newPop(founderPop, simParam=SP)
pullSegSiteGeno(pop, simParam=SP)
```

`pullSegSiteHaplo` *Pull seg site haplotypes*

Description

Retrieves haplotype data for all segregating sites

Usage

```
pullSegSiteHaplo(
  pop,
  haplo = "all",
  chr = NULL,
  asRaw = FALSE,
  simParam = NULL
)
```

Arguments

`pop`	an object of `RawPop-class` or `MapPop-class`
`haplo`	either "all" for all haplotypes or an integer for a single set of haplotypes. Use a value of 1 for female haplotypes and a value of 2 for male haplotypes in diploids.
`chr`	a vector of chromosomes to retrieve. If NULL, all chromosome are retrieved.
`asRaw`	return in raw (byte) format
`simParam`	an object of `SimParam`, not used if pop is `MapPop-class`

Value

Returns a matrix of haplotypes

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=15)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitA(10)
SP$addSnpChip(5)

#Create population
pop = newPop(founderPop, simParam=SP)
pullSegSiteHaplo(pop, simParam=SP)
```

`pullSnpGeno` *Pull SNP genotypes*

Description

Retrieves SNP genotype data

Usage

```
pullSnpGeno(pop, snpChip = 1, chr = NULL, asRaw = FALSE, simParam = NULL)
```

Arguments

`pop`	an object of `Pop-class`
`snpChip`	an integer. Indicates which SNP chip's genotypes to retrieve.
`chr`	a vector of chromosomes to retrieve. If NULL, all chromosome are retrieved.
`asRaw`	return in raw (byte) format
`simParam`	an object of `SimParam`

Value

Returns a matrix of SNP genotypes.

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=15)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitA(10)
SP$addSnpChip(5)

#Create population
pop = newPop(founderPop, simParam=SP)
pullSnpGeno(pop, simParam=SP)
```

`pullSnpHaplo` *Pull SNP haplotypes*

Description

Retrieves SNP haplotype data

Usage

```
pullSnpHaplo(
  pop,
  snpChip = 1,
  haplo = "all",
  chr = NULL,
  asRaw = FALSE,
  simParam = NULL
)
```

Arguments

`pop`	an object of `Pop-class`
`snpChip`	an integer. Indicates which SNP chip's haplotypes to retrieve.
`haplo`	either "all" for all haplotypes or an integer for a single set of haplotypes. Use a value of 1 for female haplotypes and a value of 2 for male haplotypes in diploids.
`chr`	a vector of chromosomes to retrieve. If NULL, all chromosome are retrieved.
`asRaw`	return in raw (byte) format
`simParam`	an object of `SimParam`

Value

Returns a matrix of SNP haplotypes.

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=15)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitA(10)
SP$addSnpChip(5)

#Create population
pop = newPop(founderPop, simParam=SP)
pullSnpHaplo(pop, simParam=SP)
```

`quickHaplo` *Quick founder haplotype simulation*

Description

Rapidly simulates founder haplotypes by randomly sampling 0s and 1s. This is equivalent to having all loci with allele frequency 0.5 and being in linkage equilibrium.

Usage

```
quickHaplo(nInd, nChr, segSites, genLen = 1, ploidy = 2L, inbred = FALSE)
```

Arguments

`nInd`	number of individuals to simulate
`nChr`	number of chromosomes to simulate
`segSites`	number of segregating sites per chromosome
`genLen`	genetic length of chromosomes
`ploidy`	ploidy level of organism
`inbred`	should founder individuals be inbred

Value

an object of `MapPop-class`

Examples

```
# Creates a populations of 10 outbred individuals
# Their genome consists of 1 chromosome and 100 segregating sites
founderPop = quickHaplo(nInd=10,nChr=1,segSites=100)
```

`randCross` *Make random crosses*

Description

A wrapper for `makeCross` that randomly selects parental combinations for all possible combinantions.

Usage

```
randCross(
  pop,
  nCrosses,
  nProgeny = 1,
  balance = TRUE,
  parents = NULL,
  ignoreSexes = FALSE,
  simParam = NULL
)
```

Arguments

`pop`	an object of `Pop-class`
`nCrosses`	total number of crosses to make
`nProgeny`	number of progeny per cross
`balance`	if using sexes, this option will balance the number of progeny per parent
`parents`	an optional vector of indices for allowable parents
`ignoreSexes`	should sexes be ignored
`simParam`	an object of `SimParam`

Value

Returns an object of `Pop-class`

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)


#Create population
pop = newPop(founderPop, simParam=SP)

#Make 10 crosses
pop2 = randCross(pop, 10, simParam=SP)
```

randCross2 *Make random crosses*

Description

A wrapper for `makeCross2` that randomly selects parental combinations for all possible combinantions between two populations.

Usage

```
randCross2(
  females,
  males,
  nCrosses,
  nProgeny = 1,
  balance = TRUE,
  femaleParents = NULL,
  maleParents = NULL,
  ignoreSexes = FALSE,
  simParam = NULL
)
```

Arguments

`females`	an object of `Pop-class` for female parents.
`males`	an object of `Pop-class` for male parents.
`nCrosses`	total number of crosses to make
`nProgeny`	number of progeny per cross
`balance`	this option will balance the number of progeny per parent
`femaleParents`	an optional vector of indices for allowable female parents
`maleParents`	an optional vector of indices for allowable male parents
`ignoreSexes`	should sex be ignored
`simParam`	an object of `SimParam`

Value

Returns an object of `Pop-class`

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
```

```
#Create population
pop = newPop(founderPop, simParam=SP)

#Make 10 crosses
pop2 = randCross2(pop, pop, 10, simParam=SP)
```

`RawPop-class` *Raw Population*

Description

The raw population class contains only genotype data.

Usage

```
## S4 method for signature 'RawPop'
x[i]

## S4 method for signature 'RawPop'
c(x, ...)

## S4 method for signature 'RawPop'
show(object)

isRawPop(x)
```

Arguments

`x`	a ’RawPop’ object
`i`	index of individuals
`...`	additional ’RawPop’ objects
`object`	a ’RawPop’ object

Methods (by generic)

- `[`: Extract RawPop by index
- `c(RawPop)`: Combine multiple RawPops
- `show(RawPop)`: Show population summary

Functions

- `isRawPop()`: Test if object is of a RawPop class

Slots

`nInd`	number of individuals
`nChr`	number of chromosomes
`ploidy`	level of ploidy
`nLoci`	number of loci per chromosome
`geno`	list of nChr length containing chromosome genotypes. Each element is a three dimensional array of raw values. The array dimensions are nLoci by ploidy by nInd.

`reduceGenome` *Create individuals with reduced ploidy*

Description

Creates new individuals from gametes. This function was created to model the creation of diploid potatoes from tetraploid potatoes. It can be used on any population with an even ploidy level. The newly created individuals will have half the ploidy level of the originals. The reduction can occur with or without genetic recombination.

Usage

```
reduceGenome(
  pop,
  nProgeny = 1,
  useFemale = TRUE,
  keepParents = TRUE,
  simRecomb = TRUE,
  simParam = NULL
)
```

Arguments

`pop`	an object of 'Pop' superclass
`nProgeny`	total number of progeny per individual
`useFemale`	should female recombination rates be used.
`keepParents`	should previous parents be used for mother and father.
`simRecomb`	should genetic recombination be modeled.
`simParam`	an object of 'SimParam' class

Value

Returns an object of `Pop-class`

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=2, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)


#Create population
pop = newPop(founderPop, simParam=SP)

#Create individuals with reduced ploidy
pop2 = reduceGenome(pop, simParam=SP)
```

`resetPop` *Reset population*

Description

Recalculates a population's genetic values and resets phenotypes and EBVs.

Usage

```
resetPop(pop, simParam = NULL)
```

Arguments

`pop`	an object of `Pop-class`
`simParam`	an object of `SimParam`

Value

an object of `Pop-class`

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=2, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
SP$addTraitA(10)

#Create population
pop = newPop(founderPop, simParam=SP)

#Rescale to set mean to 1
SP$rescaleTraits(mean=1)
```

```
pop = resetPop(pop, simParam=SP)
```

RRBLUP *RR-BLUP Model*

Description

Fits an RR-BLUP model for genomic predictions.

Usage

```
RRBLUP(
  pop,
  traits = 1,
  use = "pheno",
  snpChip = 1,
  useQtl = FALSE,
  maxIter = 1000L,
  simParam = NULL,
  ...
)
```

Arguments

`pop`	a `Pop-class` to serve as the training population
`traits`	an integer indicating the trait or traits to model, a vector of trait names, or a function of the traits returning a single value.
`use`	train model using phenotypes "pheno", genetic values "gv", estimated breeding values "ebv", breeding values "bv", or randomly "rand"
`snpChip`	an integer indicating which SNP chip genotype to use
`useQtl`	should QTL genotypes be used instead of a SNP chip. If TRUE, snpChip specifies which trait's QTL to use, and thus these QTL may not match the QTL underlying the phenotype supplied in traits.
`maxIter`	maximum number of iterations. Only used when number of traits is greater than 1.
`simParam`	an object of `SimParam`
`...`	additional arguments if using a function for traits

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=20)

#Set simulation parameters
SP = SimParam$new(founderPop)
```

```
SP$addTraitA(10)
SP$setVarE(h2=0.5)
SP$addSnpChip(10)

#Create population
pop = newPop(founderPop, simParam=SP)

#Run GS model and set EBV
ans = RRBLUP(pop, simParam=SP)
pop = setEBV(pop, ans, simParam=SP)

#Evaluate accuracy
cor(gv(pop), ebv(pop))
```

`RRBLUP2` *RR-BLUP Model 2*

Description

Fits an RR-BLUP model for genomic predictions. This implementation is meant for situations where `RRBLUP` is too slow. Note that RRBLUP2 is only faster in certain situations, see details below. Most users should use `RRBLUP`.

Usage

```
RRBLUP2(
  pop,
  traits = 1,
  use = "pheno",
  snpChip = 1,
  useQtl = FALSE,
  maxIter = 10,
  Vu = NULL,
  Ve = NULL,
  useEM = TRUE,
  tol = 1e-06,
  simParam = NULL,
  ...
)
```

Arguments

`pop`	a `Pop-class` to serve as the training population
`traits`	an integer indicating the trait to model, a trait name, or a function of the traits returning a single value. Unlike `RRBLUP`, only univariate models are supported.

`use`	train model using phenotypes "pheno", genetic values "gv", estimated breeding values "ebv", breeding values "bv", or randomly "rand"
`snpChip`	an integer indicating which SNP chip genotype to use
`useQtl`	should QTL genotypes be used instead of a SNP chip. If TRUE, snpChip specifies which trait's QTL to use, and thus these QTL may not match the QTL underlying the phenotype supplied in traits.
`maxIter`	maximum number of iterations.
`Vu`	marker effect variance. If value is NULL, a reasonable starting point is chosen automatically.
`Ve`	error variance. If value is NULL, a reasonable starting point is chosen automatically.
`useEM`	use EM to solve variance components. If false, the initial values are considered true.
`tol`	tolerance for EM algorithm convergence
`simParam`	an object of `SimParam`
`...`	additional arguments if using a function for traits

Details

The RRBLUP2 function works best when the number of markers is not too large. This is because it solves the RR-BLUP problem by setting up and solving Henderson's mixed model equations. Solving these equations involves a square matrix with dimensions equal to the number of fixed effects plus the number of random effects (markers). Whereas the `RRBLUP` function solves the RR-BLUP problem using the EMMA approach. This approach involves a square matrix with dimensions equal to the number of phenotypic records. This means that the RRBLUP2 function uses less memory than RRBLUP when the number of markers is approximately equal to or smaller than the number of phenotypic records.

The RRBLUP2 function is not recommend for cases where the variance components are unknown. This is uses the EM algorithm to solve for unknown variance components, which is generally considerably slower than the EMMA approach of `RRBLUP`. The number of iterations for the EM algorithm is set by maxIter. The default value is typically too small for convergence. When the algorithm fails to converge a warning is displayed, but results are given for the last iteration. These results may be "good enough". However we make no claim to this effect, because we can not generalize to all possible use cases.

The RRBLUP2 function can quickly solve the mixed model equations without estimating variance components. The variance components are set by defining Vu and Ve. Estimation of components is suppressed by setting useEM to false. This may be useful if the model is being retrained multiple times during the simulation. You could run `RRBLUP` function the first time the model is trained, and then use the variance components from this output for all future runs with the RRBLUP2 functions. Again, we can make no claim to the general robustness of this approach.

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=20)
```

```
#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitA(10)
SP$setVarE(h2=0.5)
SP$addSnpChip(10)

#Create population
pop = newPop(founderPop, simParam=SP)

#Run GS model and set EBV
ans = RRBLUP2(pop, simParam=SP)
pop = setEBV(pop, ans, simParam=SP)

#Evaluate accuracy
cor(gv(pop), ebv(pop))
```

`RRBLUPMemUse` *RRBLUP Memory Usage*

Description

Estimates the amount of RAM needed to run the `RRBLUP` and its related functions for a given training population size. Note that this function may underestimate total usage.

