Package 'MoBPS'

May 7, 2026

Type Package

Title Modular Breeding Program Simulator

Version 1.13.1

Maintainer Torsten Pook <torsten.pook@wur.nl>

Description Framework for the simulation framework for the simulation of complex breeding programs and compare their economic and genetic impact. Associated publication: Pook et al. (2020) <doi:10.1534/g3.120.401193>.

Depends R (>= 3.0),

Imports graphics, stats, utils

License GPL (>= 3)

LazyData TRUE

Suggests EMMREML, BGLR, MASS, doMPI, doRNG, compiler, foreach, sommer, vcfR, jsonlite, rrBLUP, biomaRt, Matrix, doParallel, RColorBrewer, optiSel, alstructure, NAM, gplots, phylogram, cPCG, VariantAnnotation, GenomicRanges, IRanges, MatrixGenerics, Rsamtools, visPedigree, data.table, methods, genio

Enhances miraculix (>= 0.9.10), RandomFieldsUtils (>= 0.5.9), MoBPSmaps

SystemRequirements MiXBLUP (optional; used for multi-trait genomic evaluations); blupf90-family (optional; used for multi-trait genomic evaluations)

Additional_repositories https://tpook92.github.io/drat/

RoxygenNote 7.3.2

Date 2025-10-28

NeedsCompilation no

Author Torsten Pook [aut, cre] (ORCID: <https://orcid.org/0000-0001-7874-8500>), Johannes Geibel [ctb] (ORCID: <https://orcid.org/0000-0001-7172-3263>), Azadeh Hassanpour [ctb] (ORCID: <https://orcid.org/0000-0003-1976-3457>), Tobias Niehoff [ctb] (ORCID: <https://orcid.org/0000-0003-3046-6699>)

Repository CRAN

Date/Publication 2025-10-29 14:10:02 UTC

Contents

add.array *Add a genotyping array*

Description

Function to add a genotyping array for the population

Usage

```
add.array(population, marker.included = TRUE, array.name = NULL)
```

Arguments

population	population list
marker.included	Vector with number of SNP entries coding if each marker is on the array (TRUE/FALSE)
array.name	Name of the added array

Value

Population list

Examples

```
data(ex_pop)
population <- add.array(ex_pop, marker.included = c(TRUE, FALSE), array.name="Half-density")
```

add.combi *Add a trait as a linear combination of other traits*

Description

Function to create an additional trait that is the results of a linear combination of the other traits

Usage

```
add.combi(population, trait, combi.weights, trait.name = NULL)
```

Arguments

population	population list
trait	trait nr. for which to implement a combination of other traits
combi.weights	Weights (only linear combinations of other traits are allowed!)
trait.name	Name of the trait generated

Value

Population list

Population list

Examples

```
data(ex_pop)
population <- creating.trait(ex_pop, n.additive = c(100,100), replace.traits = TRUE)
population <- add.combi(population, trait = 3, combi.weights = c(1,2))
```

`add.diag` *Add something to the diagonal*

Description

Function to add numeric to the diagonal of a matrix

Usage

```
add.diag(M, d)
```

Arguments

`M`	Matrix
`d`	Vector to add to the diagonal of the matrix

Value

Matrix with increased diagonal elements

Matrix with modified diagonal entries

Examples

```
A <- matrix(c(1,2,3,4), ncol=2)
B <- add.diag(A, 5)
```

`add.diversity` *Add additional diverse material to a population*

Description

Function to simulate and add additional diverse material to a population

Usage

```
add.diversity(
  population,
  breeding.size = 100,
  selection.rate = 0.5,
  pool.gen = NULL,
  pool.database = NULL,
  pool.cohorts = NULL,
  target.gen = NULL,
  target.database = NULL,
  target.cohorts = NULL,
  target.value = NULL,
  reduction.multiplier = 5,
  name.cohort = NULL,
  sex.quota = NULL,
  add.gen = 1,
  target.direction = NULL,
  verbose = TRUE,
  store.comp.times = TRUE,
  use.recalculate.manual = FALSE,
  pop1 = NULL,
  export.pop1 = FALSE
)
```

Arguments

`population`	Population list
`breeding.size`	Number of individuals to generate (default: 100)
`selection.rate`	Proportion of individuals to select in each breeding cycle (default: 0.5)
`pool.gen`	Generations of individuals to consider as founder pool to start from (default: NULL)
`pool.database`	Groups of individuals to consider as founder pool to start from (default: NULL)
`pool.cohorts`	Cohorts of individuals to consider as founder pool to start from (default: NULL)
`target.gen`	Generations of individuals to consider to calculate target genomic value to get to (default: NULL)
`target.database`	Groups of individuals to consider to calculate target genomic value to get to (default: NULL)

target.cohorts	Cohorts of individuals to consider to calculate target genomic value to get to (default: NULL)
target.value	Target genomic value to get (default: NULL - calculated based on target.gen/database/cohorts)
reduction.multiplier	Traits that already exceed the target are bred against. Weighting is scaled by this factor.
name.cohort	Name of the newly added cohort
sex.quota	Share of newly added female individuals (default: 0.5)
add.gen	Generation you want to add the new individuals to (default: 1)
target.direction	Manual select with traits are supposed to increase / decrease (1 target high, -1 target low)
verbose	Set to FALSE to not display any prints
store.comp.times	If TRUE store computation times in $info$comp.times.general (default: TRUE)
use.recalculate.manual	Set to TRUE to use recalculate.manual to calculate genomic values (all individuals and traits jointly, default: FALSE)
pop1	Population to start with as founder pool (default: NULL - generation from pool.gen/database/cohorts)
export.pop1	Default: FALSE. Exporting this is helpful if add.diversity is used frequently to avoid initializing this multiple times

Value

population list with newly added individuals

Examples

```
data(ex_pop)
population <- add.diversity(ex_pop, pool.gen  = 1, target.gen = 5)
```

---

add.fixed.effects *Add fixed effects to traits*

Description

Function to add fixed effects to existing set of traits

Usage

```
add.fixed.effects(population, fixed.effects, replace = FALSE)
```

Arguments

population	population list
fixed.effects	Matrix containing fixed effects (p x k -matrix with p being the number of traits and k being number of fixed effects; default: not fixed effects (NULL))
replace	Set to TRUE to delete previously added fixed effects

Value

Population list

Population list

Examples

```
data(ex_pop)
population <- add.fixed.effects(ex_pop, fixed.effects = matrix(c(3,5), nrow=1))
```

---

add.founder.kinship *Add a relationship matrix for founder individuals*

Description

Function to relationship matrix for founder individuals that is used for any calculation of the pedigree

Usage

```
add.founder.kinship(population, founder.kinship = "GBLUP", gen = 1)
```

Arguments

population	population list
founder.kinship	Default is to use vanRaden relationship. Alternative is to enter a pedigree-matrix (order of individuals is first male then female)
gen	Generation for which to enter the pedigree-matrix

Value

Population list

Examples

```
data(ex_pop)
population <- add.founder.kinship(ex_pop)
```

alpha_to_beta *Moore-Penrose-Transfomration*

Description

Internal transformation using Moore-Penrose

Usage

```
alpha_to_beta(alpha, G, Z)
```

Arguments

alpha	alpha
G	kinship-matrix
Z	genomic information matrix

Value

Vector with single marker effects

analyze.bv *Analyze genomic values*

Description

Function to analyze correlation between bv/bve/pheno

Usage

```
analyze.bv(
  population,
  gen = NULL,
  database = NULL,
  cohorts = NULL,
  bvrow = "all",
  advanced = FALSE,
  use.all.copy = FALSE
)
```

Arguments

population	Population list
gen	Quick-insert for database (vector of all generations to export)
database	Groups of individuals to consider for the export
cohorts	Quick-insert for database (vector of names of cohorts to export)
bvrow	Which traits to display
advanced	Set to TRUE to also look at offspring pheno
use.all.copy	Set to TRUE to extract phenotyping

Value

[[1]] Correlation between BV/BVE/Phenotypes [[2]] Genetic variance of the traits

Examples

```
data(ex_pop)
analyze.bv(ex_pop,gen=1)
```

---

analyze.population *Analyze allele frequency of a single marker*

Description

Function to analyze allele frequency of a single marker

Usage

```
analyze.population(
  population,
  chromosome = NULL,
  snp = NULL,
  snp.name = NULL,
  database = NULL,
  gen = NULL,
  cohorts = NULL
)
```

Arguments

population	Population list
chromosome	Number of the chromosome of the relevant SNP
snp	Number of the relevant SNP
snp.name	Name of the SNP to analyze
database	Groups of individuals to consider for the export
gen	Quick-insert for database (vector of all generations to export)
cohorts	Quick-insert for database (vector of names of cohorts to export)

Value

Frequency of AA/AB/BB in selected gen/database/cohorts

Examples

```
data(ex_pop)
analyze.population(ex_pop, snp=1, chromosome=1, gen=1:5)
```

`bit.snps` *Decoding of bitwise-storing in R*

Description

Function for decoding in bitwise-storing in R (only 30 of 32 bits are used!)

Usage

```
bit.snps(bit.seq, nbits, population = NULL, from.p.bit = 1)
```

Arguments

`bit.seq`	bitwise stored SNP sequence
`nbits`	Number of usable bits (default: 30)
`population`	Population list
`from.p.bit`	Bit to start on

Value

De-coded marker sequence

`bit.storing` *Bitwise-storing in R*

Description

Function for bitwise-storing in R (only 30 of 32 bits are used!)

Usage

```
bit.storing(snpseq, nbits)
```

Arguments

`snpseq`	SNP sequence
`nbits`	Number of usable bits (default: 30)

Value

Bit-wise coded marker sequence

`breeding.diploid` *Breeding function*

Description

Function to simulate a step in a breeding scheme

Usage

```
breeding.diploid(
  population,
  selection.size = 0,
  selection.criteria = NULL,
  selection.m.gen = NULL,
  selection.f.gen = NULL,
  selection.m.database = NULL,
  selection.f.database = NULL,
  selection.m.cohorts = NULL,
  selection.f.cohorts = NULL,
  max.selection.fullsib = Inf,
  max.selection.halfsib = Inf,
  class.m = 0,
  class.f = 0,
  add.class.cohorts = TRUE,
  multiple.bve = "add",
  selection.index.weights.m = NULL,
  selection.index.weights.f = NULL,
  selection.index.scale.m = NULL,
  selection.index.scale.f = NULL,
  selection.index.kinship = 0,
  selection.index.gen = NULL,
  selection.index.database = NULL,
  selection.index.cohorts = NULL,
  selection.highest = TRUE,
  ignore.best = 0,
  best.selection.ratio.m = 1,
  best.selection.ratio.f = NULL,
  best.selection.criteria.m = "bv",
  best.selection.criteria.f = NULL,
  best.selection.manual.ratio.m = NULL,
  best.selection.manual.ratio.f = NULL,
  best.selection.manual.reorder = TRUE,
  selection.m.random.prob = NULL,
```

```
selection.f.random.prob = NULL,
reduced.selection.panel.m = NULL,
reduced.selection.panel.f = NULL,
threshold.selection.index = NULL,
threshold.selection.value = NULL,
threshold.selection.sign = ">",
threshold.selection.criteria = NULL,
threshold.selection = NULL,
threshold.sign = ">",
remove.duplicates = TRUE,
selection.m.miesenberger = FALSE,
selection.f.miesenberger = NULL,
selection.miesenberger.reliability.est = "derived",
miesenberger.trafo = 0,
sort.selected.pos = FALSE,
ogc = FALSE,
relationship.matrix.ogc = "pedigree",
depth.pedigree.ogc = 7,
ogc.target = "min.sKin",
ogc.uniform = NULL,
ogc.ub = NULL,
ogc.lb = NULL,
ogc.ub.sKin = NULL,
ogc.lb.BV = NULL,
ogc.ub.BV = NULL,
ogc.eq.BV = NULL,
ogc.ub.sKin.increase = NULL,
ogc.lb.BV.increase = NULL,
ogc.c1 = NULL,
ogc.isCandidate = NULL,
ogc.plots = TRUE,
ogc.weight = NULL,
ogc.freq = NULL,
selection.skip = FALSE,
breeding.size = 0,
breeding.size.litter = NULL,
name.cohort = NULL,
breeding.sex = NULL,
breeding.sex.random = FALSE,
sex.s = NULL,
add.gen = 0,
share.genotyped = 0,
phenotyping.child = NULL,
fixed.effects.p = NULL,
fixed.effects.freq = NULL,
new.class = 0L,
max.offspring = Inf,
max.offspring.individual.m = NULL,
```

```
max.offspring.individual.f = NULL,
max.offspring.individual.gen = NULL,
max.offspring.individual.database = NULL,
max.offspring.individual.cohorts = NULL,
max.litter = Inf,
max.mating.pair = Inf,
avoid.mating.fullsib = FALSE,
avoid.mating.halfsib = FALSE,
avoid.mating.parent = FALSE,
avoid.mating.inb = NULL,
avoid.mating.inb.quantile = NULL,
avoid.mating.inb.min = 0,
avoid.mating.kinship = NULL,
avoid.mating.kinship.quantile = NULL,
avoid.mating.kinship.gen = NULL,
avoid.mating.kinship.database = NULL,
avoid.mating.kinship.cohorts = NULL,
avoid.mating.kinship.min = 0,
avoid.mating.kinship.median = FALSE,
avoid.mating.depth.pedigree = 7,
avoid.mating.remove = FALSE,
avoid.mating.ignore = 0,
avoid.mating.resampling = 1000,
fixed.breeding = NULL,
fixed.breeding.best = NULL,
fixed.breeding.id = NULL,
fixed.assignment = FALSE,
breeding.all.combination = FALSE,
repeat.mating = NULL,
repeat.mating.copy = NULL,
repeat.mating.fixed = NULL,
repeat.mating.overwrite = TRUE,
repeat.mating.trait = 1,
repeat.mating.max = NULL,
repeat.mating.s = NULL,
same.sex.activ = FALSE,
same.sex.sex = 0.5,
same.sex.selfing = FALSE,
selfing.mating = FALSE,
selfing.sex = 0.5,
dh.mating = FALSE,
dh.sex = 0.5,
combine = FALSE,
copy.individual = FALSE,
copy.individual.m = FALSE,
copy.individual.f = FALSE,
copy.individual.keep.bve = TRUE,
copy.individual.keep.pheno = TRUE,
```

```
  added.genotyped = NULL,
  bv.ignore.traits = NULL,
  generation.cores = NULL,
  generation.core.make.small = FALSE,
  pedigree.error = 0,
  pedigree.unknown = 0,
  genotyped.database = NULL,
  genotyped.gen = NULL,
  genotyped.cohorts = NULL,
  genotyped.share = 1,
  genotyped.array = 1,
  genotyped.remove.gen = NULL,
  genotyped.remove.database = NULL,
  genotyped.remove.cohorts = NULL,
  genotyped.remove.all.copy = TRUE,
  genotyped.selected = FALSE,
  phenotyping = NULL,
  phenotyping.gen = NULL,
  phenotyping.cohorts = NULL,
  phenotyping.database = NULL,
  n.observation = NULL,
  phenotyping.class = NULL,
  heritability = NULL,
  repeatability = NULL,
  multiple.observation = FALSE,
  phenotyping.selected = FALSE,
  share.phenotyped = 1,
  offpheno.parents.gen = NULL,
  offpheno.parents.database = NULL,
  offpheno.parents.cohorts = NULL,
  offpheno.offspring.gen = NULL,
  offpheno.offspring.database = NULL,
  offpheno.offspring.cohorts = NULL,
  sigma.e = NULL,
  sigma.e.gen = NULL,
  sigma.e.cohorts = NULL,
  sigma.e.database = NULL,
  new.residual.correlation = NULL,
  new.breeding.correlation = NULL,
  phenotyping.trafo.parameter = NULL,
  bve = FALSE,
  bve.gen = NULL,
  bve.cohorts = NULL,
  bve.database = NULL,
  relationship.matrix = "GBLUP",
  depth.pedigree = 7,
  singlestep.active = TRUE,
  bve.ignore.traits = NULL,
```

```
bve.array = NULL,
bve.imputation = TRUE,
bve.imputation.errorrate = 0,
bve.all.genotyped = FALSE,
bve.insert.gen = NULL,
bve.insert.cohorts = NULL,
bve.insert.database = NULL,
variance.correction = "none",
bve.class = NULL,
sigma.g = NULL,
sigma.g.gen = NULL,
sigma.g.cohorts = NULL,
sigma.g.database = NULL,
forecast.sigma.g = NULL,
remove.effect.position = FALSE,
estimate.add.gen.var = FALSE,
estimate.pheno.var = FALSE,
bve.avoid.duplicates = TRUE,
calculate.reliability = FALSE,
estimate.reliability = FALSE,
bve.input.phenotype = "own",
mas.bve = FALSE,
mas.markers = NULL,
mas.number = 5,
mas.effects = NULL,
mas.geno = NULL,
bve.parent.mean = FALSE,
bve.grandparent.mean = FALSE,
bve.mean.between = "bvepheno",
bve.exclude.fixed.effects = NULL,
bve.beta.hat.approx = TRUE,
bve.per.sample.sigma.e = TRUE,
bve.p_i.list = NULL,
bve.p_i.gen = NULL,
bve.p_i.database = NULL,
bve.p_i.cohorts = NULL,
bve.p_i.exclude.nongenotyped = FALSE,
bve.use.all.copy = FALSE,
bve.pedigree.error = TRUE,
mobps.bve = TRUE,
mixblup.bve = FALSE,
blupf90.bve = FALSE,
mixblup.reliability = FALSE,
emmreml.bve = FALSE,
rrblup.bve = FALSE,
sommer.bve = FALSE,
sommer.multi.bve = FALSE,
BGLR.bve = FALSE,
```

```
pseudo.bve = FALSE,
pseudo.bve.accuracy = 1,
bve.solve = "exact",
mixblup.jeremie = FALSE,
mixblup.hpblup = FALSE,
mixblup.pedfile = TRUE,
mixblup.parfile = TRUE,
mixblup.datafile = TRUE,
mixblup.inputfile = TRUE,
mixblup.genofile = TRUE,
mixblup.path = NULL,
mixblup.path.pedfile = NULL,
mixblup.path.parfile = NULL,
mixblup.path.datafile = NULL,
mixblup.path.inputfile = NULL,
mixblup.path.genofile = NULL,
mixblup.full.path.genofile = NULL,
mixblup.files = "MiXBLUP_files",
mixblup.verbose = TRUE,
blupf90.verbose = TRUE,
mixblup.genetic.cov = NULL,
mixblup.residual.cov = NULL,
mixblup.lambda = 1,
mixblup.alpha = NULL,
mixblup.beta = NULL,
mixblup.omega = NULL,
mixblup.maxit = 5000,
mixblup.stopcrit = NULL,
mixblup.maf = 0.005,
mixblup.numproc = NULL,
mixblup.apy = FALSE,
mixblup.apy.core = NULL,
mixblup.ta = FALSE,
mixblup.tac = FALSE,
mixblup.skip = FALSE,
blupf90.skip = FALSE,
mixblup.restart = FALSE,
mixblup.nopeek = FALSE,
mixblup.calcinbr.s = FALSE,
mixblup.multiple.records = FALSE,
mixblup.attach = FALSE,
mixblup.debug = FALSE,
mixblup.plink = FALSE,
mixblup.cleanup = Inf,
blupf90.pedfile = TRUE,
blupf90.parfile = TRUE,
blupf90.datafile = TRUE,
blupf90.inputfile = TRUE,
```

```
blupf90.genofile = TRUE,
mixblup.dgv = FALSE,
mixblup.dgv.freq = NULL,
mixblup.dgv.effect = NULL,
blupf90.path = NULL,
renumf90.path = NULL,
blupf90.path.pedfile = NULL,
blupf90.path.parfile = NULL,
blupf90.path.datafile = NULL,
blupf90.path.inputfile = NULL,
blupf90.path.genofile = NULL,
blupf90.files = "blupf90_files",
blupf90.blksize = NULL,
blupf90.no.quality = FALSE,
blupf90.conv_crit = NULL,
BGLR.model = "RKHS",
BGLR.burnin = 500,
BGLR.iteration = 5000,
BGLR.print = TRUE,
BGLR.save = "RKHS",
BGLR.save.random = FALSE,
miraculix = NULL,
miraculix.cores = 1,
miraculix.mult = NULL,
miraculix.chol = TRUE,
miraculix.destroyA = TRUE,
estimate.u = FALSE,
fast.uhat = TRUE,
gwas.u = FALSE,
approx.residuals = TRUE,
gwas.gen = NULL,
gwas.cohorts = NULL,
gwas.database = NULL,
gwas.group.standard = FALSE,
y.gwas.used = "pheno",
gene.editing.offspring = FALSE,
gene.editing.best = FALSE,
gene.editing.offspring.sex = TRUE,
gene.editing.best.sex = TRUE,
nr.edits = 0,
culling.non.selected = FALSE,
culling.gen = NULL,
culling.database = NULL,
culling.cohorts = NULL,
culling.type = 0,
culling.time = Inf,
culling.name = "Not_named",
culling.bv1 = 0,
```

```
culling.share1 = NULL,
culling.bv2 = NULL,
culling.share2 = NULL,
culling.index = 0,
culling.single = TRUE,
culling.all.copy = TRUE,
mutation.rate = 10^-8,
remutation.rate = 10^-8,
recombination.rate = 1,
recombination.rate.trait = 0,
recombination.function = NULL,
recombination.minimum.distance = NULL,
recombination.distance.penalty = NULL,
recombination.distance.penalty.2 = NULL,
recom.f.indicator = NULL,
import.position.calculation = NULL,
duplication.rate = 0,
duplication.length = 0.01,
duplication.recombination = 1,
gen.architecture.m = 0,
gen.architecture.f = NULL,
add.architecture = NULL,
intern.func = 0,
delete.haplotypes = NULL,
delete.recombi = NULL,
delete.recombi.only.non.genotyped = FALSE,
delete.recombi.class = NULL,
delete.individuals = NULL,
delete.gen = NULL,
delete.sex = 1:2,
delete.same.origin = FALSE,
save.recombination.history = FALSE,
store.sparse = FALSE,
storage.save = 1.05,
verbose = TRUE,
report.accuracy = TRUE,
store.breeding.totals = FALSE,
store.bve.data = FALSE,
store.comp.times = TRUE,
store.comp.times.bve = TRUE,
store.comp.times.generation = TRUE,
store.effect.freq = FALSE,
Rprof = FALSE,
randomSeed = NULL,
display.progress = NULL,
time.point = 0,
age.point = NULL,
creating.type = 0,
```

```
  import.relationship.matrix = NULL,
  export.selected = FALSE,
  export.selected.database = FALSE,
  export.relationship.matrix = FALSE,
  pen.assignments = NULL,
  pen.size = NULL,
  pen.by.sex = TRUE,
  pen.by.litter = FALSE,
  pen.size.overwrite = TRUE,
  selection.m = NULL,
  selection.f = NULL,
  new.bv.observation.gen = NULL,
  new.bv.observation.cohorts = NULL,
  new.bv.observation.database = NULL,
  best1.from.group = NULL,
  best2.from.group = NULL,
  best1.from.cohort = NULL,
  best2.from.cohort = NULL,
  new.bv.observation = NULL,
  reduce.group = NULL,
  reduce.group.selection = "random",
  new.bv.child = NULL,
  computation.A = NULL,
  computation.A.ogc = NULL,
  new.phenotype.correlation = NULL,
  offspring.bve.parents.gen = NULL,
  offspring.bve.parents.database = NULL,
  offspring.bve.parents.cohorts = NULL,
  offspring.bve.offspring.gen = NULL,
  offspring.bve.offspring.database = NULL,
  offspring.bve.offspring.cohorts = NULL,
  input.phenotype = NULL,
  multiple.bve.weights.m = 1,
  multiple.bve.weights.f = NULL,
  multiple.bve.scale.m = "bv_sd",
  multiple.bve.scale.f = NULL,
  use.recalculate.manual = NULL,
  recalculate.manual.subset = 5000,
  compute.grandparent.contribution = FALSE,
  size.scaling = NULL,
  parallel.internal = FALSE,
  varg = FALSE,
  gain.stats = FALSE,
  next.id = NULL,
  copy.individual.use = NULL,
  copy.individual.use2 = NULL
)
```

Arguments

population	Population list
selection.size	Number of selected individuals as parents (default: all individuals in selection.m/f.gen/database/gen - alt: positive numbers)
selection.criteria	What to use in the selection process (default: "bve", alt: "bv", "pheno", "random", "offpheno")
selection.m.gen, selection.m.cohorts, selection.m.database	Generations/cohorts/groups available for selection of first/paternal parent
selection.f.gen, selection.f.cohorts, selection.f.database	Generations available for selection of maternal parent
max.selection.fullsib	Maximum number of individual to select from the same family (same sire & dam)
max.selection.halfsib	Maximum number of individual to select from the same family (same sire or same dam)
class.m, class.f	For selection only individuals from this class (included in selection.m/f.gen/database/cohorts) will be considered for selection (default: 0 - which is all individuals if never used class elsewhere)
add.class.cohorts	Initial classes of cohorts used in selection.m/f.cohorts are automatically added to class.m/f (default: TRUE)
multiple.bve	Way to handle multiple traits in selection (default: "add" - use values directly in an index, alt: "ranking" - ignore values but only use ranking per trait)
selection.index.weights.m, selection.index.weights.f	Weighting between traits (default: 1)
selection.index.scale.m, selection.index.scale.f	Default: "bv_sd"; Set to "pheno_sd" when using gains per phenotypic SD, "unit" when using gains per unit, "bve" when using estimated breeding values
selection.index.kinship	Include avg. kinship to a reference population (selection.index.gen/database/cohorts) as part of the selection index (Default: 0)
selection.index.gen, selection.index.database, selection.index.cohorts	Generation/cohorts/groups to use as a reference of the kinship value in the selection index
selection.highest	If FALSE to select individuals with lowest value for the selection criterium (default c(TRUE,TRUE) - (m,w))
ignore.best	Not consider the top individuals of the selected individuals (e.g. to use 2-10 best individuals)
best.selection.ratio.m, best.selection.ratio.f	Ratio of the frequency of the selection of the best best individual and the worst best individual (default=1)

`best.selection.criteria.m`, `best.selection.criteria.f`
: Criteria to calculate this ratio (default: "bv", alt: "bve", "pheno")

`best.selection.manual.ratio.m`, `best.selection.manual.ratio.f`
: vector containing probability to draw from for every individual (e.g. c(0.1,0.2,0.7))

`best.selection.manual.reorder`
: Set to FALSE to not use the order from best to worst selected individual but plain order based on database-order

`selection.m.random.prob`, `selection.f.random.prob`
: Use this parameter to control the probability of each individual to be selected when doing random selection

`reduced.selection.panel.m`, `reduced.selection.panel.f`
: Use only a subset of individuals of the potential selected ones ("Split in user-interface")

`threshold.selection.index`
: Selection index on which to access (matrix which one index per row)

`threshold.selection.value`
: Minimum value in the selection index selected individuals have to have

`threshold.selection.sign`
: Pick all individuals above (">") the threshold. Alt: ("<", "=", "<=", ">=")

`threshold.selection.criteria`
: Criterium on which to evaluate the index (default: "bve", alt: "bv", "pheno")

`threshold.selection`
: Minimum value in the selection index selected individuals have to have

`threshold.sign` Pick all individuals above (">") the threshold. Alt: ("<", "=", "<=", ">=")

`remove.duplicates`
: Set to FALSE to select the same individual multiple times when the gen/database/cohorts for selection contains it multiple times

`selection.m.miesenberger`, `selection.f.miesenberger`
: Use Weighted selection index according to Miesenberger 1997 for paternal/maternal selection

`selection.miesenberger.reliability.est`
: If available reliability estimated are used. If not use default: "derived" (cor(BVE,BV)^2) , alt: "heritability", "estimated" (SD BVE / SD Pheno) as replacement

`miesenberger.trafo`
: Ignore all eigenvalues below this threshold and apply dimension reduction (default: 0 - use all)

`sort.selected.pos`
: Set to TRUE to arrange selected individuals according to position in the database (not by breeding value)

`ogc` If TRUE use optimal genetic contribution theory to perform selection (This requires the use of the R-package optiSel)

`relationship.matrix.ogc`
: Method to calculate relationship matrix for OGC (Default: "pedigree", alt: "van-Raden", "CE", "non_stand", "CE2", "CM")

`depth.pedigree.ogc`
: Depth of the pedigree in generations (default: 7)

`ogc.target` Target of OGC (default: "min.sKin" - minimize inbreeding; alt: "max.BV" / "min.BV" - maximize genetic gain; both under constrains selected below)

`ogc.uniform` This corresponds to the uniform constrain in optiSel

`ogc.ub` This corresponds to the ub constrain in optiSel

`ogc.lb` This corresponds to the lb constrain in optiSel

`ogc.ub.sKin` This corresponds to the ub.sKin constrain in optiSel

`ogc.lb.BV` This corresponds to the lb.BV constrain in optiSel

`ogc.ub.BV` This corresponds to the ub.BV constrain in optiSel

`ogc.eq.BV` This corresponds to the eq.BV constrain in optiSel

`ogc.ub.sKin.increase`
This corresponds to the upper bound (current sKin + ogc.ub.sKin.increase) as ub.sKin in optiSel

`ogc.lb.BV.increase`
This corresponds to the lower bound (current BV + ogc.lb.BV.increase) as lb.BV in optiSel

`ogc.c1` Only applicable when TN-version of OGC is available

`ogc.isCandidate`
Only applicable when TN-version of OGC is available

`ogc.plots` Only applicable when TN-version of OGC is available

`ogc.weight` Only applicable when B4F-version of OGC is available

`ogc.freq` Only applicable when B4F-version of OGC is available

`selection.skip` Set to FALSE in case no selection of individuals should be performed (just skips some unneccessary computations)

`breeding.size` Number of individuals to generate (default: 0, use vector with two entries to control offspring per sex)

`breeding.size.litter`
Number of litters to generate (default: NULL - use breeding.size; only single positive number input allowed)

`name.cohort` Name of the newly added cohort

`breeding.sex` Share of female individuals (if single value is used for breeding size; default: 0.5)

`breeding.sex.random`
If TRUE randomly chose sex of new individuals (default: FALSE - use expected values)

`sex.s` Specify which newly added individuals are male (1) or female (2)

`add.gen` Generation you want to add the new individuals to (default: New generation)

`share.genotyped`
Share of individuals newly generated individuals that are genotyped (Default: 0). Also applies if individuals are copied with copy.individual