Usage

```
RRBLUPMemUse(nInd, nMarker, model = "REG")
```

Arguments

`nInd`	the number of individuals in the training population
`nMarker`	the number of markers per individual
`model`	either "REG", "GCA", or "SCA" for `RRBLUP` `RRBLUP_GCA` and `RRBLUP_SCA` respectively.

Value

Returns an estimate for the required gigabytes of RAM

Examples

```
RRBLUPMemUse(nInd=1000, nMarker=5000)
```

RRBLUP_D *RR-BLUP Model with Dominance*

Description

Fits an RR-BLUP model for genomic predictions that includes dominance effects.

Usage

```
RRBLUP_D(
  pop,
  traits = 1,
  use = "pheno",
  snpChip = 1,
  useQtl = FALSE,
  maxIter = 40L,
  simParam = NULL,
  ...
)
```

Arguments

`pop`	a `Pop-class` to serve as the training population
`traits`	an integer indicating the trait to model, a trait name, or a function of the traits returning a single value.
`use`	train model using phenotypes "pheno", genetic values "gv", estimated breeding values "ebv", breeding values "bv", or randomly "rand"
`snpChip`	an integer indicating which SNP chip genotype to use
`useQtl`	should QTL genotypes be used instead of a SNP chip. If TRUE, snpChip specifies which trait's QTL to use, and thus these QTL may not match the QTL underlying the phenotype supplied in traits.
`maxIter`	maximum number of iterations. Only used when number of traits is greater than 1.
`simParam`	an object of `SimParam`
`...`	additional arguments if using a function for traits

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=20)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitAD(10, meanDD=0.5)
SP$setVarE(h2=0.5)
```

```
SP$addSnpChip(10)

#Create population
pop = newPop(founderPop, simParam=SP)

#Run GS model and set EBV
ans = RRBLUP_D(pop, simParam=SP)
pop = setEBV(pop, ans, simParam=SP)

#Evaluate accuracy
cor(gv(pop), ebv(pop))
```

`RRBLUP_D2` *RR-BLUP with Dominance Model 2*

Description

Fits an RR-BLUP model for genomic predictions that includes dominance effects. This implementation is meant for situations where `RRBLUP_D` is too slow. Note that RRBLUP_D2 is only faster in certain situations. Most users should use `RRBLUP_D`.

Usage

```
RRBLUP_D2(
  pop,
  traits = 1,
  use = "pheno",
  snpChip = 1,
  useQtl = FALSE,
  maxIter = 10,
  Va = NULL,
  Vd = NULL,
  Ve = NULL,
  useEM = TRUE,
  tol = 1e-06,
  simParam = NULL,
  ...
)
```

Arguments

`pop`	a `Pop-class` to serve as the training population
`traits`	an integer indicating the trait to model, a trait name, or a function of the traits returning a single value.
`use`	train model using phenotypes "pheno", genetic values "gv", estimated breeding values "ebv", breeding values "bv", or randomly "rand"

`snpChip`	an integer indicating which SNP chip genotype to use
`useQtl`	should QTL genotypes be used instead of a SNP chip. If TRUE, snpChip specifies which trait's QTL to use, and thus these QTL may not match the QTL underlying the phenotype supplied in traits.
`maxIter`	maximum number of iterations. Only used when number of traits is greater than 1.
`Va`	marker effect variance for additive effects. If value is NULL, a reasonable starting point is chosen automatically.
`Vd`	marker effect variance for dominance effects. If value is NULL, a reasonable starting point is chosen automatically.
`Ve`	error variance. If value is NULL, a reasonable starting point is chosen automatically.
`useEM`	use EM to solve variance components. If false, the initial values are considered true.
`tol`	tolerance for EM algorithm convergence
`simParam`	an object of `SimParam`
`...`	additional arguments if using a function for traits

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=20)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitAD(10, meanDD=0.5)
SP$setVarE(h2=0.5)
SP$addSnpChip(10)

#Create population
pop = newPop(founderPop, simParam=SP)

#Run GS model and set EBV
ans = RRBLUP_D2(pop, simParam=SP)
pop = setEBV(pop, ans, simParam=SP)

#Evaluate accuracy
cor(gv(pop), ebv(pop))
```

`RRBLUP_GCA` *RR-BLUP GCA Model*

Description

Fits an RR-BLUP model that estimates seperate marker effects for females and males. Useful for predicting GCA of parents in single cross hybrids. Can also predict performance of specific single cross hybrids.

Usage

```
RRBLUP_GCA(
  pop,
  traits = 1,
  use = "pheno",
  snpChip = 1,
  useQtl = FALSE,
  maxIter = 40L,
  simParam = NULL,
  ...
)
```

Arguments

`pop`	a `Pop-class` to serve as the training population
`traits`	an integer indicating the trait to model, a trait name, or a function of the traits returning a single value.
`use`	train model using phenotypes "pheno", genetic values "gv", estimated breeding values "ebv", breeding values "bv", or randomly "rand"
`snpChip`	an integer indicating which SNP chip genotype to use
`useQtl`	should QTL genotypes be used instead of a SNP chip. If TRUE, snpChip specifies which trait's QTL to use, and thus these QTL may not match the QTL underlying the phenotype supplied in traits.
`maxIter`	maximum number of iterations for convergence.
`simParam`	an object of `SimParam`
`...`	additional arguments if using a function for traits

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=20)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitA(10)
SP$setVarE(h2=0.5)
SP$addSnpChip(10)

#Create population
pop = newPop(founderPop, simParam=SP)
```

```
#Run GS model and set EBV
ans = RRBLUP_GCA(pop, simParam=SP)
pop = setEBV(pop, ans, simParam=SP)

#Evaluate accuracy
cor(gv(pop), ebv(pop))
```

RRBLUP_GCA2 *RR-BLUP GCA Model 2*

Description

Fits an RR-BLUP model that estimates seperate marker effects for females and males. This implementation is meant for situations where `RRBLUP_GCA` is too slow. Note that RRBLUP_GCA2 is only faster in certain situations. Most users should use `RRBLUP_GCA`.

Usage

```
RRBLUP_GCA2(
  pop,
  traits = 1,
  use = "pheno",
  snpChip = 1,
  useQtl = FALSE,
  maxIter = 10,
  VuF = NULL,
  VuM = NULL,
  Ve = NULL,
  useEM = TRUE,
  tol = 1e-06,
  simParam = NULL,
  ...
)
```

Arguments

`pop`	a `Pop-class` to serve as the training population
`traits`	an integer indicating the trait to model, a trait name, or a function of the traits returning a single value.
`use`	train model using phenotypes "pheno", genetic values "gv", estimated breeding values "ebv", breeding values "bv", or randomly "rand"
`snpChip`	an integer indicating which SNP chip genotype to use
`useQtl`	should QTL genotypes be used instead of a SNP chip. If TRUE, snpChip specifies which trait's QTL to use, and thus these QTL may not match the QTL underlying the phenotype supplied in traits.

`maxIter`	maximum number of iterations for convergence.
`VuF`	marker effect variance for females. If value is NULL, a reasonable starting point is chosen automatically.
`VuM`	marker effect variance for males. If value is NULL, a reasonable starting point is chosen automatically.
`Ve`	error variance. If value is NULL, a reasonable starting point is chosen automatically.
`useEM`	use EM to solve variance components. If false, the initial values are considered true.
`tol`	tolerance for EM algorithm convergence
`simParam`	an object of `SimParam`
`...`	additional arguments if using a function for traits

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=20)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitA(10)
SP$setVarE(h2=0.5)
SP$addSnpChip(10)

#Create population
pop = newPop(founderPop, simParam=SP)

#Run GS model and set EBV
ans = RRBLUP_GCA2(pop, simParam=SP)
pop = setEBV(pop, ans, simParam=SP)

#Evaluate accuracy
cor(gv(pop), ebv(pop))
```

`RRBLUP_SCA` *RR-BLUP SCA Model*

Description

An extention of `RRBLUP_GCA` that adds dominance effects. Note that we have not seen any consistent benefit of this model over `RRBLUP_GCA`.

Usage

```
RRBLUP_SCA(
  pop,
  traits = 1,
  use = "pheno",
  snpChip = 1,
  useQtl = FALSE,
  maxIter = 40L,
  simParam = NULL,
  ...
)
```

Arguments

`pop`	a `Pop-class` to serve as the training population
`traits`	an integer indicating the trait to model, a trait name, or a function of the traits returning a single value.
`use`	train model using phenotypes "pheno", genetic values "gv", estimated breeding values "ebv", breeding values "bv", or randomly "rand"
`snpChip`	an integer indicating which SNP chip genotype to use
`useQtl`	should QTL genotypes be used instead of a SNP chip. If TRUE, snpChip specifies which trait's QTL to use, and thus these QTL may not match the QTL underlying the phenotype supplied in traits.
`maxIter`	maximum number of iterations for convergence.
`simParam`	an object of `SimParam`
`...`	additional arguments if using a function for traits

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=2, nChr=1, segSites=20)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitA(10)
SP$setVarE(h2=0.5)
SP$addSnpChip(10)

#Create population
pop = newPop(founderPop, simParam=SP)

#Run GS model and set EBV
ans = RRBLUP_SCA(pop, simParam=SP)
pop = setEBV(pop, ans, simParam=SP)

#Evaluate accuracy
cor(gv(pop), ebv(pop))
```

RRBLUP_SCA2 *RR-BLUP SCA Model 2*

Description

Fits an RR-BLUP model that estimates seperate additive effects for females and males and a dominance effect. This implementation is meant for situations where `RRBLUP_SCA` is too slow. Note that RRBLUP_SCA2 is only faster in certain situations. Most users should use `RRBLUP_SCA`.

Usage

```
RRBLUP_SCA2(
  pop,
  traits = 1,
  use = "pheno",
  snpChip = 1,
  useQtl = FALSE,
  maxIter = 10,
  VuF = NULL,
  VuM = NULL,
  VuD = NULL,
  Ve = NULL,
  useEM = TRUE,
  tol = 1e-06,
  simParam = NULL,
  ...
)
```

Arguments

`pop`	a `Pop-class` to serve as the training population
`traits`	an integer indicating the trait to model, a trait name, or a function of the traits returning a single value.
`use`	train model using phenotypes "pheno", genetic values "gv", estimated breeding values "ebv", breeding values "bv", or randomly "rand"
`snpChip`	an integer indicating which SNP chip genotype to use
`useQtl`	should QTL genotypes be used instead of a SNP chip. If TRUE, snpChip specifies which trait's QTL to use, and thus these QTL may not match the QTL underlying the phenotype supplied in traits.
`maxIter`	maximum number of iterations for convergence.
`VuF`	marker effect variance for females. If value is NULL, a reasonable starting point is chosen automatically.
`VuM`	marker effect variance for males. If value is NULL, a reasonable starting point is chosen automatically.

`VuD`	marker effect variance for dominance. If value is NULL, a reasonable starting point is chosen automatically.
`Ve`	error variance. If value is NULL, a reasonable starting point is chosen automatically.
`useEM`	use EM to solve variance components. If false, the initial values are considered true.
`tol`	tolerance for EM algorithm convergence
`simParam`	an object of `SimParam`
`...`	additional arguments if using a function for traits

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=20)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitA(10)
SP$setVarE(h2=0.5)
SP$addSnpChip(10)

#Create population
pop = newPop(founderPop, simParam=SP)

#Run GS model and set EBV
ans = RRBLUP_SCA2(pop, simParam=SP)
pop = setEBV(pop, ans, simParam=SP)

#Evaluate accuracy
cor(gv(pop), ebv(pop))
```

`RRsol-class` *RR-BLUP Solution*

Description

Contains output from AlphaSimR’s genomic selection functions.

Slots

`gv`	Trait(s) for estimating genetic values
`bv`	Trait(s) for estimating breeding values
`female`	Trait(s) for estimating GCA in the female pool
`male`	Trait(s) for estimating GCA in the male pool
`Vu`	Estimated marker variance(s)
`Ve`	Estimated error variance

`runMacs` *Create founder haplotypes using MaCS*

Description

Uses the MaCS software to produce founder haplotypes (Chen et al. 2009).

Usage

```
runMacs(
  nInd,
  nChr = 1,
  segSites = NULL,
  inbred = FALSE,
  species = "GENERIC",
  split = NULL,
  ploidy = 2L,
  manualCommand = NULL,
  manualGenLen = NULL,
  nThreads = NULL
)
```

Arguments

`nInd`	number of individuals to simulate
`nChr`	number of chromosomes to simulate
`segSites`	number of segregating sites to keep per chromosome. A value of NULL results in all sites being retained.
`inbred`	should founder individuals be inbred
`species`	species history to simulate. See details.
`split`	an optional historic population split in terms of generations ago.
`ploidy`	ploidy level of organism
`manualCommand`	user provided MaCS options. For advanced users only.
`manualGenLen`	user provided genetic length. This must be supplied if using manualCommand. If not using manualCommand, this value will replace the predefined genetic length for the species. However, this the genetic length is only used by AlphaSimR and is not passed to MaCS, so MaCS still uses the predefined genetic length. For advanced users only.
`nThreads`	if OpenMP is available, this will allow for simulating chromosomes in parallel. If the value is NULL, the number of threads is automatically detected.