`phenotyping.child`
Starting phenotypes of newly generated individuals (default: "zero", alt: "mean" of both parents, "obs" - regular observation)

`fixed.effects.p`
: Parametrization for the fixed effects (default: c(0,0..,0), if multiple different parametrizations are possible use a matrix with one parametrization per row)

`fixed.effects.freq`
: Frequency of each different parametrization of the fixed effects

`new.class`
: Migration level of newly generated individuals (default: 0 / use vector for different classes for different sexes)

`max.offspring`
: Maximum number of offspring per individual (default: c(Inf,Inf) - (m,w))

`max.offspring.individual.m`, `max.offspring.individual.f`
: Vector with maximum number of offspring for first/second parent (default: NULL). Order in the vector by order of selection

`max.offspring.individual.gen`
: matrix with first column generation with limited number offspring, second column number of allowed offspring

`max.offspring.individual.database`
: matrix with first four columns database with limited number offspring, fifth column number of allowed offspring

`max.offspring.individual.cohorts`
: matrix with first column cohort with limited number offspring, second column number of allowed offspring

`max.litter`
: Maximum number of litters per individual (default: c(Inf,Inf) - (m,w))

`max.mating.pair`
: Maximum number of matings between two specific individuals (default: Inf)

`avoid.mating.fullsib`
: Set to TRUE to not generate offspring of full siblings

`avoid.mating.halfsib`
: Set to TRUE to not generate offspring from half or full siblings

`avoid.mating.parent`
: Set to TRUE to not generate offspring from parent / sibling matings

`avoid.mating.inb`
: Maximum allowed expected inbreeding to allow a mating combination (based on kinships)

`avoid.mating.inb.quantile`
: Use this to not perform mating between more related potential parents (quantile of all expected inbreeding levels)

`avoid.mating.inb.min`, `avoid.mating.kinship.min`
: Share of mating to at minimum perform for each individual (default: 0)

`avoid.mating.kinship`
: Maximum allowed expected kinship of an offspring to a reference group of individuals

`avoid.mating.kinship.quantile`
: Maximum allowed expected kinship of an offspring to a reference group of individuals

`avoid.mating.kinship.gen`, `avoid.mating.kinship.database`, `avoid.mating.kinship.cohorts`
: Gen/database/cohorts of individuals to consider as a reference pool in avoid.mating.kinship

`avoid.mating.kinship.median`
: Set to TRUE to use median kinship instead of mean kinship in avoid.mating.kinship (default: FALSE)

`avoid.mating.depth.pedigree`
: Depth of the pedigree to calculate expected inbreeding levels / kinships

`avoid.mating.remove`
: Set to TRUE to automatically exclude any selected individuals from the sample of parents

`avoid.mating.ignore`
: Set to value higher 0 for avoid.mating.inb/kinship restrictions to not always be applied

`avoid.mating.resampling`
: Number of sampling attempts to avoid unwanted matings ((last couple of individuals otherwise can have unwanted relatedness, default = 1000))

`fixed.breeding`
: Set of targeted matings to perform (matrix with 7 columns: database position first parent (gen, sex, nr), database position second parent (gen,sex,nr), likelihood to be female (optional))

`fixed.breeding.best`
: Perform targeted matings in the group of selected individuals (matrix with 5 columns: position first parent (male/female pool of selected individuals, ranking in selected animals), position second parent (male/female pool of selected individuals, ranking in selected animals), likelihood to be female (optional))

`fixed.breeding.id`
: Set of target matings to perform (matrix with 3 columns: id first parent, id second parent, likelihood to be female (optional))

`fixed.assignment`
: Set to "bestbest" / TRUE for targeted mating of best-best individual till worst-worst (of selected). set to "bestworst" for best-worst mating

`breeding.all.combination`
: Set to TRUE to automatically perform each mating combination possible exactly ones.

`repeat.mating`
: Generate multiple mating from the same dam/sire combination (first column: number of offspring; second column: probability)

`repeat.mating.copy`
: Generate multiple copies from a copy action (combine / copy.individual.m/f) (first column: number of offspring; second column: probability)

`repeat.mating.fixed`
: Vector containing number of times each mating is repeated. This will overwrite sampling from repeat.mating / repeat.mating.copy (default: NULL)

`repeat.mating.overwrite`
: Set to FALSE to not use the current repeat.mating / repeat.mating.copy input as the new standard values (default: TRUE)

`repeat.mating.trait`
: Trait that should be linked to the litter size

`repeat.mating.max`
: Maximum number of individuals in a litter

repeat.mating.s
: Use this parameter to manually provide the size of each litter generated

same.sex.activ
: If TRUE allow matings of individuals of same sex (Sex here is a general term with the first sex referring to the first parent, second sex second parent)

same.sex.sex
: Probability to use female individuals as parents (default: 0.5)

same.sex.selfing
: Set to TRUE to allow for selfing when using same.sex matings (default: FALSE)

selfing.mating
: If TRUE generate new individuals via selfing

selfing.sex
: Share of female individuals used for selfing (default: 0.5)

dh.mating
: If TRUE generate a DH-line in mating process

dh.sex
: Share of DH-lines generated from selected female individuals

combine
: Copy existing individuals (e.g. to merge individuals from different groups in a joined cohort). Individuals to use are used as the first parent

copy.individual
: Set TRUE to generate a copy of an already existing individual. If only one of the sexes has individuals to select from it will automatically detect with sex to chose. Otherwise the first/male parent will be copied

copy.individual.m, copy.individual.f
: If TRUE generate exactly one copy of all selected male/female in a new cohort (or more by setting breeding.size)

copy.individual.keep.bve
: Set to FALSE to not keep estimated breeding value in case of use of copying individuals instead of regular meiosis

copy.individual.keep.pheno
: Set to FALSE to not keep phenotypes in case of use of copying individuals instead of regular meiosis

added.genotyped
: (OLD! use share.genotyped) Share of individuals that is additionally genotyped (only for copy.individual, default: 0)

bv.ignore.traits
: Vector of traits to ignore in the calculation of the genomic value (default: NULL; Only recommended for high number of traits and experienced users!)

generation.cores
: Number of cores used for the generation of new individuals (This will only be active when generating more than 500 individuals)

generation.core.make.small
: Set to TRUE to delete not necessary individuals during parallelization

pedigree.error
: Share of errors in the pedigree (default: 0; vector with two entries for errors on male/female side)

pedigree.unknown
: Share of individuals with unknown parents (default: 0; vector with two entries for differences in unknown-share between male/female side)

genotyped.gen, genotyped.cohorts, genotyped.database
: Generations/cohorts/groups to generate genotype data (that can be used in a BVE)

genotyped.share
: Share of individuals in genotyped.gen/database/cohort to generate genotype data from (default: 1)

genotyped.array
: Genotyping array used

genotyped.remove.gen, genotyped.remove.database, genotyped.remove.cohorts
: Generations/cohorts/groups from which to remove genotyping information (this will affect all copies of an individual unless genotyped.remove.all.copy is set to FALSE)

genotyped.remove.all.copy
: Set to FALSE to only change the genotyping state of this particular copy of an individual (default: TRUE)

genotyped.selected
: Set to TRUE to genotype all selected individuals

phenotyping
: Quick access to phenotyping for (all: "all", non-phenotyped: "non_obs", non-phenotyped male: "non_obs_m", non-phenotyped female: "non_obs_f")

phenotyping.gen, phenotyping.cohorts, phenotyping.database
: Generations/cohorts/groups from which to generate additional phenotypes

n.observation
: Number of phenotypic observations generated per trait and per individuals (use repeatability to control correlation between observations)

phenotyping.class
: Classes of individuals for which to generate phenotypes (default: NULL –> all classes)

heritability
: Use sigma.e to obtain a certain heritability (default: NULL)

repeatability
: Set this to control the share of the residual variance (sigma.e) that is permanent (there for each observation)

multiple.observation
: If an already phenotyped trait is phenotyped again this will on NOT lead to an additional phenotyped observation unless this is set to TRUE

phenotyping.selected
: Set to TRUE to phenotype all selected individuals

share.phenotyped
: Share of the individuals to phenotype (use vector for different probabilities for different traits)

offpheno.parents.gen, offpheno.parents.database, offpheno.parents.cohorts
: Generations/groups/cohorts to consider to derive phenotype from offspring phenotypes

offpheno.offspring.gen, offpheno.offspring.cohorts, offpheno.offspring.database
: Active generations/cohorts/groups for import of offspring phenotypes

sigma.e
: Enviromental standard deviation (default: use sigma.e from last run / usually fit by use of heritability; if never provided: 10; used in BVE for variance components if manually set)

sigma.e.gen, sigma.e.cohorts, sigma.e.database
: Generations/cohorts/groups to consider when estimating sigma.e when using heritability

new.residual.correlation
: Correlation of the simulated residual variance

new.breeding.correlation
: Correlation of the simulated genetic variance (only impacts non-QTL based traits. Needs to be fit in creating.diploid/trait for QTL-based traits)

phenotyping.trafo.parameter
: Additional input parameter for phenotypic transformation function

bve
: If TRUE perform a breeding value estimation (default: FALSE)

bve.gen, bve.cohorts, bve.database
: Generations/Groups/Cohorts of individuals to consider in breeding value estimation (default: NULL)

relationship.matrix
: Method to calculate relationship matrix for the breeding value estimation. This will automatically chosen between GBLUP, ssGBLUP, pBLUP based on if genotyped individuals are available (Default: "GBLUP", alt: "pedigree", "CE", "non_stand", "CE2", "CM")

depth.pedigree
: Depth of the pedigree in generations (default: 7)

singlestep.active
: Set FALSE remove all individuals without genomic data from the breeding value estimation

bve.ignore.traits
: Vector of traits to ignore in the breeding value estimation (default: NULL, use: "zero" to not consider traits with 0 index weight in selection.index.weights.m/.w)

bve.array
: Array to use in the breeding value estimation (default: NULL; chose largest possible based on used individuals in BVE)

bve.imputation
: Set to FALSE to not perform imputation up to the highest marker density of genotyping data that is available

bve.imputation.errorrate
: Share of errors in the imputation procedure (default: 0)

bve.all.genotyped
: Set to TRUE to act as if every individual in the breeding value estimation has been genotyped

bve.insert.gen, bve.insert.cohorts, bve.insert.database
: Generations/Groups/Cohorts of individuals to compute breeding values for (default: all groups in bve.database)

variance.correction
: Correct for "parental.mean" or "generation.mean" in the estimation of sigma.g for BVE / sigma.e estimation (default: "none")

bve.class
: Consider only individuals of those class classes in breeding value estimation (default: NULL - use all)

sigma.g
: Genetic standard deviation (default: calculated based on individuals in BVE ; used in BVE for variance components if manually set; mostly recommended to be used for non-QTL based traits)

`sigma.g.gen`, `sigma.g.cohorts`, `sigma.g.database`
: Generations/cohorts/groups to consider when estimating sigma.g

`forecast.sigma.g`
: Set FALSE to not estimate sigma.g (Default: TRUE // in case sigma.g is set this is automatically set to FALSE)

`remove.effect.position`
: If TRUE remove real QTLs in breeding value estimation

`estimate.add.gen.var`
: If TRUE estimate additive genetic variance and heritability based on parent model

`estimate.pheno.var`
: If TRUE estimate total variance in breeding value estimation

`bve.avoid.duplicates`
: If set to FALSE multiple generations of the same individual can be used in the bve (only possible by using copy.individual to generate individuals)

`calculate.reliability`
: Set TRUE to calculate a reliability when performing Direct-Mixed-Model BVE

`estimate.reliability`
: Set TRUE to estimate the reliability in the BVE by calculating the correlation between estimated and real breeding values

`bve.input.phenotype`
: Select what to use in BVE (default: own phenotype ("own"), offspring phenotype ("off"), their average ("mean") or a weighted average ("weighted"))

`mas.bve`
: If TRUE use marker assisted selection in the breeding value estimation

`mas.markers`
: Vector containing markers to be used in marker assisted selection

`mas.number`
: If no markers are provided this nr of markers is selected (if single marker QTL are present highest effect markers are prioritized)

`mas.effects`
: Effects assigned to the MAS markers (Default: estimated via lm())

`mas.geno`
: Genotype dataset used in MAS (default: NULL, automatic internal calculation)

`bve.parent.mean`
: Set to TRUE to use the average parental performance as the breeding value estimate

`bve.grandparent.mean`
: Set to TRUE to use the average grandparental performance as the breeding value estimate

`bve.mean.between`
: Select if you want to use the "bve", "bv", "pheno" or "bvepheno" to form the mean (default: "bvepheno" - if available bve, else pheno)

`bve.exclude.fixed.effects`
: Vector of fixed effects to ignore in the BVE (default: NULL)

`bve.beta.hat.approx`
: Set to FALSE to use the true underlying value for beta_hat for the fixed effect in the direct BVE model. rrBLUP, BGLR, sommer will always estimate beta_hat.

`bve.per.sample.sigma.e`
: Set to FALSE to deactivate the use of a heritability based on the number of observations generated per sample

bve.p_i.list Vector of allele frequencies to be used when calculating the genomic relationship matrix (default: calculate them based on Z)

bve.p_i.gen, bve.p_i.database, bve.p_i.cohorts
Generations/cohorts/groups to use when manually calculating allele frequencies for genomic relationship matrix

bve.p_i.exclude.nongenotyped
Set to TRUE to exclude non-genotyped individuals when calculating allele frequencies for genomic relationship matrix standardization

bve.use.all.copy
Set to TRUE to use phenotypes and genotyped status from all copies of an individual instead of just the provided ones in the bve.gen/database/cohorts (default: FALSE)

bve.pedigree.error
Set to FALSE to ignore/correct for any pedigree errors

mobps.bve If TRUE predict BVEs in direct estimation with assumed known heritability (default: TRUE; activating use of any other BVE method to TRUE will overwrite this)

mixblup.bve Set to TRUE to activate breeding value estimation via MiXBLUP (requires MiXBLUP license!)

blupf90.bve Set to TRUE to activate breeding value estimation via BLUPF90 (requires blupf90 software!)

mixblup.reliability
Set to TRUE to activate breeding value estimation via MiXBLUP (requires MiXBLUP license!)

emmreml.bve If TRUE use REML estimator from R-package EMMREML in breeding value estimation

rrblup.bve If TRUE use REML estimator from R-package rrBLUP in breeding value estimation

sommer.bve If TRUE use REML estimator from R-package sommer in breeding value estimation

sommer.multi.bve
Set TRUE to use a multi-trait model in the R-package sommer for BVE

BGLR.bve If TRUE use BGLR to perform breeding value estimation

pseudo.bve If set to TRUE the breeding value estimation will be simulated with resulting accuracy pseudo.bve.accuracy (default: 1)

pseudo.bve.accuracy
The accuracy to be obtained in the "pseudo" - breeding value estimation

bve.solve Provide solver to be used in BVE (default: "exact" solution via inversion, alt: "pcg", function with inputs A, b and output y_hat)

mixblup.jeremie
Set to TRUE to use Jeremies suggested MiXBLUP settings

mixblup.hpblup Set to TRUE to use hpblup in MiXBLUP (default: FALSE)

mixblup.pedfile
Set to FALSE to manually generate your MiXBLUP pedfile

`mixblup.parfile`
: Set to FALSE to manually generate your MiXBLUP parfile

`mixblup.datafile`
: Set to FALSE to manually write your MiXBLUP datafile

`mixblup.inputfile`
: Set to FALSE to manually write your MiXBLUP inputfile

`mixblup.genofile`
: Set to FALSE to manually write the MiXBLUP genotypefile

`mixblup.path`
: Provide path to MiXBLUP.exe (default is your working directory: Windows: MixBLUP; Linux ./MixBLUP.exe)

`mixblup.path.pedfile`
: Path from where to import the MiXBLUP pedfile

`mixblup.path.parfile`
: Path from where to import the MiXBLUP parfile

`mixblup.path.datafile`
: Path from where to import the MiXBLUP datafile

`mixblup.path.inputfile`
: Path from where to import the MiXBLUP inputfile

`mixblup.path.genofile`
: Path from where to import the MiXBLUP genofile

`mixblup.full.path.genofile`
: Path from where to import the MiXBLUP genofile

`mixblup.files`
: Directory to generate all files generated when using MiXBLUP (default: MiXBLUP_files/)

`mixblup.verbose`
: Set to TRUE to display MiXBLUP prints

`blupf90.verbose`
: Set to TRUE to display blupf90 prints

`mixblup.genetic.cov`
: Provide genetic covariance matrix to be used in MiXBLUP (lower-triangle is sufficent) (default: underlying true values)

`mixblup.residual.cov`
: Provide residual covariance matrix to be used in MiXBLUP (lower-triangle is sufficent) (default: underlying true values)

`mixblup.lambda`
: Lambda parameter in MiXBLUP (default: 1)

`mixblup.alpha`
: Alpha parameter in MiXBLUP (default: 0.95, with alpha + beta = 1 , warning: MiXBLUP software this is 1)

`mixblup.beta`
: Beta parameter in MiXBLUP (default: 0.05, with alpha + beta = 1 , warning: MiXBLUP software this is 0)

`mixblup.omega`
: Omega parameter in MiXBLUP (default: mixblup.lambda)

`mixblup.maxit`
: !Maxit qualifier in MiXBLUP (default: 5.000)

`mixblup.stopcrit`
: !STOPCRIT qualifier in MiXBLUP (default: not used, suggested value 1.E-4 for ssGBLUP) // will overwrite maxit

`mixblup.maf` !MAF qualifier in MiXBLUP (default: 0.005)

`mixblup.numproc`
: Numproc parameter in MiXBLUP (default: not set // 1)

`mixblup.apy` Set to TRUE to use APY inverse in MiXBLUP (default: FALSE)

`mixblup.apy.core`
: Number of core individuals in the APY algorithm (default: 5000)

`mixblup.ta` Set to TRUE to use the !Ta flag in MixBLUP

`mixblup.tac` Set to TRUE to use the !TAC flag in MixBLUP

`mixblup.skip` Set to TRUE to skip the actually system call to MiXBLUP and only write the MiXBLUP files

`blupf90.skip` Set to TRUE to skip the actually system calls of blupf90 and only write the blupf90 input files

`mixblup.restart`
: Set to TRUE to set the !RESTART flag in MiXBLUP (requires a "Solunf" file in the working directory)

`mixblup.nopeek` Set to TRUE to set the !NOPEEK flag in MiXBLUP

`mixblup.calcinbr.s`
: Set to TRUE to set the !CalcInbr flag to S

`mixblup.multiple.records`
: Set to TRUE to write multiple phenotypic records for an individual

`mixblup.attach` Set TRUE to just extent the existing genotype file instead of writting it completely new

`mixblup.debug` Set TRUE to set debugging flags for mixblup call (-Dmst > mixblup_debug.log) (default: FALSE)

`mixblup.plink` Set TRUE to write genotype files in PLINK format (requires R-package genio, default: FALSE)

`mixblup.cleanup`
: Delete all mixblup output files above the indicated size after MiXBLUP run completes (default: Inf)

`blupf90.pedfile`
: Set to FALSE to manually generate your MiXBLUP pedfile

`blupf90.parfile`
: Set to FALSE to manually generate your MiXBLUP parfile

`blupf90.datafile`
: Set to FALSE to manually generate your blupf90 datafile

`blupf90.inputfile`
: Set to FALSE to manually write your MiXBLUP inputfile

`blupf90.genofile`
: Set to FALSE to manually write the blupf90 genotypefile

`mixblup.dgv` Set TRUE to use DGV-PBLUP (Only applicable with TAC-BLUP)

`mixblup.dgv.freq`
: Path of allele frequency file for DGV-PBLUP

`mixblup.dgv.effect`
: Path of SNP effect file for DGV-PBLUP

`blupf90.path`
: Provide path to blupf90 (default is your working directory: Windows: ./blupf90+.exe ; Linux ./blupf90+.exe)

`renumf90.path`
: Provide path to blupf90 (default is your working directory: Windows: ./renumf90.exe ; Linux ./renumf90.exe)

`blupf90.path.pedfile`
: Path from where to import the blupf90 pedfile

`blupf90.path.parfile`
: Path from where to import the blupf90 parfile

`blupf90.path.datafile`
: Path from where to import the blupf90 data file

`blupf90.path.inputfile`
: Path from where to import the blupf90 inputfile

`blupf90.path.genofile`
: Path from where to import the blupf90 genotype file

`blupf90.files`
: Directory to generate all files generated when using blupf90 (default: blupf90_files/)

`blupf90.blksize`
: blupf90 parameter blksize (Default: number of traits)

`blupf90.no.quality`
: blupf90 setting OPTION no_quality_control (Default: FALSE)

`blupf90.conv_crit`
: blupf90 parameter conv_crit (Default: blupf90 default)

`BGLR.model`
: Select which BGLR model to use (default: "RKHS", alt: "BRR", "BL", "BayesA", "BayesB", "BayesC")

`BGLR.burnin`
: Number of burn-in steps in BGLR (default: 1000)

`BGLR.iteration`
: Number of iterations in BGLR (default: 5000)

`BGLR.print`
: If TRUE set verbose to TRUE in BGLR

`BGLR.save`
: Method to use in BGLR (default: "RKHS" - alt: NON currently)

`BGLR.save.random`
: Add random number to store location of internal BGLR computations (only needed when simulating a lot in parallel!)

`miraculix`
: If TRUE use miraculix to perform computations (ideally already generate population in creating.diploid with this; default: automatic detection from population list)

`miraculix.cores`
: Number of cores used in miraculix applications (default: 1)

`miraculix.mult`
: If TRUE use miraculix for matrix multiplications even if miraculix is not used for storage

`miraculix.chol`
: Set to FALSE to deactive miraculix based Cholesky-decomposition (default: TRUE)

`miraculix.destroyA`
: If FALSE A will not be destroyed in the process of inversion (less computing / more memory)

`estimate.u`
: If TRUE estimate u in breeding value estimation (Y = Xb + Zu + e)

`fast.uhat`
: Set to FALSE to derive inverse of A in rrBLUP (only required when this becomes numerical unstable otherwise)

`gwas.u`
: If TRUE estimate u via GWAS (relevant for gene editing)

`approx.residuals`
: If FALSE calculate the variance for each marker separatly instead of using a set variance (does not change order - only p-values)

`gwas.gen`, `gwas.cohorts`, `gwas.database`
: Generations/cohorts/groups to consider in GWAS analysis

`gwas.group.standard`
: If TRUE standardize phenotypes by group mean

`y.gwas.used`
: What y value to use in GWAS study (Default: "pheno", alt: "bv", "bve")

`gene.editing.offspring`
: If TRUE perform gene editing on newly generated individuals

`gene.editing.best`
: If TRUE perform gene editing on selected individuals

`gene.editing.offspring.sex`
: Which sex to perform editing on (Default c(TRUE,TRUE), mw)

`gene.editing.best.sex`
: Which sex to perform editing on (Default c(TRUE,TRUE), mw)

`nr.edits`
: Number of edits to perform per individual

`culling.non.selected`
: Set TRUE to cull all non-selected individuals (default: FALSE)

`culling.gen`, `culling.cohorts`, `culling.database`
: Generations/cohorst/groups to consider to culling

`culling.type`
: Default: 0, can be set to code different type of culling reasons (e.g. 0 - aging, 1 - selection, 2 - health)

`culling.time`
: Age of the individuals at culling // use time.point if the age of individuals is variable and culling is executed on individuals of different ages culled at the same time

`culling.name`
: Name of the culling action (user-interface stuff)

`culling.bv1`
: Reference Breeding value

`culling.share1`
: Probability of death for individuals with bv1

`culling.bv2`
: Alternative breeding value (linear extended for other bvs)

`culling.share2`
: Probability of death for individuals with bv2

`culling.index`
: Genomic index (default:0 - no genomic impact, use: "lastindex" to use the last selection index applied in selection)

`culling.single`
: Set to FALSE to not apply the culling module on all individuals of the cohort

`culling.all.copy`
: Set to FALSE to not kill copies of the same individual in the culling module

`mutation.rate` Mutation rate in each marker (default: 10^-8)

`remutation.rate`
: Remutation rate in each marker (default: 10^-8)

`recombination.rate`
: Average number of recombination per 1 length unit (default: 1M)

`recombination.rate.trait`
: Select a trait which BV will be used as a scalar for the expected number of recombination (default: 0)

`recombination.function`
: Function used to calculate position of recombination events (default: MoBPS::recombination.function.hal

`recombination.minimum.distance`
: Minimum distance between two points of recombination (default: 0)

`recombination.distance.penalty`
: Reduced probability for recombination events closer than this value - linear penalty (default: 0)

`recombination.distance.penalty.2`
: Reduced probability for recombination events closer than this value - quadratic penalty (default: 0)

`recom.f.indicator`
: Use step function for recombination map (transform snp.positions if possible instead)

`import.position.calculation`
: Function to calculate recombination point into adjacent/following SNP

`duplication.rate`
: Share of recombination points with a duplication (default: 0 - DEACTIVATED)

`duplication.length`
: Average length of a duplication (Exponentially distributed)

`duplication.recombination`
: Average number of recombinations per 1 length uit of duplication (default: 1)

`gen.architecture.m`, `gen.architecture.f`
: Genetic architecture for male/female individuals (default: 0 - no transformation)

`add.architecture`
: List with two vectors containing (A: length of chromosomes, B: position in cM of SNPs)

`intern.func` Chose which function will be used for simulation of meiosis (default: 0, alt: 1,2) - can be faster for specific cases

`delete.haplotypes`
: Generations for with haplotypes of founders can be deleted from population list for memory reduction (default: NULL)

`delete.recombi` Generations for which recombination points can be deleted from the population list for memory reduction (default: NULL)

`delete.recombi.only.non.genotyped`
: Set TRUE to only remove points of recombination for non-genotyped individuals

`delete.recombi.class`
: Set TRUE to only remove points of recombination for individuals from a specific class

`delete.individuals`
: Generations for with individuals are completely deleted from population list for memory reduction (default: NULL)

`delete.gen`
: Generations to entirely deleted fro population list for memory reduction (default: NULL)

`delete.sex`
: Remove all individuals from these sex from generation delete.individuals (default: 1:2 ; note:delete individuals=NULL)

`delete.same.origin`
: If TRUE delete recombination points when genetic origin of adjacent segments is the same

`save.recombination.history`
: If TRUE store the time point of each recombination event

`store.sparse`
: Set to TRUE to store the pedigree relationship matrix as a sparse matrix

`storage.save`
: Lower numbers will lead to less memory but slightly higher computing time for calculation of the pedigree relationship matrix (default: 1.5, min: 1)

`verbose`
: Set to FALSE to not display any prints

`report.accuracy`
: Report the accuracy of the breeding value estimation

`store.breeding.totals`
: If TRUE store information on selected individuals in $info$breeding.totals (default: FALSE)

`store.bve.data`
: If TRUE store information of bve in $info$bve.data

`store.comp.times`
: If TRUE store computation times in $info$comp.times.general (default: TRUE)

`store.comp.times.bve`
: If TRUE store computation times of breeding value estimation in $info$comp.times.bve (default: TRUE)

`store.comp.times.generation`
: If TRUE store computation times of mating simulations in $info$comp.times.generation (default: TRUE)

`store.effect.freq`
: If TRUE store the allele frequency of effect markers per generation

`Rprof`
: Store computation times of each function

`randomSeed`
: Set random seed of the process

`display.progress`
: Set FALSE to not display progress bars. Setting verbose to FALSE will automatically deactive progress bars

`time.point`
: Time point at which the new individuals are generated

`age.point`
: Time point at which the new individuals are born (default: time.point - mostly useful in the founder generation)

`creating.type` Technique to generate new individuals (use mostly intended for web-based application)

`import.relationship.matrix`
Input the wanted relationship matrix with this parameter (default: NULL - relationship matrix will be calculated from other sources)

`export.selected`
Set to TRUE to export the list of selected individuals

`export.selected.database`
Set to TRUE to export a database of the selected individuals

`export.relationship.matrix`
Export the relationship matrix used in the breeding value estimation

`pen.assignments`
This is a placeholder to deactivate this module for now

`pen.size` Pen size. When different types of pen are used: use a matrix with two columns coding Number of individuals per pen, Probability for each pen size

`pen.by.sex` Only individuals of the same sex are put in the same pen (default: TRUE)

`pen.by.litter` Only individuals of the same litter are put in the same pen (default: FALSE)

`pen.size.overwrite`
Set to FALSE to not use the input for pen.size for down-stream use of breeding.diploid (default: TRUE)

`selection.m`, `selection.f`
(OLD! use selection criteria) Selection criteria for male/female individuals (Set to "random" to randomly select individuals - default: "function" based on selection.criteria ((usually breeding values)))

`new.bv.observation.gen`, `new.bv.observation.cohorts`, `new.bv.observation.database`
(OLD! use phenotyping.gen/cohorts/database) Vector of generation from which to generate additional phenotypes

`best1.from.group`, `best1.from.cohort`
(OLD!- use selection.m.database/cohorts) Groups of individuals to consider as First Parent / Father (also female individuals are possible)

`best2.from.group`, `best2.from.cohort`
(OLD!- use selection.f.database/cohorts) Groups of individuals to consider as Second Parent / Mother (also male individuals are possible)

`new.bv.observation`
(OLD! - use phenotyping) Quick access to phenotyping for (all: "all", non-phenotyped: "non_obs", non-phenotyped male: "non_obs_m", non-phenotyped female: "non_obs_f")

`reduce.group` (OLD! - use culling modules) Groups of individuals for reduce to a new size (by changing class to -1)

`reduce.group.selection`
(OLD! - use culling modules) Selection criteria for reduction of groups (cf. selection.m / selection.f - default: "random")