Details

There are currently three species histories available: GENERIC, CATTLE, WHEAT, and MAIZE.

The GENERIC history is meant to be a reasonable all-purpose choice. It runs quickly and models a population with an effective populations size that has gone through several historic bottlenecks. This species history is used as the default arguments in the `runMacs2` function, so the user should examine this function for the details of how the species is modeled.

The CATTLE history is based off of real genome sequence data (MacLeod et al. 2013).

The WHEAT (Gaynor et al. 2017) and MAIZE (Hickey et al. 2014) histories have been included due to their use in previous simulations. However, it should be noted that neither faithfully simulates its respective species. This is apparent by the low number of segregating sites simulated by each history relative to their real-world analogs. Adjusting these histories to better represent their real-world analogs would result in a drastic increase to runtime.

Value

an object of `MapPop-class`

References

Chen GK, Marjoram P, Wall JD (2009). "Fast and Flexible Simulation of DNA Sequence Data." *Genome Research*, **19**, 136-142. `https://genome.cshlp.org/content/19/1/136`.

Gaynor RC, Gorjanc G, Bentley AR, Ober ES, Howell P, Jackson R, Mackay IJ, Hickey JM (2017). "A Two-Part Strategy for Using Genomic Selection to Develop Inbred Lines." *Crop Science*, **57**(5), 2372–2386. ISSN 0011-183X, doi:10.2135/cropsci2016.09.0742, `https://acsess.onlinelibrary.wiley.com/doi/full/10.2135/cropsci2016.09.0742`.

Hickey JMDS, Crossa J, Hearne S, Babu R, Prasanna BM, Grondona M, Zambelli A, Windhausen VS, Mathews K, Gorjanc G (2014). "Evaluation of Genomic Selection Training Population Designs and Genotyping Strategies in Plant Breeding Programs Using Simulation." *Crop Science*, **54**(4), 1476-1488. doi:10.2135/cropsci2013.03.0195.

MacLeod IM, Larkin DM, Lewin HAHBJ, Goddard ME (2013). "Inferring Demography from Runs of Homozygosity in Whole-Genome Sequence, with Correction for Sequence Errors." *Molecular Biology and Evolution*, **30**(9), 2209–2223. doi:10.1093/molbev/mst125.

Examples

```
# Creates a populations of 10 outbred individuals
# Their genome consists of 1 chromosome and 100 segregating sites
## Not run:
founderPop = runMacs(nInd=10,nChr=1,segSites=100)

## End(Not run)
```

`runMacs2` *Alternative wrapper for MaCS*

Description

A wrapper function for `runMacs`. This wrapper is designed to provide a more intuitive interface for writing custom commands in MaCS (Chen et al. 2009). It effectively automates the creation of an appropriate line for the manualCommand argument in `runMacs` using user supplied variables, but only allows for a subset of the functionality offered by this argument. The default arguments of this function were chosen to match species="GENERIC" in `runMacs`.

Usage

```
runMacs2(
  nInd,
  nChr = 1,
  segSites = NULL,
  Ne = 100,
  bp = 1e+08,
  genLen = 1,
  mutRate = 2.5e-08,
  histNe = c(500, 1500, 6000, 12000, 1e+05),
  histGen = c(100, 1000, 10000, 1e+05, 1e+06),
  inbred = FALSE,
  split = NULL,
  ploidy = 2L,
  returnCommand = FALSE,
  nThreads = NULL
)
```

Arguments

`nInd`	number of individuals to simulate
`nChr`	number of chromosomes to simulate
`segSites`	number of segregating sites to keep per chromosome
`Ne`	effective population size
`bp`	base pair length of chromosome
`genLen`	genetic length of chromosome in Morgans
`mutRate`	per base pair mutation rate
`histNe`	effective population size in previous generations
`histGen`	number of generations ago for effective population sizes given in histNe
`inbred`	should founder individuals be inbred
`split`	an optional historic population split in terms of generations ago
`ploidy`	ploidy level of organism

`returnCommand`	should the command passed to manualCommand in `runMacs` be returned. If TRUE, MaCS will not be called and the command is returned instead.
`nThreads`	if OpenMP is available, this will allow for simulating chromosomes in parallel. If the value is NULL, the number of threads is automatically detected.

Value

an object of `MapPop-class` or if returnCommand is true a string giving the MaCS command passed to the manualCommand argument of `runMacs`.

References

Chen GK, Marjoram P, Wall JD (2009). "Fast and Flexible Simulation of DNA Sequence Data." *Genome Research*, **19**, 136-142. https://genome.cshlp.org/content/19/1/136.

Examples

```
# Creates a populations of 10 outbred individuals
# Their genome consists of 1 chromosome and 100 segregating sites
# The command is equivalent to using species="GENERIC" in runMacs
## Not run:
founderPop = runMacs2(nInd=10,nChr=1,segSites=100)

# runMacs() Implementation of the cattle demography following
#  Macleod et al. (2013) https://doi.org/10.1093/molbev/mst125
cattleChrSum = 2.8e9 # https://www.ncbi.nlm.nih.gov/datasets/genome/GCF_002263795.3/
(cattleChrBp = cattleChrSum / 30)
recRate = 9.26e-09
(cattleGenLen = recRate * cattleChrBp)
mutRate = 1.20e-08
runMacs2(nInd = 10, nChr = 1, Ne = 90, bp = cattleChrBp,
         genLen = cattleGenLen, mutRate = 1.20e-08,
       histNe  = c(120, 250, 350, 1000, 1500, 2000, 2500, 3500, 7000, 10000, 17000, 62000),
        histGen = c(  3,   6,  12,   18,   24,  154,  454,  654, 1754,  2354,  3354, 33154),
         returnCommand = TRUE)

## End(Not run)
```

`sampleHaplo` *Sample haplotypes from a MapPop*

Description

Creates a new `MapPop-class` from an existing `MapPop-class` by randomly sampling haplotypes.

Usage

```
sampleHaplo(mapPop, nInd, inbred = FALSE, ploidy = NULL, replace = TRUE)
```

Arguments

`mapPop`	the `MapPop-class` used to sample haplotypes
`nInd`	the number of individuals to create
`inbred`	should new individuals be fully inbred
`ploidy`	new ploidy level for organism. If NULL, the ploidy level of the mapPop is used.
`replace`	should haplotypes be sampled with replacement

Value

an object of `MapPop-class`

Examples

```
founderPop = quickHaplo(nInd=2,nChr=1,segSites=11,inbred=TRUE)
founderPop = sampleHaplo(mapPop=founderPop,nInd=20)
```

`selectCross` *Select and randomly cross*

Description

This is a wrapper that combines the functionalities of `randCross` and `selectInd`. The purpose of this wrapper is to combine both selection and crossing in one function call that minimized the amount of intermediate populations created. This reduces RAM usage and simplifies code writing. Note that this wrapper does not provide the full functionality of either function.

Usage

```
selectCross(
  pop,
  nInd = NULL,
  nFemale = NULL,
  nMale = NULL,
  nCrosses,
  nProgeny = 1,
  trait = 1,
  use = "pheno",
  selectTop = TRUE,
  simParam = NULL,
  ...,
  balance = TRUE
)
```

Arguments

`pop`	an object of `Pop-class`
`nInd`	the number of individuals to select. These individuals are selected without regards to sex and it supercedes values for nFemale and nMale. Thus if the simulation uses sexes, it is likely better to leave this value as NULL and use nFemale and nMale instead.
`nFemale`	the number of females to select. This value is ignored if nInd is set.
`nMale`	the number of males to select. This value is ignored if nInd is set.
`nCrosses`	total number of crosses to make
`nProgeny`	number of progeny per cross
`trait`	the trait for selection. Either a number indicating a single trait or a function returning a vector of length nInd.
`use`	select on genetic values "gv", estimated breeding values "ebv", breeding values "bv", phenotypes "pheno", or randomly "rand"
`selectTop`	selects highest values if true. Selects lowest values if false.
`simParam`	an object of `SimParam`
`...`	additional arguments if using a function for trait
`balance`	if using sexes, this option will balance the number of progeny per parent. This argument occurs after ..., so the argument name must be matched exactly.

Value

Returns an object of `Pop-class`

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitA(10)
SP$setVarE(h2=0.5)

#Create population
pop = newPop(founderPop, simParam=SP)

#Select 4 individuals and make 8 crosses
pop2 = selectCross(pop, nInd=4, nCrosses=8, simParam=SP)
```

`selectFam` *Select families*

Description

Selects a subset of full-sib families from a population.

Usage

```
selectFam(
  pop,
  nFam,
  trait = 1,
  use = "pheno",
  sex = "B",
  famType = "B",
  selectTop = TRUE,
  returnPop = TRUE,
  candidates = NULL,
  simParam = NULL,
  ...
)
```

Arguments

`pop`	and object of `Pop-class`, `HybridPop-class` or `MultiPop-class`
`nFam`	the number of families to select
`trait`	the trait for selection. Either a number indicating a single trait or a function returning a vector of length nInd. The function must work on a vector or matrix of use values as `trait(pop@use, ...)` - depending on what use is. See the examples and `selIndex`.
`use`	the selection criterion. Either a character (genetic values "gv", estimated breeding values "ebv", breeding values "bv", phenotypes "pheno", or randomly "rand") or a function returning a vector of length nInd. The function must work on pop as `use(pop, trait, ...)` or as `trait(pop@use, ...)` depending on what `trait` is. See the examples.
`sex`	which sex to select. Use "B" for both, "F" for females and "M" for males. If the simulation is not using sexes, the argument is ignored.
`famType`	which type of family to select. Use "B" for full-sib families, "F" for half-sib families on female side and "M" for half-sib families on the male side.
`selectTop`	selects highest values if true. Selects lowest values if false.
`returnPop`	should results be returned as a `Pop-class`. If FALSE, only the index of selected individuals is returned.
`candidates`	an optional vector of eligible selection candidates.
`simParam`	an object of `SimParam`
`...`	additional arguments if using a function for `trait` and `use`

Value

Returns an object of `Pop-class`, `HybridPop-class` or `MultiPop-class`

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitA(10)
SP$setVarE(h2=0.5)

#Create population
pop = newPop(founderPop, simParam=SP)

#Create 3 biparental families with 10 progeny
pop2 = randCross(pop, nCrosses=3, nProgeny=10, simParam=SP)

#Select best 2 families
pop3 = selectFam(pop2, 2, simParam=SP)
```

`selectInd` *Select individuals*

Description

Selects a subset of nInd individuals from a population.

Usage

```
selectInd(
  pop,
  nInd,
  trait = 1,
  use = "pheno",
  sex = "B",
  selectTop = TRUE,
  returnPop = TRUE,
  candidates = NULL,
  simParam = NULL,
  ...
)
```

Arguments

`pop`	and object of `Pop-class`, `HybridPop-class` or `MultiPop-class`
`nInd`	the number of individuals to select
`trait`	the trait for selection. Either a number indicating a single trait or a function returning a vector of length nInd. The function must work on a vector or matrix of use values as `trait(pop@use, ...)` - depending on what use is. See the examples and `selIndex`.
`use`	the selection criterion. Either a character (genetic values "gv", estimated breeding values "ebv", breeding values "bv", phenotypes "pheno", or randomly "rand") or a function returning a vector of length nInd. The function must work on pop as `use(pop, trait, ...)` or as `trait(pop@use, ...)` depending on what `trait` is. See the examples.
`sex`	which sex to select. Use "B" for both, "F" for females and "M" for males. If the simulation is not using sexes, the argument is ignored.
`selectTop`	selects highest values if true. Selects lowest values if false.
`returnPop`	should results be returned as a `Pop-class`. If FALSE, only the index of selected individuals is returned.
`candidates`	an optional vector of eligible selection candidates.
`simParam`	an object of `SimParam`
`...`	additional arguments if using a function for `trait` or `use`

Value

Returns an object of `Pop-class`, `HybridPop-class` or `MultiPop-class`

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitA(10)
SP$setVarE(h2=0.5)

#Create population
pop = newPop(founderPop, simParam=SP)

#Select top 5 (directional selection)
pop2 = selectInd(pop, 5, simParam=SP)
hist(pop@pheno); abline(v=pop@pheno, lwd=2)
abline(v=pop2@pheno, col="red", lwd=2)

#Select 5 most deviating from an optima (disruptive selection)
squaredDeviation = function(x, optima=0) (x - optima)^2
pop3 = selectInd(pop, 5, trait=squaredDeviation, selectTop=TRUE, simParam=SP)
```

```
hist(pop@pheno); abline(v=pop@pheno, lwd=2)
abline(v=pop3@pheno, col="red", lwd=2)

#Select 5 least deviating from an optima (stabilising selection)
pop4 = selectInd(pop, 5, trait=squaredDeviation, selectTop=FALSE, simParam=SP)
hist(pop@pheno); abline(v=pop@pheno, lwd=2)
abline(v=pop4@pheno, col="red", lwd=2)

#Select 5 individuals based on miscelaneous information with use function
pop@misc = list(smth=rnorm(10), smth2=rnorm(10))
useFunc = function(pop, trait=NULL) pop@misc$smth + pop@misc$smth2
pop5 = selectInd(pop, 5, use=useFunc, simParam=SP)
pop5@id

#... equivalent result with the use & trait function
useFunc2 = function(pop, trait=NULL) cbind(pop@misc$smth, pop@misc$smth2)
trtFunc = function(x) rowSums(x)
pop6 = selectInd(pop, 5, trait=trtFunc, use=useFunc2, simParam=SP)
pop6@id
```

`selectOP` *Select open pollinating plants*

Description

This function models selection in an open pollinating plant population. It allows for varying the percentage of selfing. The function also provides an option for modeling selection as occuring before or after pollination.

Usage

```
selectOP(
  pop,
  nInd,
  nSeeds,
  probSelf = 0,
  pollenControl = FALSE,
  trait = 1,
  use = "pheno",
  selectTop = TRUE,
  candidates = NULL,
  simParam = NULL,
  ...
)
```

Arguments

`pop`	and object of `Pop-class` or `MultiPop-class`
`nInd`	the number of plants to select
`nSeeds`	number of seeds per plant
`probSelf`	percentage of seeds expected from selfing. Value ranges from 0 to 1.
`pollenControl`	are plants selected before pollination
`trait`	the trait for selection. Either a number indicating a single trait or a function returning a vector of length nInd. The function must work on a vector or matrix of use values as `trait(pop@use, ...)` - depending on what use is. See the examples and `selIndex`.
`use`	the selection criterion. Either a character (genetic values "gv", estimated breeding values "ebv", breeding values "bv", phenotypes "pheno", or randomly "rand") or a function returning a vector of length nInd. The function must work on pop as `use(pop, trait, ...)` or as `trait(pop@use, ...)` depending on what `trait` is. See the examples.
`selectTop`	selects highest values if true. Selects lowest values if false.
`candidates`	an optional vector of eligible selection candidates.
`simParam`	an object of `SimParam`
`...`	additional arguments if using a function for `trait` and `use`

Value

Returns an object of `Pop-class` or `MultiPop-class`

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitA(10)
SP$setVarE(h2=0.5)

#Create population
pop = newPop(founderPop, simParam=SP)

#Create new population by selecting the best 3 plant
#Assuming 50% selfing in plants and 10 seeds per plant
pop2 = selectOP(pop, nInd=3, nSeeds=10, probSelf=0.5, simParam=SP)
```

`selectWithinFam` *Select individuals within families*

Description

Selects a subset of nInd individuals from each full-sib family within a population. Will return all individuals from a full-sib family if it has less than or equal to nInd individuals.

Usage

```
selectWithinFam(
  pop,
  nInd,
  trait = 1,
  use = "pheno",
  sex = "B",
  famType = "B",
  selectTop = TRUE,
  returnPop = TRUE,
  candidates = NULL,
  simParam = NULL,
  ...
)
```

Arguments

`pop`	and object of `Pop-class`, `HybridPop-class` or `MultiPop-class`
`nInd`	the number of individuals to select within a family
`trait`	the trait for selection. Either a number indicating a single trait or a function returning a vector of length nInd. The function must work on a vector or matrix of `use` values as `trait(pop@use, ...)` - depending on what `use` is. See the examples and `selIndex`.
`use`	the selection criterion. Either a character (genetic values "gv", estimated breeding values "ebv", breeding values "bv", phenotypes "pheno", or randomly "rand") or a function returning a vector of length nInd. The function must work on pop as `use(pop, trait, ...)` or as `trait(pop@use, ...)` depending on what `trait` is. See the examples.
`sex`	which sex to select. Use "B" for both, "F" for females and "M" for males. If the simulation is not using sexes, the argument is ignored.
`famType`	which type of family to select. Use "B" for full-sib families, "F" for half-sib families on female side and "M" for half-sib families on the male side.
`selectTop`	selects highest values if true. Selects lowest values if false.
`returnPop`	should results be returned as a `Pop-class`. If FALSE, only the index of selected individuals is returned.
`candidates`	an optional vector of eligible selection candidates.

`simParam`	an object of `SimParam`
`...`	additional arguments if using a function for `trait` and `use`

Value

Returns an object of `Pop-class`, `HybridPop-class` or `MultiPop-class`

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitA(10)
SP$setVarE(h2=0.5)

#Create population
pop = newPop(founderPop, simParam=SP)

#Create 3 biparental families with 10 progeny
pop2 = randCross(pop, nCrosses=3, nProgeny=10, simParam=SP)

#Select best individual per family
pop3 = selectWithinFam(pop2, 1, simParam=SP)
```

`self` *Self individuals*

Description

Creates selfed progeny from each individual in a population. Only works when sexes is "no".

Usage

```
self(pop, nProgeny = 1, parents = NULL, keepParents = TRUE, simParam = NULL)
```

Arguments

`pop`	an object of `Pop-class`
`nProgeny`	total number of selfed progeny per individual
`parents`	an optional vector of indices for allowable parents
`keepParents`	should previous parents be used for mother and father.
`simParam`	an object of `SimParam`

Value

Returns an object of Pop-class

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=2, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)


#Create population
pop = newPop(founderPop, simParam=SP)

#Self pollinate each individual
pop2 = self(pop, simParam=SP)
```

`selIndex` *Selection index*

Description

Calculates values of a selection index given trait values and weights. This function is intended to be used in combination with selection functions working on populations such as `selectInd`.

Usage

```
selIndex(Y, b, scale = FALSE)
```

Arguments

`Y`	a matrix of trait values
`b`	a vector of weights
`scale`	should Y be scaled and centered

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)

#Model two genetically correlated traits
G = 1.5*diag(2)-0.5 #Genetic correlation matrix
SP$addTraitA(10, mean=c(0,0), var=c(1,1), corA=G)
```

```
SP$setVarE(h2=c(0.5,0.5))

#Create population
pop = newPop(founderPop, simParam=SP)

#Calculate Smith-Hazel weights
econWt = c(1, 1)
b = smithHazel(econWt, varG(pop), varP(pop))

#Selection 2 best individuals using Smith-Hazel index
#selIndex is used as a trait
pop2 = selectInd(pop, nInd=2, trait=selIndex,
                 simParam=SP, b=b)
```

`selInt` *Selection intensity*

Description

Calculates the standardized selection intensity

Usage

```
selInt(p)
```

Arguments

`p`	the proportion of individuals selected

Examples

```
selInt(0.1)
```

`setEBV` *Set estimated breeding values (EBV)*

Description

Adds genomic estimated values to a populations's EBV slot using output from a genomic selection functions. The genomic estimated values can be either estimated breeding values, estimated genetic values, or estimated general combining values.

Usage

```
setEBV(
  pop,
  solution,
  value = "gv",
  targetPop = NULL,
  append = FALSE,
  simParam = NULL
)
```

Arguments

`pop`	an object of `Pop-class`
`solution`	an object of `RRsol-class`
`value`	the genomic value to be estimated. Can be either "gv", "bv", "female", or "male".
`targetPop`	an optional target population that can be used when value is "bv", "female", or "male". When supplied, the allele frequency in the targetPop is used to set these values.
`append`	should estimated values be appended to existing data in the EBV slot. If TRUE, a new column is added. If FALSE, existing data is replaced with the new estimates.
`simParam`	an object of `SimParam`

Value

Returns an object of `Pop-class`

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=20)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitA(10)
SP$setVarE(h2=0.5)
SP$addSnpChip(10)

#Create population
pop = newPop(founderPop, simParam=SP)

#Run GS model and set EBV
ans = RRBLUP(pop, simParam=SP)
pop = setEBV(pop, ans, simParam=SP)

#Evaluate accuracy
cor(gv(pop), ebv(pop))
```

`setMarkerHaplo` *Set marker haplotypes*

Description

Manually sets the haplotypes in a population for all individuals at one or more loci.

Usage

```
setMarkerHaplo(pop, haplo, simParam = NULL)
```

Arguments

`pop`	an object of `RawPop-class` or `MapPop-class`
`haplo`	a matrix of haplotypes, see details
`simParam`	an object of `SimParam`, not used if pop is `MapPop-class`

Details

The format of the haplotype matrix should match the format of the output from `pullMarkerHaplo` with the option haplo="all". Thus, it is recommended that this function is first used to extract the haplotypes and that any desired changes be made to the output of pullMarkerHaplo before passing the matrix to setMarkerHaplo. Any changes made to QTL may potentially result in changes to an individuals genetic value. These changes will be reflected in the gv and/or gxe slot. All other slots will remain unchanged, so the ebv and pheno slots will not reflect the new genotypes.

Value

an object of the same class as the "pop" input

Examples

```
# Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=15)

# Extract haplotypes for marker "1_1"
H = pullMarkerHaplo(founderPop, markers="1_1")

# Set the first haplotype to 1
H[1,1] = 1L

# Set marker haplotypes
founderPop = setMarkerHaplo(founderPop, haplo=H)
```

`setPheno` *Set phenotypes*

Description

Sets phenotypes for all traits by adding random error from a multivariate normal distribution.

Usage

```
setPheno(
  pop,
  h2 = NULL,
  H2 = NULL,
  varE = NULL,
  corE = NULL,
  reps = 1,
  fixEff = 1L,
  p = NULL,
  onlyPheno = FALSE,
  traits = NULL,
  simParam = NULL
)
```

Arguments

`pop`	an object of `Pop-class` or `HybridPop-class`
`h2`	a vector of desired narrow-sense heritabilities for each trait. See details.
`H2`	a vector of desired broad-sense heritabilities for each trait. See details.
`varE`	error (co)variances for traits. See details.
`corE`	an optional matrix for correlations between errors. See details.
`reps`	number of replications for phenotype. See details.
`fixEff`	fixed effect to assign to the population. Used by genomic selection models only.
`p`	the p-value for the environmental covariate used by GxE traits. If NULL, a value is sampled at random.
`onlyPheno`	should only the phenotype be returned, see return
`traits`	an integer vector indicate which traits to set. If NULL, all traits will be set.
`simParam`	an object of `SimParam`

Details

There are three arguments for setting the error variance of a phenotype: h2, H2, and varE. The user should only use one of these arguments. If the user supplies values for more than one, only one will be used according to order in which they are listed above.

The h2 argument allows the user to specify the error variance according to narrow-sense heritability. This calculation uses the additive genetic variance and total genetic variance in the founder population. Thus, the heritability relates to the founder population and not the current population.

The H2 argument allows the user to specify the error variance according to broad-sense heritability. This calculation uses the total genetic variance in the founder population. Thus, the heritability relates to the founder population and not the current population.

The varE argument allows the user to specify the error variance directly. The user may supply a vector describing the error variance for each trait or supply a matrix that specify the covariance of the errors.

The corE argument allows the user to specify correlations for the error covariance matrix. These correlations are be supplied in addition to the h2, H2, or varE arguments. These correlations will be used to construct a covariance matrix from a vector of variances. If the user supplied a covariance matrix to varE, these correlations will supercede values provided in that matrix.

The reps parameter is for convenient representation of replicated data. It is intended to represent replicated yield trials in plant breeding programs. In this case, varE is set to the plot error and reps is set to the number of plots per entry. The resulting phenotype represents the entry-means.

Value

Returns an object of `Pop-class` or `HybridPop-class` if onlyPheno=FALSE, if onlyPheno=TRUE a matrix is returned

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitA(10)

#Create population
pop = newPop(founderPop, simParam=SP)

#Add phenotype with error variance of 1
pop = setPheno(pop, varE=1)
```

`setPhenoGCA` *Set GCA as phenotype*

Description

Calculates general combining ability from a set of testers and returns these values as phenotypes for a population.

Usage

```
setPhenoGCA(
  pop,
  testers,
  use = "pheno",
  h2 = NULL,
  H2 = NULL,
  varE = NULL,
  corE = NULL,
  reps = 1,
  fixEff = 1L,
  p = NULL,
  inbred = FALSE,
  onlyPheno = FALSE,
  simParam = NULL
)
```

Arguments

`pop`	an object of `Pop-class`
`testers`	an object of `Pop-class`
`use`	true genetic value (gv) or phenotypes (pheno, default)
`h2`	a vector of desired narrow-sense heritabilities for each trait. See details in `setPheno`.
`H2`	a vector of desired broad-sense heritabilities for each trait. See details in `setPheno`.
`varE`	error (co)variances for traits. See details in `setPheno`.
`corE`	an optional matrix for correlations between errors. See details in `setPheno`.
`reps`	number of replications for phenotype. See details in `setPheno`.
`fixEff`	fixed effect to assign to the population. Used by genomic selection models only.
`p`	the p-value for the environmental covariate used by GxE traits. If NULL, a value is sampled at random.
`inbred`	are both pop and testers fully inbred. They are only fully inbred if created by `newPop` using inbred founders or by the `makeDH` function
`onlyPheno`	should only the phenotype be returned, see return
`simParam`	an object of `SimParam`

Value

Returns an object of `Pop-class` or a matrix if onlyPheno=TRUE

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10, inbred=TRUE)
```

```
#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitA(10)

#Create population
pop = newPop(founderPop, simParam=SP)

#Set phenotype to average per
pop2 = setPhenoGCA(pop, pop, use="gv", inbred=TRUE, simParam=SP)
```

`setPhenoProgTest` *Set progeny test as phenotype*

Description

Models a progeny test of individuals in 'pop'. Returns 'pop' with a phenotype representing the average performance of their progeny. The phenotype is generated by mating individuals in 'pop' to randomly chosen individuals in testPop a number of times equal to 'nMatePerInd'.