`new.bv.child` (OLD! - use phenotyping.child) Starting phenotypes of newly generated individuals (default: "zero", alt: "mean" of both parents, "obs" - regular observation)

computation.A (OLD! - use relationship.matrix) Method to calculate relationship matrix for the breeding value estimation (Default: "vanRaden", alt: "pedigree", "CE", "non_stand", "CE2", "CM")

computation.A.ogc
(OLD! use relationship.matrix.ogc) Method to calculate pedigree matrix in OGC (Default: "pedigree", alt: "vanRaden", "CE", "non_stand", "CE2", "CM")

new.phenotype.correlation
(OLD! - use new.residual.correlation!) Correlation of the simulated enviromental variance

offspring.bve.parents.gen, offspring.bve.parents.cohorts, offspring.bve.parents.database
(OLD! use offpheno.parents.gen/database/cohorts) Generations/cohorts/groups to consider to derive phenotype from offspring phenotypes

offspring.bve.offspring.gen, offspring.bve.offspring.cohorts, offspring.bve.offspring.database
(OLD! use offpheno.offspring.gen/database/cohorts) Active generations/cohorts/groups for import of offspring phenotypes

input.phenotype
(OLD! use bve.input.phenotype) Select what to use in BVE (default: own phenotype ("own"), offspring phenotype ("off"), their average ("mean") or a weighted average ("weighted"))

multiple.bve.weights.m, multiple.bve.weights.f
(OLD! use selection.index.weights.m/f) Weighting between traits (default: 1)

multiple.bve.scale.m, multiple.bve.scale.f
(OLD! use selection.index.scale.m/f) Default: "bv_sd"; Set to "pheno_sd" when using gains per phenotypic SD, "unit" when using gains per unit, "bve" when using estimated breeding values

use.recalculate.manual
Set to TRUE to use recalculate.manual to calculate genomic values (all individuals and traits jointly, default: FALSE)

recalculate.manual.subset
Maximum number of individuals to process at the same time ((genotypes are in memory))

compute.grandparent.contribution
compute share of genome inherited from each grandparent based on recombination points (default: FALSE)

size.scaling Set to value to scale all input for breeding.size / selection.size (This will not work for all breeding programs / less general than json.simulation)

parallel.internal
Internal parameter for the parallelization

varg Experimental parameter for Tobias Niehoff (do not touch!)

gain.stats Set to FALSE to not compute genetic gains compared to previous generation (selection)

next.id Id to assign to first next individual generated

copy.individual.use, copy.individual.use2
Use this to skip copying some entries from the internal storage ((minor speed up))

Value

Population-list

Examples

```
population <- creating.diploid(nsnp=1000, nindi=100)
population <- breeding.diploid(population, breeding.size=100, selection.size=c(25,25))
```

---

breeding.intern *Internal function to simulate one meiosis*

Description

Internal function to simulate one meiosis

Usage

```
breeding.intern(
  info.parent,
  parent,
  population,
  mutation.rate = 10^-5,
  remutation.rate = 10^-5,
  recombination.rate = 1,
  recom.f.indicator = NULL,
  duplication.rate = 0,
  duplication.length = 0.01,
  duplication.recombination = 1,
  delete.same.origin = FALSE,
  gene.editing = FALSE,
  nr.edits = 0,
  gen.architecture = 0,
  decodeOriginsU = MoBPS::decodeOriginsR,
  recombination.function = MoBPS::recombination.function.haldane,
  dup_activ = FALSE,
  rt_activ = FALSE,
  grandsib_activ = FALSE
)
```

Arguments

info.parent	position of the parent in the dataset
parent	list of information regarding the parent
population	Population list
mutation.rate	Mutation rate in each marker (default: 10^-5)

`remutation.rate`
: Remutation rate in each marker (default: 10^-5)

`recombination.rate`
: Average number of recombination per 1 length unit (default: 1M)

`recom.f.indicator`
: Use step function for recombination map (transform snp.positions if possible instead)

`duplication.rate`
: Share of recombination points with a duplication (default: 0 - DEACTIVATED)

`duplication.length`
: Average length of a duplication (Exponentially distributed)

`duplication.recombination`
: Average number of recombinations per 1 length uit of duplication (default: 1)

`delete.same.origin`
: If TRUE delete recombination points when genetic origin of adjacent segments is the same

`gene.editing`
: If TRUE perform gene editing on newly generated individual

`nr.edits`
: Number of edits to perform per individual

`gen.architecture`
: Used underlying genetic architecture (genome length in M)

`decodeOriginsU`
: Used function for the decoding of genetic origins [[5]]/[[6]]

`recombination.function`
: Function used to calculate position of recombination events (default: MoBPS::recombination.function.hal

`dup_activ`
: Internal parameter to check if duplications have to be simulated

`rt_activ`
: Internal parameter to check if RTs have to be simulated

`grandsib_activ`
: Internal parameter to check if grandsibling contributions have to be calculated

Value

Inherited parent gamete

Examples

```
data(ex_pop)
child_gamete <- breeding.intern(info.parent = c(1,1,1), parent = ex_pop$breeding[[1]][[1]][[1]],
                                population = ex_pop)
```

`breeding.intern1` *Internal function to simulate one meiosis*

Description

Internal function to simulate one meiosis

Usage

```
breeding.intern1(
  info.parent,
  parent,
  population,
  mutation.rate = 10^-5,
  remutation.rate = 10^-5,
  recombination.rate = 1,
  recom.f.indicator = NULL,
  duplication.rate = 0,
  duplication.length = 0.01,
  duplication.recombination = 1,
  delete.same.origin = FALSE,
  gene.editing = FALSE,
  nr.edits = 0,
  gen.architecture = 0,
  decodeOriginsU = MoBPS::decodeOriginsR,
  recombination.function = MoBPS::recombination.function.haldane,
  dup_activ = TRUE,
  rt_activ = TRUE,
  grandsib_activ = TRUE
)
```

Arguments

`info.parent`	position of the parent in the dataset
`parent`	list of information regarding the parent
`population`	Population list
`mutation.rate`	Mutation rate in each marker (default: 10^-5)
`remutation.rate`	Remutation rate in each marker (default: 10^-5)
`recombination.rate`	Average number of recombination per 1 length unit (default: 1M)
`recom.f.indicator`	Use step function for recombination map (transform snp.positions if possible instead)
`duplication.rate`	Share of recombination points with a duplication (default: 0 - DEACTIVATED)

duplication.length
: Average length of a duplication (Exponentially distributed)

duplication.recombination
: Average number of recombinations per 1 length uit of duplication (default: 1)

delete.same.origin
: If TRUE delete recombination points when genetic origin of adjacent segments is the same

gene.editing
: If TRUE perform gene editing on newly generated individual

nr.edits
: Number of edits to perform per individual

gen.architecture
: Used underlying genetic architecture (genome length in M)

decodeOriginsU
: Used function for the decoding of genetic origins [[5]]/[[6]]

recombination.function
: Function used to calculate position of recombination events (default: MoBPS::recombination.function.hal

dup_activ
: Internal parameter to check if duplications have to be simulated

rt_activ
: Internal parameter to check if RTs have to be simulated

grandsib_activ
: Internal parameter to check if grandsibling contributions have to be calculated

Value

Inherited parent gamete

Examples

```
data(ex_pop)
child_gamete <- breeding.intern(info.parent = c(1,1,1), parent = ex_pop$breeding[[1]][[1]][[1]],
                                population = ex_pop)
```

---

breeding.intern2 *Internal function to simulate one meiosis*

Description

Internal function to simulate one meiosis

Usage

```
breeding.intern2(
  info.parent,
  parent,
  population,
  mutation.rate = 10^-5,
  remutation.rate = 10^-5,
  recombination.rate = 1,
  recom.f.indicator = NULL,
```

```
  duplication.rate = 0,
  duplication.length = 0.01,
  duplication.recombination = 1,
  delete.same.origin = FALSE,
  gene.editing = FALSE,
  nr.edits = 0,
  gen.architecture = 0,
  decodeOriginsU = MoBPS::decodeOriginsR,
  recombination.function = MoBPS::recombination.function.haldane,
  dup_activ = TRUE,
  rt_activ = TRUE,
  grandsib_activ = TRUE
)
```

Arguments

info.parent	position of the parent in the dataset
parent	list of information regarding the parent
population	Population list
mutation.rate	Mutation rate in each marker (default: 10^-5)
remutation.rate	Remutation rate in each marker (default: 10^-5)
recombination.rate	Average number of recombination per 1 length unit (default: 1M)
recom.f.indicator	Use step function for recombination map (transform snp.positions if possible instead)
duplication.rate	Share of recombination points with a duplication (default: 0 - DEACTIVATED)
duplication.length	Average length of a duplication (Exponentially distributed)
duplication.recombination	Average number of recombinations per 1 length uit of duplication (default: 1)
delete.same.origin	If TRUE delete recombination points when genetic origin of adjacent segments is the same
gene.editing	If TRUE perform gene editing on newly generated individual
nr.edits	Number of edits to perform per individual
gen.architecture	Used underlying genetic architecture (genome length in M)
decodeOriginsU	Used function for the decoding of genetic origins [[5]]/[[6]]
recombination.function	Function used to calculate position of recombination events (default: MoBPS::recombination.function.hal
dup_activ	Internal parameter to check if duplications have to be simulated
rt_activ	Internal parameter to check if RTs have to be simulated
grandsib_activ	Internal parameter to check if grandsibling contributions have to be calculated

Value

Inherited parent gamete

Examples

```
data(ex_pop)
child_gamete <- breeding.intern2(info.parent = c(1,1,1), parent = ex_pop$breeding[[1]][[1]][[1]],
                                 population = ex_pop)
```

`breeding.intern3` *Internal function to simulate one meiosis*

Description

Internal function to simulate one meiosis

Usage

```
breeding.intern3(
  info.parent,
  parent,
  population,
  mutation.rate = 10^-5,
  remutation.rate = 10^-5,
  recombination.rate = 1,
  recom.f.indicator = NULL,
  duplication.rate = 0,
  duplication.length = 0.01,
  duplication.recombination = 1,
  delete.same.origin = FALSE,
  gene.editing = FALSE,
  nr.edits = 0,
  gen.architecture = 0,
  decodeOriginsU = MoBPS::decodeOriginsR,
  recombination.function = MoBPS::recombination.function.haldane,
  dup_activ = TRUE,
  rt_activ = TRUE,
  grandsib_activ = TRUE
)
```

Arguments

`info.parent`	position of the parent in the dataset
`parent`	list of information regarding the parent
`population`	Population list
`mutation.rate`	Mutation rate in each marker (default: 10^-5)

remutation.rate
: Remutation rate in each marker (default: 10^-5)

recombination.rate
: Average number of recombination per 1 length unit (default: 1M)

recom.f.indicator
: Use step function for recombination map (transform snp.positions if possible instead)

duplication.rate
: Share of recombination points with a duplication (default: 0 - DEACTIVATED)

duplication.length
: Average length of a duplication (Exponentially distributed)

duplication.recombination
: Average number of recombinations per 1 length uit of duplication (default: 1)

delete.same.origin
: If TRUE delete recombination points when genetic origin of adjacent segments is the same

gene.editing
: If TRUE perform gene editing on newly generated individual

nr.edits
: Number of edits to perform per individual

gen.architecture
: Used underlying genetic architecture (genome length in M)

decodeOriginsU
: Used function for the decoding of genetic origins [[5]]/[[6]]

recombination.function
: Function used to calculate position of recombination events (default: MoBPS::recombination.function.hal

dup_activ
: Internal parameter to check if duplications have to be simulated

rt_activ
: Internal parameter to check if RTs have to be simulated

grandsib_activ
: Internal parameter to check if grandsibling contributions have to be calculated

Value

Inherited parent gamete

Examples

```
data(ex_pop)
child_gamete <- breeding.intern3(info.parent = c(1,1,1), parent = ex_pop$breeding[[1]][[1]][[1]],
                                 population = ex_pop)
```

`breeding.intern4` *Internal function to simulate one meiosis*

Description

Internal function to simulate one meiosis

Usage

```
breeding.intern4(
  info.parent,
  parent,
  population,
  mutation.rate = 10^-5,
  remutation.rate = 10^-5,
  recombination.rate = 1,
  recom.f.indicator = NULL,
  duplication.rate = 0,
  duplication.length = 0.01,
  duplication.recombination = 1,
  delete.same.origin = FALSE,
  gene.editing = FALSE,
  nr.edits = 0,
  gen.architecture = 0,
  decodeOriginsU = MoBPS::decodeOriginsR,
  recombination.function = MoBPS::recombination.function.haldane,
  dup_activ = TRUE,
  rt_activ = TRUE,
  grandsib_activ = TRUE
)
```

Arguments

`info.parent`	position of the parent in the dataset
`parent`	list of information regarding the parent
`population`	Population list
`mutation.rate`	Mutation rate in each marker (default: 10^-5)
`remutation.rate`	Remutation rate in each marker (default: 10^-5)
`recombination.rate`	Average number of recombination per 1 length unit (default: 1M)
`recom.f.indicator`	Use step function for recombination map (transform snp.positions if possible instead)
`duplication.rate`	Share of recombination points with a duplication (default: 0 - DEACTIVATED)

duplication.length
: Average length of a duplication (Exponentially distributed)

duplication.recombination
: Average number of recombinations per 1 length uit of duplication (default: 1)

delete.same.origin
: If TRUE delete recombination points when genetic origin of adjacent segments is the same

gene.editing
: If TRUE perform gene editing on newly generated individual

nr.edits
: Number of edits to perform per individual

gen.architecture
: Used underlying genetic architecture (genome length in M)

decodeOriginsU
: Used function for the decoding of genetic origins [[5]]/[[6]]

recombination.function
: Function used to calculate position of recombination events (default: MoBPS::recombination.function.hal

dup_activ
: Internal parameter to check if duplications have to be simulated

rt_activ
: Internal parameter to check if RTs have to be simulated

grandsib_activ
: Internal parameter to check if grandsibling contributions have to be calculated

Value

Inherited parent gamete

Examples

```
data(ex_pop)
child_gamete <- breeding.intern4(info.parent = c(1,1,1), parent = ex_pop$breeding[[1]][[1]][[1]],
                                 population = ex_pop)
```

---

breeding.intern5 *Internal function to simulate one meiosis*

Description

Internal function to simulate one meiosis

Usage

```
breeding.intern5(
  info.parent,
  parent,
  population,
  mutation.rate = 10^-5,
  remutation.rate = 10^-5,
  recombination.rate = 1,
  recom.f.indicator = NULL,
```

```
  duplication.rate = 0,
  duplication.length = 0.01,
  duplication.recombination = 1,
  delete.same.origin = FALSE,
  gene.editing = FALSE,
  nr.edits = 0,
  gen.architecture = 0,
  decodeOriginsU = MoBPS::decodeOriginsR,
  recombination.function = MoBPS::recombination.function.haldane,
  dup_activ = TRUE,
  rt_activ = TRUE,
  grandsib_activ = TRUE
)
```

Arguments

info.parent	position of the parent in the dataset
parent	list of information regarding the parent
population	Population list
mutation.rate	Mutation rate in each marker (default: 10^-5)
remutation.rate	Remutation rate in each marker (default: 10^-5)
recombination.rate	Average number of recombination per 1 length unit (default: 1M)
recom.f.indicator	Use step function for recombination map (transform snp.positions if possible instead)
duplication.rate	Share of recombination points with a duplication (default: 0 - DEACTIVATED)
duplication.length	Average length of a duplication (Exponentially distributed)
duplication.recombination	Average number of recombinations per 1 length uit of duplication (default: 1)
delete.same.origin	If TRUE delete recombination points when genetic origin of adjacent segments is the same
gene.editing	If TRUE perform gene editing on newly generated individual
nr.edits	Number of edits to perform per individual
gen.architecture	Used underlying genetic architecture (genome length in M)
decodeOriginsU	Used function for the decoding of genetic origins [[5]]/[[6]]
recombination.function	Function used to calculate position of recombination events (default: MoBPS::recombination.function.hal
dup_activ	Internal parameter to check if duplications have to be simulated
rt_activ	Internal parameter to check if RTs have to be simulated
grandsib_activ	Internal parameter to check if grandsibling contributions have to be calculated

Value

Inherited parent gamete

Examples

```
data(ex_pop)
child_gamete <- breeding.intern5(info.parent = c(1,1,1), parent = ex_pop$breeding[[1]][[1]][[1]],
                                 population = ex_pop)
```