Usage

```
setPhenoProgTest(
  pop,
  testPop,
  nMatePerInd = 1L,
  use = "pheno",
  h2 = NULL,
  H2 = NULL,
  varE = NULL,
  corE = NULL,
  reps = 1,
  fixEff = 1L,
  p = NULL,
  onlyPheno = FALSE,
  simParam = NULL
)
```

Arguments

`pop`	an object of `Pop-class`
`testPop`	an object of `Pop-class`
`nMatePerInd`	number of times an individual in 'pop' is mated to an individual in testPop
`use`	true genetic value (gv) or phenotypes (pheno, default)
`h2`	a vector of desired narrow-sense heritabilities for each trait. See details in `setPheno`.

`H2`	a vector of desired broad-sense heritabilities for each trait. See details in `setPheno`.
`varE`	error (co)variances for traits. See details in `setPheno`.
`corE`	an optional matrix for correlations between errors. See details in `setPheno`.
`reps`	number of replications for phenotype. See details in `setPheno`.
`fixEff`	fixed effect to assign to the population. Used by genomic selection models only.
`p`	the p-value for the environmental covariate used by GxE traits. If NULL, a value is sampled at random.
`onlyPheno`	should only the phenotype be returned, see return
`simParam`	an object of `SimParam`

Details

The reps parameter is for convenient representation of replicated data. It was intended for representation of replicated yield trials in plant breeding programs. In this case, varE is set to the plot error and reps is set to the number plots per entry. The resulting phenotype would reflect the mean of all replications.

Value

Returns an object of `Pop-class` or a matrix if onlyPheno=TRUE

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10, inbred=TRUE)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitA(10)

#Create two populations of 5 individuals
pop1 = newPop(founderPop[1:5], simParam=SP)
pop2 = newPop(founderPop[6:10], simParam=SP)

#Set phenotype according to a progeny test
pop3 = setPhenoProgTest(pop1, pop2, use="gv", simParam=SP)
```

`SimParam` *Simulation parameters*

Description

Container for global simulation parameters. Saving this object as SP will allow it to be accessed by function defaults.

Public fields

- `nThreads` number of threads used on platforms with OpenMP support
- `snpChips` list of SNP chips
- `invalidQtl` list of segregating sites that aren't valid QTL
- `invalidSnp` list of segregating sites that aren't valid SNP
- `founderPop` founder population used for variance scaling
- `finalizePop` function applied to newly created populations. Currently does nothing and should only be changed by expert users.
- `allowEmptyPop` if true, population arguments with nInd=0 will return an empty population with a warning instead of an error.
- `v` the crossover interference parameter for a gamma model of recombination. A value of 1 indicates no crossover interference (e.g. Haldane mapping function). A value of 2.6 approximates the degree of crossover interference implied by the Kosambi mapping function. (default is 2.6)
- `p` the proportion of crossovers coming from a non-interfering pathway. (default is 0)
- `quadProb` the probability of quadrivalent pairing in an autopolyploid. (default is 0)

Active bindings

- `traitNames` vector of trait names
- `snpChipNames` vector of chip names
- `traits` list of traits
- `nChr` number of chromosomes
- `nTraits` number of traits
- `nSnpChips` number of SNP chips
- `segSites` segregating sites per chromosome
- `sexes` sexes used for mating
- `sepMap` are there seperate genetic maps for males and females
- `genMap` "matrix" of chromosome genetic maps
- `femaleMap` "matrix" of chromosome genetic maps for females
- `maleMap` "matrix" of chromosome genetic maps for males
- `centromere` position of centromeres genetic map
- `femaleCentromere` position of centromeres on female genetic map
- `maleCentromere` position of centromeres on male genetic map
- `lastId` last ID number assigned
- `isTrackPed` is pedigree being tracked
- `pedigree` pedigree matrix for all individuals
- `isTrackRec` is recombination being tracked
- `recHist` list of historic recombination events
- `haplotypes` list of computed IBD haplotypes
- `varA` additive genetic variance in founderPop
- `varG` total genetic variance in founderPop
- `varE` default error variance
- `version` the version of AlphaSimR used to generate this object

Methods

Public methods:

- `SimParam$new()`
- `SimParam$setTrackPed()`
- `SimParam$setTrackRec()`
- `SimParam$resetPed()`
- `SimParam$restrSegSites()`
- `SimParam$setSexes()`
- `SimParam$setFounderHap()`
- `SimParam$addSnpChip()`
- `SimParam$addSnpChipByName()`
- `SimParam$addStructuredSnpChip()`
- `SimParam$addTraitA()`
- `SimParam$addTraitAD()`
- `SimParam$altAddTraitAD()`
- `SimParam$addTraitAG()`
- `SimParam$addTraitADG()`
- `SimParam$addTraitAE()`
- `SimParam$addTraitADE()`
- `SimParam$addTraitAEG()`
- `SimParam$addTraitADEG()`
- `SimParam$manAddTrait()`
- `SimParam$importTrait()`
- `SimParam$switchTrait()`
- `SimParam$removeTrait()`
- `SimParam$setVarE()`
- `SimParam$setCorE()`
- `SimParam$rescaleTraits()`
- `SimParam$setRecombRatio()`
- `SimParam$switchGenMap()`
- `SimParam$switchFemaleMap()`
- `SimParam$switchMaleMap()`
- `SimParam$addToRec()`
- `SimParam$ibdHaplo()`
- `SimParam$updateLastId()`
- `SimParam$addToPed()`
- `SimParam$clone()`

Method `new()`: Starts the process of building a new simulation by creating a new SimParam object and assigning a founder population to the class. It is recommended that you save the object with the name "SP", because subsequent functions will check your global environment for an object of this name if their simParam arguments are NULL. This allows you to call these functions without explicitly supplying a simParam argument with every call.

Usage:

```
SimParam$new(founderPop)
```

Arguments:

`founderPop` an object of `MapPop-class`

Examples:

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
```

Method `setTrackPed()`**:** Sets pedigree tracking for the simulation. By default pedigree tracking is turned off. When turned on, the pedigree of all individuals created will be tracked, except those created by `hybridCross`. Turning off pedigree tracking will turn off recombination tracking if it is turned on.

Usage:

```
SimParam$setTrackPed(isTrackPed, force = FALSE)
```

Arguments:

`isTrackPed` should pedigree tracking be on.

`force` should the check for a running simulation be ignored. Only set to TRUE if you know what you are doing.

Examples:

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
\dontshow{SP$nThreads = 1L}
SP$setTrackPed(TRUE)
```

Method `setTrackRec()`**:** Sets recombination tracking for the simulation. By default recombination tracking is turned off. When turned on recombination tracking will also turn on pedigree tracking. Recombination tracking keeps records of all individuals created, except those created by `hybridCross`, because their pedigree is not tracked.

Usage:

```
SimParam$setTrackRec(isTrackRec, force = FALSE)
```

Arguments:

`isTrackRec` should recombination tracking be on.

`force` should the check for a running simulation be ignored. Only set to TRUE if you know what you are doing.

Examples:

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)
```

```
#Set simulation parameters
SP = SimParam$new(founderPop)
\dontshow{SP$nThreads = 1L}
SP$setTrackRec(TRUE)
```

Method `resetPed()`**:** Resets the internal lastId, the pedigree and recombination tracking (if in use) to the supplied lastId. Be careful using this function because it may introduce a bug if you use individuals from the deleted portion of the pedigree.

Usage:

```
SimParam$resetPed(lastId = 0L)
```

Arguments:

`lastId` last ID to include in pedigree

Examples:

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
\dontshow{SP$nThreads = 1L}

#Create population
pop = newPop(founderPop, simParam=SP)
pop@id # 1:10

#Create another population after reseting pedigree
SP$resetPed()
pop2 = newPop(founderPop, simParam=SP)
pop2@id # 1:10
```

Method `restrSegSites()`**:** Sets restrictions on which segregating sites can serve as a SNP and/or QTL.

Usage:

```
SimParam$restrSegSites(
  minQtlPerChr = NULL,
  minSnpPerChr = NULL,
  excludeQtl = NULL,
  excludeSnp = NULL,
  overlap = FALSE,
  minSnpFreq = NULL
)
```

Arguments:

`minQtlPerChr` the minimum number of segregating sites for QTLs. Can be a single value or a vector values for each chromosome.

`minSnpPerChr` the minimum number of segregating sites for SNPs. Can be a single value or a vector values for each chromosome.

`excludeQtl` an optional vector of segregating site names to exclude from consideration as a viable QTL.

`excludeSnp` an optional vector of segregating site names to exclude from consideration as a viable SNP.

`overlap` should SNP and QTL sites be allowed to overlap.

`minSnpFreq` minimum allowable frequency for SNP loci. No minimum SNP frequency is used if value is NULL.

Examples:

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
\dontshow{SP$nThreads = 1L}
SP$restrSegSites(minQtlPerChr=5, minSnpPerChr=5)
```

Method `setSexes()`**:** Changes how sexes are determined in the simulation. The default sexes is "no", indicating all individuals are hermaphrodites. To add sexes to the simulation, run this function with "yes_sys" or "yes_rand". The value "yes_sys" will systematically assign sexes to newly created individuals as first male and then female. Populations with an odd number of individuals will have one more male than female. The value "yes_rand" will randomly assign a sex to each individual.

Usage:

```
SimParam$setSexes(sexes, force = FALSE)
```

Arguments:

`sexes` acceptable value are "no", "yes_sys", or "yes_rand"

`force` should the check for a running simulation be ignored. Only set to TRUE if you know what you are doing.

Examples:

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
\dontshow{SP$nThreads = 1L}
SP$setSexes("yes_sys")
```

Method `setFounderHap()`**:** Allows for the manual setting of founder haplotypes. This functionality is not fully documented, because it is still experimental.

Usage:

```
SimParam$setFounderHap(hapMap)
```

Arguments:

`hapMap` a list of founder haplotypes

Method `addSnpChip()`**:** Randomly assigns eligible SNPs to a SNP chip

Usage:

```
SimParam$addSnpChip(nSnpPerChr, minSnpFreq = NULL, refPop = NULL, name = NULL)
```

Arguments:

`nSnpPerChr` number of SNPs per chromosome. Can be a single value or nChr values.

`minSnpFreq` minimum allowable frequency for SNP loci. If NULL, no minimum frequency is used.

`refPop` reference population for calculating SNP frequency. If NULL, the founder population is used.

`name` optional name for chip

Examples:

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
\dontshow{SP$nThreads = 1L}
SP$addSnpChip(10)
```

Method `addSnpChipByName()`**:** Assigns SNPs to a SNP chip by supplying marker names. This function does check against excluded SNPs and will not add the SNPs to the list of excluded QTL for the purpose of avoiding overlap between SNPs and QTL. Excluding these SNPs from being used as QTL can be accomplished using the excludeQtl argument in SimParam's restrSegSites function.

Usage:

```
SimParam$addSnpChipByName(markers, name = NULL)
```

Arguments:

`markers` a vector of names for the markers

`name` optional name for chip

Examples:

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
SP$addSnpChipByName(c("1_1","1_3"))
```

Method `addStructuredSnpChip()`**:** Randomly selects the number of snps in structure and then assigns them to chips based on structure

Usage:

```
SimParam$addStructuredSnpChip(nSnpPerChr, structure, force = FALSE)
```

Arguments:

`nSnpPerChr` number of SNPs per chromosome. Can be a single value or nChr values.

`structure` a matrix. Rows are snp chips, columns are chips. If value is true then that snp is on that chip.

`force` should the check for a running simulation be ignored. Only set to TRUE if you know what you are doing.

Method `addTraitA()`**:** Randomly assigns eligible QTLs for one or more additive traits. If simulating more than one trait, all traits will be pleiotropic with correlated additive effects.

Usage:

```
SimParam$addTraitA(
  nQtlPerChr,
  mean = 0,
  var = 1,
  corA = NULL,
  gamma = FALSE,
  shape = 1,
  force = FALSE,
  name = NULL
)
```

Arguments:

`nQtlPerChr` number of QTLs per chromosome. Can be a single value or nChr values.

`mean` a vector of desired mean genetic values for one or more traits

`var` a vector of desired genetic variances for one or more traits

`corA` a matrix of correlations between additive effects

`gamma` should a gamma distribution be used instead of normal

`shape` the shape parameter for the gamma distribution (the rate/scale parameter of the gamma distribution is accounted for via the desired level of genetic variance, the var argument)

`force` should the check for a running simulation be ignored. Only set to TRUE if you know what you are doing.

`name` optional name for trait(s)

Examples:

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
\dontshow{SP$nThreads = 1L}
SP$addTraitA(10)
```

Method `addTraitAD()`**:** Randomly assigns eligible QTLs for one or more traits with dominance. If simulating more than one trait, all traits will be pleiotropic with correlated effects.