`breeding.intern6` *Internal function to simulate one meiosis*

Description

Internal function to simulate one meiosis

Usage

```
breeding.intern6(
  info.parent,
  parent,
  population,
  mutation.rate = 10^-5,
  remutation.rate = 10^-5,
  recombination.rate = 1,
  recom.f.indicator = NULL,
  duplication.rate = 0,
  duplication.length = 0.01,
  duplication.recombination = 1,
  delete.same.origin = FALSE,
  gene.editing = FALSE,
  nr.edits = 0,
  gen.architecture = 0,
  decodeOriginsU = MoBPS::decodeOriginsR,
  recombination.function = MoBPS::recombination.function.haldane,
  dup_activ = TRUE,
  rt_activ = TRUE,
  grandsib_activ = TRUE
)
```

Arguments

`info.parent`	position of the parent in the dataset
`parent`	list of information regarding the parent
`population`	Population list
`mutation.rate`	Mutation rate in each marker (default: 10^-5)

`remutation.rate`
: Remutation rate in each marker (default: 10^-5)

`recombination.rate`
: Average number of recombination per 1 length unit (default: 1M)

`recom.f.indicator`
: Use step function for recombination map (transform snp.positions if possible instead)

`duplication.rate`
: Share of recombination points with a duplication (default: 0 - DEACTIVATED)

`duplication.length`
: Average length of a duplication (Exponentially distributed)

`duplication.recombination`
: Average number of recombinations per 1 length uit of duplication (default: 1)

`delete.same.origin`
: If TRUE delete recombination points when genetic origin of adjacent segments is the same

`gene.editing`
: If TRUE perform gene editing on newly generated individual

`nr.edits`
: Number of edits to perform per individual

`gen.architecture`
: Used underlying genetic architecture (genome length in M)

`decodeOriginsU`
: Used function for the decoding of genetic origins [[5]]/[[6]]

`recombination.function`
: Function used to calculate position of recombination events (default: MoBPS::recombination.function.hal

`dup_activ`
: Internal parameter to check if duplications have to be simulated

`rt_activ`
: Internal parameter to check if RTs have to be simulated

`grandsib_activ`
: Internal parameter to check if grandsibling contributions have to be calculated

Value

Inherited parent gamete

Examples

```
data(ex_pop)
child_gamete <- breeding.intern6(info.parent = c(1,1,1), parent = ex_pop$breeding[[1]][[1]][[1]],
                                 population = ex_pop)
```

`breeding.intern7` *Internal function to simulate one meiosis*

Description

Internal function to simulate one meiosis

Usage

```
breeding.intern7(
  info.parent,
  parent,
  population,
  mutation.rate = 10^-5,
  remutation.rate = 10^-5,
  recombination.rate = 1,
  recom.f.indicator = NULL,
  duplication.rate = 0,
  duplication.length = 0.01,
  duplication.recombination = 1,
  delete.same.origin = FALSE,
  gene.editing = FALSE,
  nr.edits = 0,
  gen.architecture = 0,
  decodeOriginsU = MoBPS::decodeOriginsR,
  recombination.function = MoBPS::recombination.function.haldane,
  dup_activ = TRUE,
  rt_activ = TRUE,
  grandsib_activ = TRUE
)
```

Arguments

`info.parent`	position of the parent in the dataset
`parent`	list of information regarding the parent
`population`	Population list
`mutation.rate`	Mutation rate in each marker (default: 10^-5)
`remutation.rate`	Remutation rate in each marker (default: 10^-5)
`recombination.rate`	Average number of recombination per 1 length unit (default: 1M)
`recom.f.indicator`	Use step function for recombination map (transform snp.positions if possible instead)
`duplication.rate`	Share of recombination points with a duplication (default: 0 - DEACTIVATED)

duplication.length
: Average length of a duplication (Exponentially distributed)

duplication.recombination
: Average number of recombinations per 1 length uit of duplication (default: 1)

delete.same.origin
: If TRUE delete recombination points when genetic origin of adjacent segments is the same

gene.editing
: If TRUE perform gene editing on newly generated individual

nr.edits
: Number of edits to perform per individual

gen.architecture
: Used underlying genetic architecture (genome length in M)

decodeOriginsU
: Used function for the decoding of genetic origins [[5]]/[[6]]

recombination.function
: Function used to calculate position of recombination events (default: MoBPS::recombination.function.hal

dup_activ
: Internal parameter to check if duplications have to be simulated

rt_activ
: Internal parameter to check if RTs have to be simulated

grandsib_activ
: Internal parameter to check if grandsibling contributions have to be calculated

Value

Inherited parent gamete

Examples

```
data(ex_pop)
child_gamete <- breeding.intern7(info.parent = c(1,1,1), parent = ex_pop$breeding[[1]][[1]][[1]],
                                 population = ex_pop)
```

---

breeding.intern8 *Internal function to simulate one meiosis*

Description

Internal function to simulate one meiosis

Usage

```
breeding.intern8(
  info.parent,
  parent,
  population,
  mutation.rate = 10^-5,
  remutation.rate = 10^-5,
  recombination.rate = 1,
  recom.f.indicator = NULL,
```

```
  duplication.rate = 0,
  duplication.length = 0.01,
  duplication.recombination = 1,
  delete.same.origin = FALSE,
  gene.editing = FALSE,
  nr.edits = 0,
  gen.architecture = 0,
  decodeOriginsU = MoBPS::decodeOriginsR,
  recombination.function = MoBPS::recombination.function.haldane,
  dup_activ = TRUE,
  rt_activ = TRUE,
  grandsib_activ = TRUE
)
```

Arguments

info.parent	position of the parent in the dataset
parent	list of information regarding the parent
population	Population list
mutation.rate	Mutation rate in each marker (default: 10^-5)
remutation.rate	Remutation rate in each marker (default: 10^-5)
recombination.rate	Average number of recombination per 1 length unit (default: 1M)
recom.f.indicator	Use step function for recombination map (transform snp.positions if possible instead)
duplication.rate	Share of recombination points with a duplication (default: 0 - DEACTIVATED)
duplication.length	Average length of a duplication (Exponentially distributed)
duplication.recombination	Average number of recombinations per 1 length uit of duplication (default: 1)
delete.same.origin	If TRUE delete recombination points when genetic origin of adjacent segments is the same
gene.editing	If TRUE perform gene editing on newly generated individual
nr.edits	Number of edits to perform per individual
gen.architecture	Used underlying genetic architecture (genome length in M)
decodeOriginsU	Used function for the decoding of genetic origins [[5]]/[[6]]
recombination.function	Function used to calculate position of recombination events (default: MoBPS::recombination.function.hal
dup_activ	Internal parameter to check if duplications have to be simulated
rt_activ	Internal parameter to check if RTs have to be simulated
grandsib_activ	Internal parameter to check if grandsibling contributions have to be calculated

Value

Inherited parent gamete

Examples

```
data(ex_pop)
child_gamete <- breeding.intern8(info.parent = c(1,1,1), parent = ex_pop$breeding[[1]][[1]][[1]],
                                 population = ex_pop)
```

`bv.development` *Development of genetic/breeding value*

Description

Function to plot genetic/breeding values for multiple generation/cohorts

Usage

```
bv.development(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  confidence = c(1, 2, 3),
  development = c(1, 2, 3),
  quantile = 0.95,
  bvrow = "all",
  ignore.zero = TRUE,
  json = FALSE,
  display.time.point = FALSE,
  display.creating.type = FALSE,
  display.cohort.name = FALSE,
  display.sex = FALSE,
  equal.spacing = FALSE,
  time_reorder = FALSE,
  display.line = TRUE,
  ylim = NULL,
  fix_mfrow = FALSE
)
```

Arguments

`population`	population list
`database`	Groups of individuals to consider for the export
`gen`	Quick-insert for database (vector of all generations to export)
`cohorts`	Quick-insert for database (vector of names of cohorts to export)

confidence	Draw confidence intervals for (1- bv, 2- bve, 3- pheno; default: c(1,2,3))
development	Include development of (1- bv, 2- bve, 3- pheno; default: c(1,2,3))
quantile	Quantile of the confidence interval to draw (default: 0.05)
bvrow	Which traits to display (for multiple traits separate plots (par(mfrow)))
ignore.zero	Cohorts with only 0 individuals are not displayed (default: TRUE)
json	If TRUE extract which cohorts to plot according to the json-file used in json.simulation
display.time.point	Set TRUE to use time point of generated to sort groups
display.creating.type	Set TRUE to show Breedingtype used in generation (web-interface)
display.cohort.name	Set TRUE to display the name of the cohort in the x-axis
display.sex	Set TRUE to display the creating.type (Shape of Points - web-based-application)
equal.spacing	Equal distance between groups (independent of time.point)
time_reorder	Set TRUE to order cohorts according to the time point of generation
display.line	Set FALSE to not display the line connecting cohorts
ylim	Set this to fix the y-axis of the plot
fix_mfrow	Set TRUE to not use mfrow - use for custom plots

Value

Genomic values of selected gen/database/cohort

Examples

```
data(ex_pop)
bv.development(ex_pop, gen=1:5)
```

---

bv.development.box *Development of genetic/breeding value using a boxplot*

Description

Function to plot genetic/breeding values for multiple generation/cohorts using box plots

Usage

```
bv.development.box(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  bvrow = "all",
  json = FALSE,
```

```
  display = "bv",
  display.selection = FALSE,
  display.reproduction = FALSE,
  ylim = NULL,
  fix_mfrow = FALSE
)
```

Arguments

population	population list
database	Groups of individuals to consider for the export
gen	Quick-insert for database (vector of all generations to export)
cohorts	Quick-insert for database (vector of names of cohorts to export)
bvrow	Which traits to display (for multiple traits separate plots (par(mfrow)))
json	If TRUE extract which cohorts to plot according to the json-file used in json.simulation
display	Choose between "bv", "pheno", "bve" (default: "bv")
display.selection	Display lines between generated cohorts via selection (webinterface)
display.reproduction	Display lines between generated cohorts via reproduction (webinterface)
ylim	Set this to fix the y-axis of the plot
fix_mfrow	Set TRUE to not use mfrow - use for custom plots

Value

Genomic values of selected gen/database/cohort

Examples

```
data(ex_pop)
bv.development.box(ex_pop, gen=1:5)
```

---

bv.standardization *BV standardization*

Description

Function to get mean and genetic variance of a trait to a fixed value

Usage

```
bv.standardization(
  population,
  mean.target = NA,
  var.target = NA,
  gen = NULL,
  database = NULL,
  cohorts = NULL,
  adapt.bve = TRUE,
  adapt.pheno = NULL,
  verbose = FALSE,
  set.zero = FALSE,
  adapt.sigma.e = FALSE,
  traits = NULL
)
```

Arguments

population	Population list
mean.target	Target mean
var.target	Target variance
gen	Quick-insert for database (vector of all generations to export)
database	Groups of individuals to consider for the export
cohorts	Quick-insert for database (vector of names of cohorts to export)
adapt.bve	Modify previous breeding value estimations by scaling (default: TRUE)
adapt.pheno	Modify previous phenotypes by scaling (default: TRUE)
verbose	Set to TRUE to display prints
set.zero	Set to TRUE to have no effect on the 0 genotype (or 00 for QTLs with 2 underlying SNPs)
adapt.sigma.e	Set to TRUE to scale sigma.e values used based on scaling
traits	Use this parameter to only perform scaling of these traits (alternatively set values in mean/var.target to NA, default: all traits)

Value

Population-list with scaled QTL-effects

Examples

```
population <- creating.diploid(nsnp=1000, nindi=100, n.additive=100)
population <- bv.standardization(population, mean.target=200, var.target=5)
```

`calculate.bv` *Calculate breeding values*

Description

Internal function to calculate the breeding value of a given individual

Usage

```
calculate.bv(
  population,
  gen,
  sex,
  nr,
  activ_bv,
  import.position.calculation = NULL,
  decodeOriginsU = decodeOriginsR,
  store.effect.freq = FALSE,
  bit.storing = FALSE,
  nbits = 30,
  output_compressed = FALSE,
  bv.ignore.traits = NULL
)
```

Arguments

`population`	Population list
`gen`	Generation of the individual of interest
`sex`	Sex of the individual of interest
`nr`	Number of the individual of interest
`activ_bv`	traits to consider
`import.position.calculation`	Function to calculate recombination point into adjacent/following SNP
`decodeOriginsU`	Used function for the decoding of genetic origins [[5]]/[[6]]
`store.effect.freq`	If TRUE store the allele frequency of effect markers per generation
`bit.storing`	Set to TRUE if the MoBPS (not-miraculix! bit-storing is used)
`nbits`	Bits available in MoBPS-bit-storing
`output_compressed`	Set to TRUE to get a miraculix-compressed genotype/haplotype
`bv.ignore.traits`	Vector of traits to ignore in the calculation of the genomic value (default: NULL; Only recommended for high number of traits and experienced users!)

Value

[[1]] true genomic value [[2]] allele frequency at QTL markers

Examples

```
data(ex_pop)
calculate.bv(ex_pop, gen=1, sex=1, nr=1, activ_bv = 1)
```

---

cattle_chip *Cattle chip*

Description

Genome for cattle according to Ma et al.

Usage

```
cattle_chip
```

Author(s)

Torsten Pook <torsten.pook@wur.nl>

Source

Ma et al 2015

---

check.parents *Relatedness check between two individuals*

Description

Internal function to check the relatedness between two individuals

Usage

```
check.parents(
  population,
  info.father,
  info.mother,
  max.rel = 2,
  avoid.mating.parent = FALSE,
  still.check = FALSE
)
```

Arguments

population	Population list
info.father	position of the first parent in the dataset
info.mother	position of the second parent in the dataset
max.rel	maximal allowed relationship (default: 2, alt: 1 no full-sibs, 0 no half-sibs)
avoid.mating.parent	Set to TRUE to avoid matings of an individual to its parents
still.check	Internal parameter (avoid.mating.parent check)

Value

logical with TRUE if relatedness does not exceed max.rel / FALSE otherwise.

Examples

```
data(ex_pop)
check.parents(ex_pop, info.father=c(4,1,1,1), info.mother=c(4,2,1,1))
```

---

chicken_chip *chicken chip*

Description

Genome for chicken according to Groenen et al.

Usage

```
chicken_chip
```

Author(s)

Torsten Pook <torsten.pook@wur.nl>

Source

Groenen et al 2009

clean.up *Clean-up recombination points*

Description

Function to remove recombination points + origins with no influence on markers

Usage

```
clean.up(population, gen = "all", database = NULL, cohorts = NULL)
```

Arguments

population	Population list
gen	Generations to clean up (default: "current")
database	Groups of individuals to consider
cohorts	Quick-insert for database (vector of names of cohorts to export)

Value

Population-list with deleted irrelevant recombination points

Examples

```
data(ex_pop)
ex_pop <- clean.up(ex_pop)
```

codeOriginsR *Origins-coding(R)*

Description

R-Version of the internal bitwise-coding of origins

Usage

```
codeOriginsR(M)
```

Arguments

M	Origins matrix

Value

Bit-wise coded origins

Examples

```
codeOriginsR(cbind(1,1,1,1))
```

`combine.traits` *Combine traits*

Description

Function to combine traits in the BVE

Usage

```
combine.traits(
  population,
  combine.traits = NULL,
  combine.name = NULL,
  remove.combine = NULL,
  remove.all = FALSE
)
```

Arguments

`population`	Population list
`combine.traits`	Vector containing the traits (numbers) to combine into a joined trait
`combine.name`	Name of the combined trait
`remove.combine`	Remove a selected previously generated combined trait
`remove.all`	Set TRUE to remove all previously generated combined traits

Value

Population-list

Examples

```
population <- creating.diploid(nsnp=100, nindi=100, n.additive = c(50,50))
population <- combine.traits(population, combine.traits=1:2)
population <- breeding.diploid(population, bve=TRUE, phenotyping.gen=1, heritability=0.3)
```

`computing.costs` *Compute costs of a breeding program*

Description

Function to derive the costs of a breeding program / population-list

Usage

```
computing.costs(
  population,
  phenotyping.costs = 10,
  genotyping.costs = 100,
  fix.costs = 0,
  fix.costs.annual = 0,
  profit.per.bv = 1,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  interest.rate = 1,
  base.gen = 1
)
```

Arguments

`population`	population-list
`phenotyping.costs`	Costs for the generation of a phenotype
`genotyping.costs`	Costs for the generation of a genotype
`fix.costs`	one time occurring fixed costs
`fix.costs.annual`	annually occurring fixed costs
`profit.per.bv`	profit generated by bv per animal
`database`	Groups of individuals to consider
`gen`	Quick-insert for database (vector of all generations to consider)
`cohorts`	Quick-insert for database (vector of names of cohorts to consider)
`interest.rate`	Applied yearly interest rate
`base.gen`	Base generation (application of interest rate)

Value

Cost-table for selected gen/database/cohorts of a population-list

Examples

```
data(ex_pop)
computing.costs(ex_pop, gen=1:5)
```

`computing.costs.cohorts` *Compute costs of a breeding program by cohorts*

Description

Function to derive the costs of a breeding program / population-list by cohorts

Usage

```
computing.costs.cohorts(
  population,
  gen = NULL,
  database = NULL,
  cohorts = NULL,
  json = TRUE,
  phenotyping.costs = NULL,
  genotyping.costs = 0,
  housing.costs = NULL,
  fix.costs = 0,
  fix.costs.annual = 0,
  profit.per.bv = 1,
  interest.rate = 1,
  verbose = TRUE
)
```

Arguments

`population`	population-list
`gen`	Quick-insert for database (vector of all generations to consider)
`database`	Groups of individuals to consider
`cohorts`	Quick-insert for database (vector of names of cohorts to consider)
`json`	If TRUE extract which cohorts to plot according to the json-file used in json.simulation
`phenotyping.costs`	Costs for the generation of a phenotype
`genotyping.costs`	Costs for the generation of a genotype
`housing.costs`	Costs for housing
`fix.costs`	one time occurring fixed costs
`fix.costs.annual`	annually occurring fixed costs

profit.per.bv	profit generated by bv per animal
interest.rate	Applied yearly interest rate
verbose	Set to FALSE to not display any prints

Value

Cost-table for selected gen/database/cohorts of a population-list

Examples

```
data(ex_pop)
computing.costs.cohorts(ex_pop, gen=1:5, genotyping.costs=25, json=FALSE)
```

---

computing.snps *Compute genotype/haplotype*

Description

Internal function for the computation of genotypes & haplotypes

Usage

```
computing.snps(
  population,
  gen,
  sex,
  nr,
  faster = TRUE,
  import.position.calculation = NULL,
  from_p = 1,
  to_p = Inf,
  decodeOriginsU = decodeOriginsR,
  bit.storing = FALSE,
  nbits = 30,
  output_compressed = FALSE
)
```

Arguments

population	Population list
gen	Generation of the individual to compute
sex	Gender of the individual to compute
nr	Number of the individual to compute
faster	If FALSE use slower version to compute markers between recombination points
import.position.calculation	Function to calculate recombination point into adjacent/following SNP

from_p	First SNP to consider
to_p	Last SNP to consider
decodeOriginsU	Used function for the decoding of genetic origins [[5]]/[[6]]
bit.storing	Set to TRUE if the MoBPS (not-miraculix! bit-storing is used)
nbits	Bits available in MoBPS-bit-storing
output_compressed	Set to TRUE to get a miraculix-compressed genotype/haplotype

Value

haplotypes for the selected individual

Examples

```
data(ex_pop)
computing.snps(ex_pop, gen=1, sex=1, nr=1)
```

---

computing.snps_single *Compute genotype/haplotype in gene editing application*

Description

Internal function for the computation of genotypes & haplotypes in gene editing application

Usage

```
computing.snps_single(
  population,
  current.recombi,
  current.mut,
  current.ursprung,
  faster = TRUE,
  import.position.calculation = NULL,
  decodeOriginsU = decodeOriginsR
)
```

Arguments

population	Population list
current.recombi	vector of currently activ recombination points
current.mut	vector of currently activ mutations
current.ursprung	vector of currently activ origins
faster	If FALSE use slower version to compute markers between recombination points
import.position.calculation	Function to calculate recombination point into adjacent/following SNP
decodeOriginsU	Used function for the decoding of genetic origins [[5]]/[[6]]

Value

haplotypes for the selected individual

`creating.diploid` *Generation of the starting population*

Description

Generation of the starting population

Usage

```
creating.diploid(
  population = NULL,
  nsnp = 0,
  nindi = 0,
  nqtl = 0,
  name.cohort = NULL,
  generation = 1,
  founder.pool = 1,
  one.sex.mode = FALSE,
  database.sex.mode = FALSE,
  sex.s = "fixed",
  sex.quota = 0.5,
  class = 0L,
  verbose = TRUE,
  map = NULL,
  chr.nr = NULL,
  chromosome.length = NULL,
  bp = NULL,
  snps.equidistant = NULL,
  template.chip = NULL,
  snp.position = NULL,
  change.order = TRUE,
  add.chromosome = FALSE,
  bpcm.conversion = 0,
  snp.name = NULL,
  hom0 = NULL,
  hom1 = NULL,
  dataset = NULL,
  freq = "beta",
  beta.shape1 = 1,
  beta.shape2 = 1,
  share.genotyped = 0,
  genotyped.s = NULL,
  vcf = NULL,
```

```
vcf.maxsnp = Inf,
vcf.maxindi = Inf,
vcf.chromosomes = NULL,
vcf.VA = TRUE,
trait.name = NULL,
mean.target = NULL,
var.target = NULL,
qtl.position.shared = FALSE,
trait.cor = NULL,
trait.cor.include = NULL,
n.additive = 0,
n.equal.additive = 0,
n.dominant = 0,
n.equal.dominant = 0,
n.overdominant = 0,
n.equal.overdominant = 0,
n.qualitative = 0,
n.quantitative = 0,
effect.distribution = "gauss",
gamma.shape1 = 1,
gamma.shape2 = 1,
real.bv.add = NULL,
real.bv.mult = NULL,
real.bv.dice = NULL,
new.residual.correlation = NULL,
new.breeding.correlation = NULL,
litter.effect.covariance = NULL,
pen.effect.covariance = NULL,
is.maternal = NULL,
is.paternal = NULL,
fixed.effects = NULL,
trait.pool = 0,
gxe.correlation = NULL,
n.locations = NULL,
gxe.max = 0.85,
gxe.min = 0.7,
location.name = NULL,
gxe.combine = TRUE,
n.traits = 0,
base.bv = NULL,
dominant.only.positive = FALSE,
exclude.snps = NULL,
var.additive.l = NULL,
var.dominant.l = NULL,
var.overdominant.l = NULL,
var.qualitative.l = NULL,
var.quantitative.l = NULL,
effect.size.equal.add = 1,
```

```
  effect.size.equal.dom = 1,
  effect.size.equal.over = 1,
  polygenic.variance = 100,
  bve.mult.factor = NULL,
  bve.poly.factor = NULL,
  set.zero = FALSE,
  bv.standard = FALSE,
  replace.real.bv = FALSE,
  bv.ignore.traits = NULL,
  remove.invalid.qtl = TRUE,
  randomSeed = NULL,
  add.architecture = NULL,
  time.point = 0,
  creating.type = 0,
  size.scaling = 1,
  progress.bar = TRUE,
  miraculix = TRUE,
  miraculix.dataset = TRUE,
  add.chromosome.ends = TRUE,
  use.recalculate.manual = FALSE,
  store.comp.times = TRUE,
  skip.rest = FALSE,
  enter.bv = TRUE,
  internal = FALSE,
  internal.geno = TRUE,
  internal.dataset = NULL,
  nbits = 30,
  bit.storing = FALSE,
  new.phenotype.correlation = NULL,
  length.before = 5,
  length.behind = 5,
  position.scaling = FALSE,
  shuffle.cor = NULL,
  shuffle.traits = NULL,
  bv.total = 0
)
```

Arguments

population	Population list
nsnp	Number of markers to generate (Split equally across chromosomes (chr.nr) unless vector is used)
nindi	Number of individuals to generate (you can also provide number males / females in a vector)
nqtl	Number of QTLs to generate (this will be a subset of the generated SNPs; default: NULL; all SNPs are potential QTLs)
name.cohort	Name of the newly added cohort

generation	Generation to which newly individuals are added (default: 1)
founder.pool	Founder pool an individual is assign to (default: 1)
one.sex.mode	Activating this will ignore all sex specific parameters and handle each individual as part of the first sex (default: FALSE)
database.sex.mode	Set TRUE to automatically remove females in selection.m and remove males in selection.f
sex.s	Specify which newly added individuals are male (1) or female (2)
sex.quota	Share of newly added female individuals (deterministic if sex.s="fixed", alt: sex.s="random")
class	Migration level of the newly added individuals (default: 0)
verbose	Set to FALSE to not display any prints
map	map-file that contains up to 5 colums (chromosome, SNP-id, M-position, Bp-position, allele freq - Everything not provides it set to NA). A map can be imported via MoBPSmaps::ensembl.map()
chr.nr	Number of chromosomes (SNPs are equally split) or vector containing the associated chromosome for each marker
chromosome.length	Length of the newly added chromosome in Morgan; can be a vector when generating multiple chromosomes (default: 5)
bp	Vector containing the physical position (bp) for each marker (default: 1,2,3...)
snps.equidistant	Use equidistant markers (computationally faster! ; default: TRUE)
template.chip	Import genetic map and chip from a species ("cattle", "chicken", "pig")
snp.position	Location of each marker on the genetic map
change.order	Markers are automatically sorted according to their snp.position unless this is set to FALSE (default: TRUE)
add.chromosome	If TRUE add an additional chromosome to the population
bpcm.conversion	Convert physical position (bp) into a cM position (default: 0 - not done)
snp.name	Vector containing the name of each marker (default ChrXSNPY - XY chosen accordingly)
hom0	Vector containing the first allelic variant in each marker (default: 0)
hom1	Vector containing the second allelic variant in each marker (default: 1)
dataset	SNP dataset, use "random", "allhetero" "all0" when generating a dataset via nsnp,nindi
freq	frequency of allele 1 when randomly generating a dataset (default: "beta" with parameters beta.shape1, beta.shape2; Use "same" when generating additional individuals and using the same allele frequencies)
beta.shape1	First parameter of the beta distribution for simulating allele frequencies
beta.shape2	Second parameter of the beta distribution for simulating allele frequencies

share.genotyped
: Share of individuals genotyped in the founders

genotyped.s
: Specify with newly added individuals are genotyped (1) or not (0)

vcf
: Path to a vcf-file used as input genotypes (correct haplotype phase is assumed!)

vcf.maxsnp
: Maximum number of SNPs to include in the genotype file (default: Inf)

vcf.maxindi
: Maximum number of individuals to include in the genotype file (default: Inf)

vcf.chromosomes
: Vector of chromosomes to import from vcf. Use on bgziped and tabixed vcf only. (default: NULL - all chromosomes)

vcf.VA
: Use the VariantAnnotation package to load in a vcf file when available (default: TRUE)

trait.name
: Name of the traits generated

mean.target
: Target mean for each trait

var.target
: Target variance for each trait

qtl.position.shared
: Set to TRUE to put QTL effects on the same markers for different traits

trait.cor
: Target correlation between QTL-based traits (underlying true genomic values)

trait.cor.include
: Vector of traits to be included in the modelling of correlated traits (default: all - needs to match with trait.cor)

n.additive
: Number of additive QTL with effect size drawn from a gaussian distribution

n.equal.additive
: Number of additive QTL with equal effect size (effect.size)

n.dominant
: Number of dominant QTL with effect size drawn from a gaussian distribution

n.equal.dominant
: Number of dominant QTL with equal effect size

n.overdominant
: Number of overdominant QTL with effect size drawn from absolute value of a gaussian distribution

n.equal.overdominant
: Number of overdominant QTL with equal effect size

n.qualitative
: Number of qualitative epistatic QTL

n.quantitative
: Number of quantitative epistatic QTL

effect.distribution
: Set to "gamma" for gamma distribution effects with gamma.shape1, gamma.shape2 instead of gaussian (default: "gauss")

gamma.shape1
: Default: 1

gamma.shape2
: Default: 1

real.bv.add
: Single Marker effects (list for each trait with columns for: SNP Nr, Chr Nr, Effect 00, Effect 01, Effect 11, Position (optional), Founder pool genotype (optional), Founder pool origin (optional))

real.bv.mult
: Two Marker effects

`real.bv.dice` Multi-marker effects

`new.residual.correlation`
Correlation of the simulated enviromental variance

`new.breeding.correlation`
Correlation of the simulated genetic variance (child share! heritage is not influenced!

`litter.effect.covariance`
Covariance matrix of the litter effect (default: no effects)

`pen.effect.covariance`
Covariance matrix of the pen effect (default: no effects)

`is.maternal` Vector coding if a trait is caused by a maternal effect (Default: FALSE)

`is.paternal` Vector coding if a trait is caused by a paternal effect (Default: FALSE)

`fixed.effects` Matrix containing fixed effects (p x k -matrix with p being the number of traits and k being number of fixed effects; default: not fixed effects (NULL))

`trait.pool` Vector providing information for which pools QTLs of which trait are activ (default: 0 - all pools)

`gxe.correlation`
Correlation matrix between locations / environments (default: only one location, sampled from gxe.max / gxe.min)

`n.locations` Number of locations / environments to consider for the GxE model

`gxe.max` Maximum correlation between locations / environments when generating correlation matrix via sampling (default: 0.85)

`gxe.min` Minimum correlation between locations / environments when generating correlation matrix via sampling (default: 0.70)

`location.name` Same of the different locations / environments used

`gxe.combine` Set to FALSE to not view the same trait from different locations / environments as the sample trait in the prediction model (default: TRUE)

`n.traits` Number of traits (If more than traits via real.bv.X use traits with no directly underlying QTL)

`base.bv` Intercept of underlying true genomic values (excluding all QTL effects, default: 100)

`dominant.only.positive`
Set to TRUE to always assign the heterozygous variant with the higher of the two homozygous effects (e.g. hybrid breeding); default: FALSE

`exclude.snps` Vector contain markers on which no QTL effects are placed

`var.additive.l` Variance of additive QTL

`var.dominant.l` Variance of dominante QTL

`var.overdominant.l`
Variance of overdominante QTL

`var.qualitative.l`
Variance of qualitative epistatic QTL

`var.quantitative.l`
Variance of quantitative epistatic QTL

`effect.size.equal.add`
: Effect size of the QTLs in n.equal.additive

`effect.size.equal.dom`
: Effect size of the QTLs in n.equal.dominant

`effect.size.equal.over`
: Effect size of the QTLs in n.equal.overdominant

`polygenic.variance`
: Genetic variance of traits with no underlying QTL

`bve.mult.factor`
: Multiplicate trait value times this

`bve.poly.factor`
: Potency trait value over this

`set.zero`
: Set to TRUE to have no effect on the 0 genotype (or 00 for QTLs with 2 underlying SNPs)

`bv.standard`
: Set TRUE to standardize trait mean and variance via bv.standardization() - automatically set to TRUE when mean/var.target are used

`replace.real.bv`
: If TRUE delete the simulated traits added before

`bv.ignore.traits`
: Vector of traits to ignore in the calculation of the genomic value (default: NULL; Only recommended for high number of traits and experienced users!)