Usage:

```
SimParam$addTraitAD(
  nQtlPerChr,
  mean = 0,
  var = 1,
  meanDD = 0,
  varDD = 0,
  corA = NULL,
```

```
  corDD = NULL,
  useVarA = TRUE,
  gamma = FALSE,
  shape = 1,
  force = FALSE,
  name = NULL
)
```

Arguments:

`nQtlPerChr` number of QTLs per chromosome. Can be a single value or nChr values.

`mean` a vector of desired mean genetic values for one or more traits

`var` a vector of desired genetic variances for one or more traits

`meanDD` mean dominance degree

`varDD` variance of dominance degree

`corA` a matrix of correlations between additive effects

`corDD` a matrix of correlations between dominance degrees

`useVarA` tune according to additive genetic variance if true. If FALSE, tuning is performed according to total genetic variance.

`gamma` should a gamma distribution be used instead of normal

`shape` the shape parameter for the gamma distribution (the rate/scale parameter of the gamma distribution is accounted for via the desired level of genetic variance, the var argument)

`force` should the check for a running simulation be ignored. Only set to TRUE if you know what you are doing.

`name` optional name for trait(s)

Examples:

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
\dontshow{SP$nThreads = 1L}
SP$addTraitAD(10, meanDD=0.5)
```

Method `altAddTraitAD()`**:** An alternative method for adding a trait with additive and dominance effects to an AlphaSimR simulation. The function attempts to create a trait matching user defined values for number of QTL, inbreeding depression, additive genetic variance and dominance genetic variance.

Usage:

```
SimParam$altAddTraitAD(
  nQtlPerChr,
  mean = 0,
  varA = 1,
  varD = 0,
  inbrDepr = 0,
  limMeanDD = c(0, 1.5),
  limVarDD = c(0, 0.5),
```

```
  silent = FALSE,
  force = FALSE,
  name = NULL
)
```

Arguments:

`nQtlPerChr` number of QTLs per chromosome. Can be a single value or nChr values.

`mean` desired mean of the trait

`varA` desired additive variance

`varD` desired dominance variance

`inbrDepr` desired inbreeding depression, see details

`limMeanDD` limits for meanDD, see details

`limVarDD` limits for varDD, see details

`silent` should summary details be printed to the console

`force` should the check for a running simulation be ignored. Only set to TRUE if you know what you are doing.

`name` optional name for trait

Details: This function will always add a trait to 'SimParam', unless an error occurs with picking QTLs. The resulting trait will always have the desired mean and additive genetic variance. However, it may not have the desired values for inbreeding depression and dominance variance. Thus, it is strongly recommended to check the output printed to the console to determine how close the trait's parameters came to these desired values.

The mean and additive genetic variance will always be achieved exactly. The function attempts to achieve the desired dominance variance and inbreeding depression while staying within the user supplied constraints for the acceptable range of dominance degree mean and variance. If the desired values are not being achieved, the acceptable range need to be increased and/or the number of QTL may need to be increased. There are not limits to setting the range for dominance degree mean and variance, but care should be taken to with regards to the biological feasibility of the limits that are supplied. The default limits were somewhat arbitrarily set, so I make not claim to how reasonable these limits are for routine use.

Inbreeding depression in this function is defined as the difference in mean genetic value between a population with the same allele frequency as the reference population (population used to initialize SimParam) in Hardy-Weinberg equilibrium compared to a population with the same allele frequency that is fully inbred. This is equivalent to the amount the mean of a population increases when going from an inbreeding coefficient of 1 (fully inbred) to a population with an inbreeding coefficient of 0 (Hardy-Weinberg equilibrium). Note that the sign of the value should (usually) be positive. This corresponds to a detrimental effect of inbreeding when higher values of the trait are considered biologically beneficial.

Summary information on this trait is printed to the console when silent=FALSE. The summary information reports the inbreeding depression and dominance variance for the population as well as the dominance degree mean and variance applied to the trait.

Examples:

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
```

```
\dontshow{SP$nThreads = 1L}
SP$altAddTraitAD(nQtlPerChr=10, mean=0, varA=1, varD=0.05, inbrDepr=0.2)
```

Method `addTraitAG()`**:** Randomly assigns eligible QTLs for one or more additive GxE traits. If simulating more than one trait, all traits will be pleiotropic with correlated effects.

Usage:

```
SimParam$addTraitAG(
  nQtlPerChr,
  mean = 0,
  var = 1,
  varGxE = 1e-06,
  varEnv = 0,
  corA = NULL,
  corGxE = NULL,
  gamma = FALSE,
  shape = 1,
  force = FALSE,
  name = NULL
)
```

Arguments:

`nQtlPerChr` number of QTLs per chromosome. Can be a single value or nChr values.

`mean` a vector of desired mean genetic values for one or more traits

`var` a vector of desired genetic variances for one or more traits

`varGxE` a vector of total genotype-by-environment variances for the traits

`varEnv` a vector of environmental variances for one or more traits

`corA` a matrix of correlations between additive effects

`corGxE` a matrix of correlations between GxE effects

`gamma` should a gamma distribution be used instead of normal

`shape` the shape parameter for the gamma distribution (the rate/scale parameter of the gamma distribution is accounted for via the desired level of genetic variance, the var argument)

`force` should the check for a running simulation be ignored. Only set to TRUE if you know what you are doing.

`name` optional name for trait(s)

Examples:

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
\dontshow{SP$nThreads = 1L}
SP$addTraitAG(10, varGxE=2)
```

Method `addTraitADG()`**:** Randomly assigns eligible QTLs for a trait with dominance and GxE.

Usage:

```
SimParam$addTraitADG(
  nQtlPerChr,
  mean = 0,
  var = 1,
  varEnv = 0,
  varGxE = 1e-06,
  meanDD = 0,
  varDD = 0,
  corA = NULL,
  corDD = NULL,
  corGxE = NULL,
  useVarA = TRUE,
  gamma = FALSE,
  shape = 1,
  force = FALSE,
  name = NULL
)
```

Arguments:

`nQtlPerChr` number of QTLs per chromosome. Can be a single value or nChr values.

`mean` a vector of desired mean genetic values for one or more traits

`var` a vector of desired genetic variances for one or more traits

`varEnv` a vector of environmental variances for one or more traits

`varGxE` a vector of total genotype-by-environment variances for the traits

`meanDD` mean dominance degree

`varDD` variance of dominance degree

`corA` a matrix of correlations between additive effects

`corDD` a matrix of correlations between dominance degrees

`corGxE` a matrix of correlations between GxE effects

`useVarA` tune according to additive genetic variance if true

`gamma` should a gamma distribution be used instead of normal

`shape` the shape parameter for the gamma distribution (the rate/scale parameter of the gamma distribution is accounted for via the desired level of genetic variance, the var argument)

`force` should the check for a running simulation be ignored. Only set to TRUE if you know what you are doing.

`name` optional name for trait(s)

Examples:

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
\dontshow{SP$nThreads = 1L}
SP$addTraitADG(10, meanDD=0.5, varGxE=2)
```

Method `addTraitAE()`**:** Randomly assigns eligible QTLs for one or more additive and epistasis traits. If simulating more than one trait, all traits will be pleiotropic with correlated additive effects.

Usage:

```
SimParam$addTraitAE(
  nQtlPerChr,
  mean = 0,
  var = 1,
  relAA = 0,
  corA = NULL,
  corAA = NULL,
  useVarA = TRUE,
  gamma = FALSE,
  shape = 1,
  force = FALSE,
  name = NULL
)
```

Arguments:

`nQtlPerChr` number of QTLs per chromosome. Can be a single value or nChr values.

`mean` a vector of desired mean genetic values for one or more traits

`var` a vector of desired genetic variances for one or more traits

`relAA` the relative value of additive-by-additive variance compared to additive variance in a diploid organism with allele frequency 0.5

`corA` a matrix of correlations between additive effects

`corAA` a matrix of correlations between additive-by-additive effects

`useVarA` tune according to additive genetic variance if true. If FALSE, tuning is performed according to total genetic variance.

`gamma` should a gamma distribution be used instead of normal

`shape` the shape parameter for the gamma distribution (the rate/scale parameter of the gamma distribution is accounted for via the desired level of genetic variance, the var argument)

`force` should the check for a running simulation be ignored. Only set to TRUE if you know what you are doing.

`name` optional name for trait(s)

Examples:

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
\dontshow{SP$nThreads = 1L}
SP$addTraitAE(10, relAA=0.1)
```

Method `addTraitADE()`**:** Randomly assigns eligible QTLs for one or more traits with dominance and epistasis. If simulating more than one trait, all traits will be pleiotropic with correlated effects.

Usage:

```
SimParam$addTraitADE(
  nQtlPerChr,
```

```
    mean = 0,
    var = 1,
    meanDD = 0,
    varDD = 0,
    relAA = 0,
    corA = NULL,
    corDD = NULL,
    corAA = NULL,
    useVarA = TRUE,
    gamma = FALSE,
    shape = 1,
    force = FALSE,
    name = NULL
  )
```

Arguments:

`nQtlPerChr` number of QTLs per chromosome. Can be a single value or nChr values.

`mean` a vector of desired mean genetic values for one or more traits

`var` a vector of desired genetic variances for one or more traits

`meanDD` mean dominance degree

`varDD` variance of dominance degree

`relAA` the relative value of additive-by-additive variance compared to additive variance in a diploid organism with allele frequency 0.5

`corA` a matrix of correlations between additive effects

`corDD` a matrix of correlations between dominance degrees

`corAA` a matrix of correlations between additive-by-additive effects

`useVarA` tune according to additive genetic variance if true. If FALSE, tuning is performed according to total genetic variance.

`gamma` should a gamma distribution be used instead of normal

`shape` the shape parameter for the gamma distribution (the rate/scale parameter of the gamma distribution is accounted for via the desired level of genetic variance, the var argument)

`force` should the check for a running simulation be ignored. Only set to TRUE if you know what you are doing.

`name` optional name for trait(s)

Examples:

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
\dontshow{SP$nThreads = 1L}
SP$addTraitADE(10)
```

Method `addTraitAEG()`**:** Randomly assigns eligible QTLs for one or more additive and epistasis GxE traits. If simulating more than one trait, all traits will be pleiotropic with correlated effects.

Usage:

```
SimParam$addTraitAEG(
  nQtlPerChr,
  mean = 0,
  var = 1,
  relAA = 0,
  varGxE = 1e-06,
  varEnv = 0,
  corA = NULL,
  corAA = NULL,
  corGxE = NULL,
  useVarA = TRUE,
  gamma = FALSE,
  shape = 1,
  force = FALSE,
  name = NULL
)
```

Arguments:

`nQtlPerChr` number of QTLs per chromosome. Can be a single value or nChr values.

`mean` a vector of desired mean genetic values for one or more traits

`var` a vector of desired genetic variances for one or more traits

`relAA` the relative value of additive-by-additive variance compared to additive variance in a diploid organism with allele frequency 0.5

`varGxE` a vector of total genotype-by-environment variances for the traits

`varEnv` a vector of environmental variances for one or more traits

`corA` a matrix of correlations between additive effects

`corAA` a matrix of correlations between additive-by-additive effects

`corGxE` a matrix of correlations between GxE effects

`useVarA` tune according to additive genetic variance if true. If FALSE, tuning is performed according to total genetic variance.

`gamma` should a gamma distribution be used instead of normal

`shape` the shape parameter for the gamma distribution (the rate/scale parameter of the gamma distribution is accounted for via the desired level of genetic variance, the var argument)

`force` should the check for a running simulation be ignored. Only set to TRUE if you know what you are doing.

`name` optional name for trait(s)

Examples:

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
\dontshow{SP$nThreads = 1L}
SP$addTraitAEG(10, varGxE=2)
```

Method `addTraitADEG()`**:** Randomly assigns eligible QTLs for a trait with dominance, epistasis and GxE.

Usage:

```
SimParam$addTraitADEG(
  nQtlPerChr,
  mean = 0,
  var = 1,
  varEnv = 0,
  varGxE = 1e-06,
  meanDD = 0,
  varDD = 0,
  relAA = 0,
  corA = NULL,
  corDD = NULL,
  corAA = NULL,
  corGxE = NULL,
  useVarA = TRUE,
  gamma = FALSE,
  shape = 1,
  force = FALSE,
  name = NULL
)
```

Arguments:

`nQtlPerChr` number of QTLs per chromosome. Can be a single value or nChr values.

`mean` a vector of desired mean genetic values for one or more traits

`var` a vector of desired genetic variances for one or more traits

`varEnv` a vector of environmental variances for one or more traits

`varGxE` a vector of total genotype-by-environment variances for the traits

`meanDD` mean dominance degree

`varDD` variance of dominance degree

`relAA` the relative value of additive-by-additive variance compared to additive variance in a diploid organism with allele frequency 0.5

`corA` a matrix of correlations between additive effects

`corDD` a matrix of correlations between dominance degrees

`corAA` a matrix of correlations between additive-by-additive effects

`corGxE` a matrix of correlations between GxE effects

`useVarA` tune according to additive genetic variance if true

`gamma` should a gamma distribution be used instead of normal

`shape` the shape parameter for the gamma distribution (the rate/scale parameter of the gamma distribution is accounted for via the desired level of genetic variance, the var argument)

`force` should the check for a running simulation be ignored. Only set to TRUE if you know what you are doing.

`name` optional name for trait(s)

Examples:

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)
```

```
#Set simulation parameters
SP = SimParam$new(founderPop)
\dontshow{SP$nThreads = 1L}
SP$addTraitADEG(10, meanDD=0.5, varGxE=2)
```

Method `manAddTrait()`**:** Manually add a new trait to the simulation. Trait must be formatted as a `LociMap-class`. If the trait is not already formatted, consider using importTrait.

Usage:

```
SimParam$manAddTrait(lociMap, varE = NA_real_, force = FALSE)
```

Arguments:

`lociMap` a new object descended from `LociMap-class`

`varE` default error variance for phenotype, optional

`force` should the check for a running simulation be ignored. Only set to TRUE if you know what you are doing

Method `importTrait()`**:** Manually add a new trait(s) to the simulation. Unlike the manAddTrait function, this function does not require formatting the trait as a `LociMap-class`. The formatting is performed automatically for the user, with more user friendly data.frames or matrices taken as inputs. This function only works for A and AD trait types.

Usage:

```
SimParam$importTrait(
  markerNames,
  addEff,
  domEff = NULL,
  intercept = NULL,
  name = NULL,
  varE = NULL,
  force = FALSE
)
```

Arguments:

`markerNames` a vector of names for the QTL

`addEff` a matrix of additive effects (nLoci x nTraits). Alternatively, a vector of length nLoci can be supplied for a single trait.

`domEff` optional dominance effects for each locus

`intercept` optional intercepts for each trait

`name` optional name(s) for the trait(s)

`varE` default error variance for phenotype, optional

`force` should the check for a running simulation be ignored. Only set to TRUE if you know what you are doing

Method `switchTrait()`**:** Switch a trait in the simulation.

Usage:

```
SimParam$switchTrait(traitPos, lociMap, varE = NA_real_, force = FALSE)
```

Arguments:

`traitPos` an integer indicate which trait to switch `lociMap` a new object descended from `LociMap-class`

`varE` default error variance for phenotype, optional

`force` should the check for a running simulation be ignored. Only set to TRUE if you know what you are doing

Method `removeTrait()`**:** Remove a trait from the simulation

Usage:

```
SimParam$removeTrait(traits, force = FALSE)
```

Arguments:

`traits` an integer vector indicating which traits to remove

`force` should the check for a running simulation be ignored. Only set to TRUE if you know what you are doing

Method `setVarE()`**:** Defines a default values for error variances used in `setPheno`. These defaults will be used to automatically generate phenotypes when new populations are created. See the details section of `setPheno` for more information about each arguments and how they should be used.

Usage:

```
SimParam$setVarE(h2 = NULL, H2 = NULL, varE = NULL, corE = NULL)
```

Arguments:

`h2` a vector of desired narrow-sense heritabilities

`H2` a vector of desired broad-sense heritabilities

`varE` a vector or matrix of error variances

`corE` an optional matrix of error correlations

Examples:

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
\dontshow{SP$nThreads = 1L}
SP$addTraitA(10)
SP$setVarE(h2=0.5)
```

Method `setCorE()`**:** Defines a correlation structure for default error variances. You must call `setVarE` first to define the default error variances.

Usage:

```
SimParam$setCorE(corE)
```

Arguments:

`corE` a correlation matrix for the error variances

Examples:

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
\dontshow{SP$nThreads = 1L}
SP$addTraitA(10, mean=c(0,0), var=c(1,1), corA=diag(2))
SP$setVarE(varE=c(1,1))
E = 0.5*diag(2)+0.5 #Positively correlated error
SP$setCorE(E)
```

Method `rescaleTraits()`**:** Linearly scales all traits to achieve desired values of means and variances in the founder population.

Usage:

```
SimParam$rescaleTraits(
  mean = 0,
  var = 1,
  varEnv = 0,
  varGxE = 1e-06,
  useVarA = TRUE
)
```

Arguments:

`mean` a vector of new trait means

`var` a vector of new trait variances

`varEnv` a vector of new environmental variances

`varGxE` a vector of new GxE variances

`useVarA` tune according to additive genetic variance if true

Examples:

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
SP$addTraitA(10)

#Create population
pop = newPop(founderPop, simParam=SP)
meanG(pop)

#Change mean to 1
SP$rescaleTraits(mean=1)
\dontshow{SP$nThreads = 1L}
#Run resetPop for change to take effect
pop = resetPop(pop, simParam=SP)
meanG(pop)
```

Method `setRecombRatio()`: Set the relative recombination rates between males and females. This allows for sex-specific recombination rates, under the assumption of equivalent recombination landscapes.

Usage:

```
SimParam$setRecombRatio(femaleRatio)
```

Arguments:

`femaleRatio` relative ratio of recombination in females compared to males. A value of 2 indicate twice as much recombination in females. The value must be greater than 0. (default is 1)

Examples:

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
\dontshow{SP$nThreads = 1L}
SP$setRecombRatio(2) #Twice as much recombination in females
```

Method `switchGenMap()`: Replaces existing genetic map.

Usage:

```
SimParam$switchGenMap(genMap, centromere = NULL)
```

Arguments:

`genMap` a list of length nChr containing numeric vectors for the position of each segregating site on a chromosome.

`centromere` a numeric vector of centromere positions. If NULL, the centromere are assumed to be metacentric.

Method `switchFemaleMap()`: Replaces existing female genetic map.

Usage:

```
SimParam$switchFemaleMap(genMap, centromere = NULL)
```

Arguments:

`genMap` a list of length nChr containing numeric vectors for the position of each segregating site on a chromosome.

`centromere` a numeric vector of centromere positions. If NULL, the centromere are assumed to be metacentric.

Method `switchMaleMap()`: Replaces existing male genetic map.

Usage:

```
SimParam$switchMaleMap(genMap, centromere = NULL)
```

Arguments:

`genMap` a list of length nChr containing numeric vectors for the position of each segregating site on a chromosome.

`centromere` a numeric vector of centromere positions. If NULL, the centromere are assumed to be metacentric.

Method `addToRec()`**:** For internal use only.

Usage:

`SimParam$addToRec(lastId, id, mother, father, isDH, hist, ploidy)`

Arguments:

`lastId` ID of last individual

`id` the name of each individual

`mother` vector of mother iids

`father` vector of father iids

`isDH` indicator for DH lines

`hist` new recombination history

`ploidy` ploidy level

Method `ibdHaplo()`**:** For internal use only.

Usage:

`SimParam$ibdHaplo(iid)`

Arguments:

`iid` internal ID

Method `updateLastId()`**:** For internal use only.

Usage:

`SimParam$updateLastId(lastId)`

Arguments:

`lastId` last ID assigned

Method `addToPed()`**:** For internal use only.

Usage:

`SimParam$addToPed(lastId, id, mother, father, isDH)`

Arguments:

`lastId` ID of last individual

`id` the name of each individual

`mother` vector of mother iids

`father` vector of father iids

`isDH` indicator for DH lines

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

`SimParam$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Note

By default the founder population is the population used to initalize the SimParam object. This population can be changed by replacing the population in the founderPop slot. You must run `resetPop` on any existing populations to obtain the new trait values.

Examples

```
## ------------------------------------------------
## Method `SimParam$new`
## ------------------------------------------------

#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)

## ------------------------------------------------
## Method `SimParam$setTrackPed`
## ------------------------------------------------

#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$setTrackPed(TRUE)

## ------------------------------------------------
## Method `SimParam$setTrackRec`
## ------------------------------------------------

#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$setTrackRec(TRUE)

## ------------------------------------------------
## Method `SimParam$resetPed`
## ------------------------------------------------

#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)


#Create population
pop = newPop(founderPop, simParam=SP)
pop@id # 1:10

#Create another population after reseting pedigree
SP$resetPed()
```

```
pop2 = newPop(founderPop, simParam=SP)
pop2@id # 1:10

## ------------------------------------------------
## Method `SimParam$restrSegSites`
## ------------------------------------------------

#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$restrSegSites(minQtlPerChr=5, minSnpPerChr=5)

## ------------------------------------------------
## Method `SimParam$setSexes`
## ------------------------------------------------

#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$setSexes("yes_sys")

## ------------------------------------------------
## Method `SimParam$addSnpChip`
## ------------------------------------------------

#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addSnpChip(10)

## ------------------------------------------------
## Method `SimParam$addSnpChipByName`
## ------------------------------------------------

#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
SP$addSnpChipByName(c("1_1","1_3"))

## ------------------------------------------------
## Method `SimParam$addTraitA`
## ------------------------------------------------
```

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitA(10)

## ------------------------------------------------
## Method `SimParam$addTraitAD`
## ------------------------------------------------

#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitAD(10, meanDD=0.5)

## ------------------------------------------------
## Method `SimParam$altAddTraitAD`
## ------------------------------------------------

#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$altAddTraitAD(nQtlPerChr=10, mean=0, varA=1, varD=0.05, inbrDepr=0.2)

## ------------------------------------------------
## Method `SimParam$addTraitAG`
## ------------------------------------------------

#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitAG(10, varGxE=2)

## ------------------------------------------------
## Method `SimParam$addTraitADG`
## ------------------------------------------------

#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
```

```
SP = SimParam$new(founderPop)

SP$addTraitADG(10, meanDD=0.5, varGxE=2)

## ------------------------------------------------
## Method `SimParam$addTraitAE`
## ------------------------------------------------

#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitAE(10, relAA=0.1)

## ------------------------------------------------
## Method `SimParam$addTraitADE`
## ------------------------------------------------

#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitADE(10)

## ------------------------------------------------
## Method `SimParam$addTraitAEG`
## ------------------------------------------------

#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitAEG(10, varGxE=2)

## ------------------------------------------------
## Method `SimParam$addTraitADEG`
## ------------------------------------------------

#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitADEG(10, meanDD=0.5, varGxE=2)

## ------------------------------------------------
```

```
## Method `SimParam$setVarE`
## ------------------------------------------------

#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitA(10)
SP$setVarE(h2=0.5)

## ------------------------------------------------
## Method `SimParam$setCorE`
## ------------------------------------------------

#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitA(10, mean=c(0,0), var=c(1,1), corA=diag(2))
SP$setVarE(varE=c(1,1))
E = 0.5*diag(2)+0.5 #Positively correlated error
SP$setCorE(E)

## ------------------------------------------------
## Method `SimParam$rescaleTraits`
## ------------------------------------------------

#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
SP$addTraitA(10)

#Create population
pop = newPop(founderPop, simParam=SP)
meanG(pop)

#Change mean to 1
SP$rescaleTraits(mean=1)

#Run resetPop for change to take effect
pop = resetPop(pop, simParam=SP)
meanG(pop)

## ------------------------------------------------
## Method `SimParam$setRecombRatio`
## ------------------------------------------------
```

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$setRecombRatio(2) #Twice as much recombination in females
```

`smithHazel` *Calculate Smith-Hazel weights*

Description

Calculates weights for Smith-Hazel index given economice weights and phenotypic and genotypic variance-covariance matrices.

Usage

```
smithHazel(econWt, varG, varP)
```

Arguments

`econWt`	vector of economic weights
`varG`	the genetic variance-covariance matrix
`varP`	the phenotypic variance-covariance matrix

Value

a vector of weight for calculating index values

Examples

```
G = 1.5*diag(2)-0.5
E = diag(2)
P = G+E
wt = c(1,1)
smithHazel(wt, G, P)
```

`solveMKM` *Solve Multikernel Model*

Description

Solves a univariate mixed model with multiple random effects.

Usage

```
solveMKM(y, X, Zlist, Klist, maxIter = 40L, tol = 1e-04)
```

Arguments

`y`	a matrix with n rows and 1 column
`X`	a matrix with n rows and x columns
`Zlist`	a list of Z matrices
`Klist`	a list of K matrices
`maxIter`	maximum number of iteration
`tol`	tolerance for convergence

`solveMVM` *Solve Multivariate Model*

Description

Solves a multivariate mixed model of form $Y = X\beta + Zu + e$

Usage

```
solveMVM(Y, X, Z, K, tol = 1e-06, maxIter = 1000L)
```

Arguments

`Y`	a matrix with n rows and q columns
`X`	a matrix with n rows and x columns
`Z`	a matrix with n rows and m columns
`K`	a matrix with m rows and m columns
`tol`	tolerance for convergence
`maxIter`	maximum number of iteration

`solveRRBLUP` *Solve RR-BLUP*

Description

Solves a univariate mixed model of form $y = X\beta + Mu + e$

Usage

```
solveRRBLUP(y, X, M)
```

Arguments

`y`	a matrix with n rows and 1 column
`X`	a matrix with n rows and x columns
`M`	a matrix with n rows and m columns

`solveRRBLUPMK` *Solve Multikernel RR-BLUP*

Description

Solves a univariate mixed model with multiple random effects.

Usage

```
solveRRBLUPMK(y, X, Mlist, maxIter = 40L)
```

Arguments

`y`	a matrix with n rows and 1 column
`X`	a matrix with n rows and x columns
`Mlist`	a list of M matrices
`maxIter`	maximum number of iteration

`solveRRBLUPMV` *Solve Multivariate RR-BLUP*

Description

Solves a multivariate mixed model of form $Y = X\beta + Mu + e$

Usage

```
solveRRBLUPMV(Y, X, M, maxIter = 1000L, tol = 1e-06)
```

Arguments

`Y`	a matrix with n rows and q columns
`X`	a matrix with n rows and x columns
`M`	a matrix with n rows and m columns
`maxIter`	maximum number of iteration
`tol`	tolerance for convergence

`solveRRBLUP_EM` *Solve RR-BLUP with EM*

Description

Solves a univariate mixed model of form $y = X\beta + Mu + e$ using the Expectation-Maximization algorithm.

Usage

```
solveRRBLUP_EM(Y, X, M, Vu, Ve, tol, maxIter, useEM)
```

Arguments

`Y`	a matrix with n rows and 1 column
`X`	a matrix with n rows and x columns
`M`	a matrix with n rows and m columns
`Vu`	initial guess for variance of marker effects
`Ve`	initial guess for error variance
`tol`	tolerance for declaring convergence
`maxIter`	maximum iteration for attempting convergence
`useEM`	should EM algorithm be used. If false, no estimation of variance components is performed. The initial values are treated as true.

`solveRRBLUP_EM2` *Solve RR-BLUP with EM and 2 random effects*

Description

Solves a univariate mixed model of form $y = X\beta + M_1u_1 + M_2u_2 + e$ using the Expectation-Maximization algorithm.

Usage

```
solveRRBLUP_EM2(Y, X, M1, M2, Vu1, Vu2, Ve, tol, maxIter, useEM)
```

Arguments

`Y`	a matrix with n rows and 1 column
`X`	a matrix with n rows and x columns
`M1`	a matrix with n rows and m1 columns
`M2`	a matrix with n rows and m2 columns
`Vu1`	initial guess for variance of the first marker effects
`Vu2`	initial guess for variance of the second marker effects
`Ve`	initial guess for error variance
`tol`	tolerance for declaring convergence
`maxIter`	maximum iteration for attempting convergence
`useEM`	should EM algorithm be used. If false, no estimation of variance components is performed. The initial values are treated as true.

`solveRRBLUP_EM3` *Solve RR-BLUP with EM and 3 random effects*

Description

Solves a univariate mixed model of form $y = X\beta + M_1u_1 + M_2u_2 + M_3u_3 + e$ using the Expectation-Maximization algorithm.

Usage

```
solveRRBLUP_EM3(Y, X, M1, M2, M3, Vu1, Vu2, Vu3, Ve, tol, maxIter, useEM)
```

Arguments

`Y`	a matrix with n rows and 1 column
`X`	a matrix with n rows and x columns
`M1`	a matrix with n rows and m1 columns
`M2`	a matrix with n rows and m2 columns
`M3`	a matrix with n rows and m3 columns
`Vu1`	initial guess for variance of the first marker effects
`Vu2`	initial guess for variance of the second marker effects
`Vu3`	initial guess for variance of the second marker effects
`Ve`	initial guess for error variance
`tol`	tolerance for declaring convergence
`maxIter`	maximum iteration for attempting convergence
`useEM`	should EM algorithm be used. If false, no estimation of variance components is performed. The initial values are treated as true.

`solveUVM` *Solve Univariate Model*

Description

Solves a univariate mixed model of form $y = X\beta + Zu + e$

Usage

```
solveUVM(y, X, Z, K)
```

Arguments

`y`	a matrix with n rows and 1 column
`X`	a matrix with n rows and x columns
`Z`	a matrix with n rows and m columns
`K`	a matrix with m rows and m columns

`TraitA-class` *Additive trait*

Description

Extends `LociMap-class` to model additive traits

Slots

`addEff` additive effects

`intercept` adjustment factor for gv

`TraitA2-class` *Sex specific additive trait*

Description

Extends `TraitA-class` to model seperate additive effects for parent of origin. Used exclusively for genomic selection.

Slots

`addEffMale` additive effects

`TraitA2D-class` *Sex specific additive and dominance trait*

Description

Extends `TraitA2-class` to add dominance

Slots

`domEff` dominance effects

`TraitAD-class` *Additive and dominance trait*

Description

Extends `TraitA-class` to add dominance

Slots

`domEff` dominance effects

`TraitADE-class` *Additive, dominance, and epistatic trait*

Description

Extends `TraitAD-class` to add epistasis

Slots

`epiEff` epistatic effects

`TraitADEG-class` *Additive, dominance, epistasis, and GxE trait*

Description

Extends `TraitADE-class` to add GxE effects

Slots

`gxeEff`	GxE effects
`gxeInt`	GxE intercept
`envVar`	Environmental variance

`TraitADG-class` *Additive, dominance and GxE trait*

Description

Extends `TraitAD-class` to add GxE effects

Slots

`gxeEff`	GxE effects
`gxeInt`	GxE intercept
`envVar`	Environmental variance

`TraitAE-class` *Additive and epistatic trait*

Description

Extends `TraitA-class` to add epistasis

Slots

`epiEff`	epistatic effects

`TraitAEG-class` *Additive, epistasis and GxE trait*

Description

Extends `TraitAE-class` to add GxE effects

Slots

`gxeEff`	GxE effects
`gxeInt`	GxE intercept
`envVar`	Environmental variance

`TraitAG-class` *Additive and GxE trait*

Description

Extends `TraitA-class` to add GxE effects

Slots

`gxeEff`	GxE effects
`gxeInt`	GxE intercept
`envVar`	Environmental variance

`transMat` *Linear transformation matrix*

Description

Creates an m by m linear transformation matrix that can be applied to n by m uncorrelated deviates sampled from a standard normal distribution to produce correlated deviates with an arbitrary correlation of R. If R is not positive semi-definite, the function returns smoothing and returns a warning (see details).

Usage

```
transMat(R)
```

Arguments

`R`	a correlation matrix

Details

An eigendecomposition is applied to the correlation matrix and used to test if it is positive semi-definite. If the matrix is not positive semi-definite, it is not a valid correlation matrix. In this case, smoothing is applied to the matrix (as described in the 'cor.smooth' of the 'psych' library) to obtain a valid correlation matrix. The resulting deviates will thus not exactly match the desired correlation, but will hopefully be close if the input matrix wasn't too far removed from a valid correlation matrix.

Examples

```
# Create an 2x2 correlation matrix
R = 0.5*diag(2) + 0.5

# Sample 1000 uncorrelated deviates from a
# bivariate standard normal distribution
X = matrix(rnorm(2*1000), ncol=2)

# Compute the transformation matrix
T = transMat(R)

# Apply the transformation to the deviates
Y = X%*%T

# Measure the sample correlation
cor(Y)
```

`usefulness` *Usefulness criterion*

Description

Calculates the usefulness criterion

Usage

```
usefulness(
  pop,
  trait = 1,
  use = "gv",
  p = 0.1,
  selectTop = TRUE,
  simParam = NULL,
  ...
)
```

Arguments

`pop`	and object of `Pop-class` or `HybridPop-class`
`trait`	the trait for selection. Either a number indicating a single trait or a function returning a vector of length nInd.
`use`	select on genetic values (gv, default), estimated breeding values (ebv), breeding values (bv), or phenotypes (pheno)
`p`	the proportion of individuals selected
`selectTop`	selects highest values if true. Selects lowest values if false.
`simParam`	an object of `SimParam`
`...`	additional arguments if using a function for trait

Value

Returns a numeric value

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=2, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)

SP$addTraitA(10)

#Create population
pop = newPop(founderPop, simParam=SP)

#Determine usefulness of population
usefulness(pop, simParam=SP)

#Should be equivalent to GV of best individual
max(gv(pop))
```

`varA` *Additive variance*

Description

Returns additive variance for all traits

Usage

```
varA(pop, simParam = NULL)
```

Arguments

`pop`	an object of `Pop-class`
`simParam`	an object of `SimParam`

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
SP$addTraitAD(10, meanDD=0.5)
SP$setVarE(h2=0.5)


#Create population
pop = newPop(founderPop, simParam=SP)
varA(pop, simParam=SP)
```

`varAA` *Additive-by-additive epistatic variance*

Description

Returns additive-by-additive epistatic variance for all traits

Usage

```
varAA(pop, simParam = NULL)
```

Arguments

`pop`	an object of `Pop-class`
`simParam`	an object of `SimParam`

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
SP$addTraitAD(10, meanDD=0.5)
SP$setVarE(h2=0.5)


#Create population
```

```
pop = newPop(founderPop, simParam=SP)
varAA(pop, simParam=SP)
```

---

varD *Dominance variance*

Description

Returns dominance variance for all traits

Usage

```
varD(pop, simParam = NULL)
```

Arguments

`pop`	an object of `Pop-class`
`simParam`	an object of `SimParam`

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
SP$addTraitAD(10, meanDD=0.5)
SP$setVarE(h2=0.5)


#Create population
pop = newPop(founderPop, simParam=SP)
varD(pop, simParam=SP)
```

---

varEBV *Variance of estimated breeding values*

Description

Returns variance of estimated breeding values for all traits

Usage

```
varEBV(pop)
```

Arguments

`pop`	an object of `Pop-class` or `HybridPop-class`

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
SP$addTraitA(10)
trtH2 = 0.5
SP$setVarE(h2=trtH2)


#Create population
pop = newPop(founderPop, simParam=SP)
pop@ebv = trtH2 * (pop@pheno - meanP(pop)) #ind performance based EBV
varA(pop)
varEBV(pop)
```

---

varG *Total genetic variance*

Description

Returns total genetic variance for all traits

Usage

```
varG(pop)
```

Arguments

`pop`	an object of `Pop-class` or `HybridPop-class`

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
SP$addTraitA(10)
SP$setVarE(h2=0.5)


#Create population
```

```
pop = newPop(founderPop, simParam=SP)
varG(pop)
```

---

varP *Phenotypic variance*

Description

Returns phenotypic variance for all traits

Usage

```
varP(pop)
```

Arguments

`pop`	an object of `Pop-class` or `HybridPop-class`

Examples

```
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=10)

#Set simulation parameters
SP = SimParam$new(founderPop)
SP$addTraitA(10)
SP$setVarE(h2=0.5)


#Create population
pop = newPop(founderPop, simParam=SP)
varP(pop)
```

---

writePlink *Writes a Pop-class as PLINK files*

Description

Writes a Pop-class to PLINK PED and MAP files. The arguments for this function were chosen for consistency with `RRBLUP2`. The base pair coordinate will the locus position as stored in AlphaSimR and not an actual base pair position. This is because AlphaSimR doesn't track base pair positions, only relative positions for the loci used in the simulation.

Usage

```
writePlink(
  pop,
  baseName,
  traits = 1,
  use = "pheno",
  snpChip = 1,
  useQtl = FALSE,
  simParam = NULL,
  ...
)
```

Arguments

`pop`	an object of `Pop-class`
`baseName`	basename for PED and MAP files.
`traits`	an integer indicating the trait to write, a trait name, or a function of the traits returning a single value.
`use`	what to use for PLINK's phenotype field. Either phenotypes "pheno", genetic values "gv", estimated breeding values "ebv", breeding values "bv", or random values "rand".
`snpChip`	an integer indicating which SNP chip genotype to use
`useQtl`	should QTL genotypes be used instead of a SNP chip. If TRUE, snpChip specifies which trait's QTL to use, and thus these QTL may not match the QTL underlying the phenotype supplied in traits.
`simParam`	an object of `SimParam`
`...`	additional arguments if using a function for traits

Examples

```
## Not run:
#Create founder haplotypes
founderPop = quickHaplo(nInd=10, nChr=1, segSites=15)

#Set simulation parameters
SP = SimParam$new(founderPop)
\dontshow{SP$nThreads = 1L}
SP$setSexes(sex="yes_rand")
SP$addTraitA(nQtlPerChr=10)
SP$addSnpChip(nSnpPerChr=5)
SP$setVarE(h2=0.5)

#Create population
pop = newPop(rawPop = founderPop)

# Write out PLINK files
writePlink(pop, baseName="test")

## End(Not run)
```

`writeRecords` *Write data records*

Description

Saves a population's phenotypic and marker data to a directory.

Usage

```
writeRecords(
  pop,
  dir,
  snpChip = 1,
  useQtl = FALSE,
  includeHaplo = FALSE,
  append = TRUE,
  simParam = NULL
)
```

Arguments

`pop`	an object of `Pop-class`
`dir`	path to a directory for saving output
`snpChip`	which SNP chip genotype to save. If useQtl=TRUE, this value will indicate which trait's QTL genotype to save. A value of 0 will skip writing a snpChip.
`useQtl`	should QTL genotype be written instead of SNP chip genotypes.
`includeHaplo`	should markers be separated by female and male haplotypes.
`append`	if true, new records are added to any existing records. If false, any existing records are deleted before writing new records. Note that this will delete all files in the 'dir' directory.
`simParam`	an object of `SimParam`

Index