`remove.invalid.qtl`
: Set to FALSE to deactive the automatic removal of QTLs on markers that do not exist

`randomSeed`
: Set random seed of the process

`add.architecture`
: Add genetic architecture (marker positions)

`time.point`
: Time point at which the new individuals are generated

`creating.type`
: Technique to generate new individuals (usage in web-based application)

`size.scaling`
: Set to value to scale all input for breeding.size / selection.size (This will not work for all breeding programs / less general than json.simulation)

`progress.bar`
: Set to FALSE to not use progress bars in any application of breeding.diploid() downstream (Keep log-files lean!)

`miraculix`
: If TRUE use miraculix package for data storage, computations and dataset generation

`miraculix.dataset`
: Set FALSE to deactivate miraculix package for dataset generation

`add.chromosome.ends`
: Add chromosome ends as recombination points

`use.recalculate.manual`
: Set to TRUE to use recalculate.manual to calculate genomic values (all individuals and traits jointly, default: FALSE)

`store.comp.times`
: Set to FALSE to not store computing times needed to execute creating.diploid in $info$comp.times.creating

skip.rest	Internal variable needed when adding multiple chromosomes jointly
enter.bv	Internal parameter
internal	Do not touch!
internal.geno	Do not touch!
internal.dataset	Do not touch!
nbits	Bits available in MoBPS-bit-storing
bit.storing	Set to TRUE if the MoBPS (not-miraculix! bit-storing is used)
new.phenotype.correlation	(OLD! - use new.residual.correlation) Correlation of the simulated enviromental variance
length.before	Length before the first SNP of the dataset (default: 5)
length.behind	Length after the last SNP of the dataset (default: 5)
position.scaling	Manual scaling of snp.position
shuffle.cor	OLD! Use trait.cor - Target Correlation between traits
shuffle.traits	OLD! Use trait.cor.include - Vector of traits to be included for modelling of correlated traits (default: all - needs to match with shuffle.cor)
bv.total	OLD! Use n.traits instead. Number of traits (If more than traits via real.bv.X use traits with no directly underlying QTL)

Value

Population-list

Examples

```
population <- creating.diploid(nsnp=1000, nindi=100)
```

---

creating.phenotypic.transform

Create a phenotypic transformation

Description

Function to perform create a transformation of phenotypes

Usage

```
creating.phenotypic.transform(
  population,
  phenotypic.transform.function = NULL,
  trait = 1,
  test.h2 = TRUE,
  gen = NULL,
  database = NULL,
  cohorts = NULL,
  h2 = seq(0.05, 0.5, by = 0.05),
  export.h2 = FALSE,
  n.sample = 1000
)
```

Arguments

population	Population list
phenotypic.transform.function	Phenotypic transformation to apply
trait	Trait for which a transformation is to be applied
test.h2	Set to FALSE to not perform heritability check
gen	Quick-insert for database (vector of all generations to export)
database	Groups of individuals to consider for the export
cohorts	Quick-insert for database (vector of names of cohorts to export)
h2	Vector of heritability input to test (before introducing noise from trafo; default: seq(0.05,0.5, by = 0.05))
export.h2	Set TRUE to export matrix of heritability before/after transformation
n.sample	Sample size to use in test.h2 (default: 1000)

Value

Population-list with a new phenotypic transformation function

Examples

```
data(ex_pop)
trafo <- function(x){return(x^2)}
population <- creating.phenotypic.transform(ex_pop, phenotypic.transform.function=trafo,
n.sample = 100)
```

creating.trait *Generation of genomic traits*

Description

Generation of the trait in a starting population

Usage

```
creating.trait(
  population,
  trait.name = NULL,
  mean.target = NULL,
  var.target = NULL,
  qtl.position.shared = FALSE,
  trait.cor = NULL,
  trait.cor.include = NULL,
  n.additive = 0,
  n.equal.additive = 0,
  n.dominant = 0,
  n.equal.dominant = 0,
  n.overdominant = 0,
  n.equal.overdominant = 0,
  n.qualitative = 0,
  n.quantitative = 0,
  effect.distribution = "gauss",
  gamma.shape1 = 1,
  gamma.shape2 = 1,
  real.bv.add = NULL,
  real.bv.mult = NULL,
  real.bv.dice = NULL,
  n.traits = 0,
  base.bv = NULL,
  new.residual.correlation = NULL,
  new.breeding.correlation = NULL,
  is.maternal = NULL,
  is.paternal = NULL,
  fixed.effects = NULL,
  trait.pool = 0,
  gxe.correlation = NULL,
  n.locations = NULL,
  gxe.max = 0.85,
  gxe.min = 0.7,
  location.name = NULL,
  gxe.combine = TRUE,
  dominant.only.positive = FALSE,
  exclude.snps = NULL,
```

```
  var.additive.l = NULL,
  var.dominant.l = NULL,
  var.overdominant.l = NULL,
  var.qualitative.l = NULL,
  var.quantitative.l = NULL,
  effect.size.equal.add = 1,
  effect.size.equal.dom = 1,
  effect.size.equal.over = 1,
  polygenic.variance = 100,
  bve.mult.factor = NULL,
  bve.poly.factor = NULL,
  set.zero = FALSE,
  bv.standard = FALSE,
  replace.traits = FALSE,
  remove.invalid.qtl = TRUE,
  randomSeed = NULL,
  verbose = TRUE,
  use.recalculate.manual = NULL,
  new.phenotype.correlation = NULL,
  shuffle.traits = NULL,
  shuffle.cor = NULL,
  bv.total = 0
)
```

Arguments

population	Population list
trait.name	Name of the trait generated
mean.target	Target mean
var.target	Target variance
qtl.position.shared	Set to TRUE to put QTL effects on the same markers for different traits
trait.cor	Target correlation between QTL-based traits (underlying true genomic values)
trait.cor.include	Vector of traits to be included in the modelling of correlated traits (default: all - needs to match with trait.cor)
n.additive	Number of additive QTL with effect size drawn from a gaussian distribution
n.equal.additive	Number of additive QTL with equal effect size (effect.size)
n.dominant	Number of dominant QTL with effect size drawn from a gaussian distribution
n.equal.dominant	Number of dominant QTL with equal effect size
n.overdominant	Number of overdominant QTL with effect size drawn from absolute value of a gaussian distribution
n.equal.overdominant	Number of overdominant QTL with equal effect size

`n.qualitative` Number of qualitative epistatic QTL

`n.quantitative` Number of quantitative epistatic QTL

`effect.distribution`
Set to "gamma" for gamma distribution effects with gamma.shape1, gamma.shape2 instead of gaussian (default: "gauss")

`gamma.shape1` Default: 1

`gamma.shape2` Default: 1

`real.bv.add` Single Marker effects

`real.bv.mult` Two Marker effects

`real.bv.dice` Multi-marker effects

`n.traits` Number of traits (If more than traits via real.bv.X use traits with no directly underlying QTL)

`base.bv` Average genetic value of a trait

`new.residual.correlation`
Correlation of the simulated enviromental variance

`new.breeding.correlation`
Correlation of the simulated genetic variance (child share! heritage is not influenced!

`is.maternal` Vector coding if a trait is caused by a maternal effect (Default: all FALSE)

`is.paternal` Vector coding if a trait is caused by a paternal effect (Default: all FALSE)

`fixed.effects` Matrix containing fixed effects (p x k -matrix with p being the number of traits and k being number of fixed effects; default: p x 1 matrix with 0s (additional intercept))

`trait.pool` Vector providing information for which pools QTLs of this trait are activ (default: 0 - all pools)

`gxe.correlation`
Correlation matrix between locations / environments (default: only one location, sampled from gxe.max / gxe.min)

`n.locations` Number of locations / environments to consider for the GxE model

`gxe.max` Maximum correlation between locations / environments when generating correlation matrix via sampling (default: 0.85)

`gxe.min` Minimum correlation between locations / environments when generating correlation matrix via sampling (default: 0.70)

`location.name` Same of the different locations / environments used

`gxe.combine` Set to FALSE to not view the same trait from different locations / environments as the sample trait in the prediction model (default: TRUE)

`dominant.only.positive`
Set to TRUE to always assign the heterozygous variant with the higher of the two homozygous effects (e.g. hybrid breeding); default: FALSE

`exclude.snps` Marker were no QTL are simulated on

`var.additive.l` Variance of additive QTL

`var.dominant.l` Variance of dominante QTL

`var.overdominant.l`
Variance of overdominante QTL

`var.qualitative.l`
Variance of qualitative epistatic QTL

`var.quantitative.l`
Variance of quantitative epistatic QTL

`effect.size.equal.add`
Effect size of the QTLs in n.equal.additive

`effect.size.equal.dom`
Effect size of the QTLs in n.equal.dominant

`effect.size.equal.over`
Effect size of the QTLs in n.equal.overdominant

`polygenic.variance`
Genetic variance of traits with no underlying QTL

`bve.mult.factor`
Multiplicate trait value times this

`bve.poly.factor`
Potency trait value over this

`set.zero` Set to TRUE to have no effect on the 0 genotype (or 00 for QTLs with 2 underlying SNPs)

`bv.standard` Set TRUE to standardize trait mean and variance via bv.standardization()

`replace.traits` If TRUE delete the simulated traits added before

`remove.invalid.qtl`
Set to FALSE to deactive the automatic removal of QTLs on markers that do not exist

`randomSeed` Set random seed of the process

`verbose` Set to FALSE to not display any prints

`use.recalculate.manual`
Set to TRUE to use recalculate.manual to calculate genomic values (all individuals and traits jointly, default: FALSE)

`new.phenotype.correlation`
(OLD! - use new.residual.correlation) Correlation of the simulated enviromental variance

`shuffle.traits` OLD! Use trait.cor.include - Vector of traits to be included for modelling of correlated traits (default: all - needs to match with shuffle.cor)

`shuffle.cor` OLD! Use trait.cor - Target Correlation between traits

`bv.total` OLD! Use n.traits instead. Number of traits (If more than traits via real.bv.X use traits with no directly underlying QTL)

Value

Population-list with one or more additional new traits

Examples

```
population <- creating.diploid(nsnp=1000, nindi=100)
population <- creating.trait(population, n.additive=100)
```

`decodeOriginsR` *Origins-Decoding(R)*

Description

R-Version of the internal bitwise-decoding of origins

Usage

```
decodeOriginsR(P, row)
```

Arguments

`P`	coded origins vector
`row`	row to decode

Value

de-coded origins

Examples

```
decodeOriginsR(0L)
```

`demiraculix` *Remove miraculix-coding for genotypes*

Description

Internal function to decode all genotypes to non-miraculix objects

Usage

```
demiraculix(population)
```

Arguments

`population`	Population list

Value

Population list

Examples

```
# This is only relevant with the package miraculix is installed and used
population <- creating.diploid(nsnp=100, nindi=50)
population <- demiraculix(population)
```

`derive.loop.elements` *Derive loop elements*

Description

Internal function to derive the position of all individuals to consider for BVE/GWAS

Usage

```
derive.loop.elements(
  population,
  bve.database,
  bve.class,
  bve.avoid.duplicates,
  store.adding = FALSE,
  store.which.adding = FALSE,
  list.of.copys = FALSE
)
```

Arguments

`population`	Population list
`bve.database`	Groups of individuals to consider in breeding value estimation
`bve.class`	Consider only animals of those class classes in breeding value estimation (default: NULL - use all)
`bve.avoid.duplicates`	If set to FALSE multiple generatations of the same individual can be used in the bve (only possible by using copy.individual to generate individuals)
`store.adding`	Internal parameter to derive number of added individuals per database entry (only relevant internally for GWAS)
`store.which.adding`	Internal parameter to derive which individuals are copy entries
`list.of.copys`	Internal parameter to derive further information on the copies individuals

Value

Matrix of individuals in the entered database

Examples

```
data(ex_pop)
derive.loop.elements(ex_pop, bve.database=get.database(ex_pop, gen=2),
bve.class=NULL, bve.avoid.duplicates=TRUE)
```

diag.mobps *Diagonal matrix*

Description

Function to add a genotyping array for the population

Usage

```
diag.mobps(elements)
```

Arguments

elements	vector with entries to put on the diagonal of a matrix

Value

Diagonal matrix

Examples

```
diag.mobps(5)
```

edges.fromto *Detection of parental/child nodes*

Description

Internal function to extract parental/child node of an edge

Usage

```
edges.fromto(edges)
```

Arguments

edges	Edges of the json-file generated via the web-interface

Value

Matrix of Parent/Child-nodes for the considered edges

`edit_animal` *Internal gene editing function*

Description

Internal function to perform gene editing

Usage

```
edit_animal(
  population,
  gen,
  sex,
  nr,
  nr.edits,
  decodeOriginsU = decodeOriginsR,
  bit.storing = FALSE,
  nbits = 30
)
```

Arguments

`population`	Population list
`gen`	Generation of the individual to edit
`sex`	Gender of the individual to edit
`nr`	Number of the individual to edit
`nr.edits`	Number of edits to perform
`decodeOriginsU`	Used function for the decoding of genetic origins [[5]]/[[6]]
`bit.storing`	Set to TRUE if the MoBPS (not-miraculix! bit-storing is used)
`nbits`	Bits available in MoBPS-bit-storing

Value

animal after genome editing

effect.estimate.add *Estimation of marker effects*

Description

Function to estimate marker effects

Usage

```
effect.estimate.add(geno, pheno, map = NULL, scaling = TRUE)
```

Arguments

geno	genotype dataset (marker x individuals)
pheno	phenotype dataset (each phenotype in a row)
map	genomic map
scaling	Set FALSE to not perform variance scaling

Value

Empirical kinship matrix (IBD-based since Founders)

Examples

```
data(ex_pop)
pheno <- get.pheno(ex_pop, gen=1:5)
geno <- get.geno(ex_pop, gen=1:5)
map <- get.map(ex_pop, use.snp.nr=TRUE)
real.bv.add <- effect.estimate.add(geno, pheno, map)
```

effective.size *Estimate effective population size*

Description

Internal function to estimate the effective population size

Usage

```
effective.size(ld, dist, n)
```

Arguments

ld	ld between markers
dist	distance between markers in Morgan
n	Population size

Value

Estimated effective population size

`epi` *Martini-Test function*

Description

Internal function to perform martini test

Usage

```
epi(y, Z, G = NULL)
```

Arguments

`y`	y
`Z`	genomic information matrix
`G`	kinship matrix

Value

Estimated breeding values

`exist.cohort` *Function to extract if a cohort exists*

Description

Function to extract if a cohort exists

Usage

```
exist.cohort(population, cohort)
```

Arguments

`population`	Population list
`cohort`	Cohort to check if it is contained in the population list

Value

TRUE/FALSE

Examples

```
data(ex_pop)
exist.cohort(ex_pop, cohort = "StrangeName_42")
exist.cohort(ex_pop, cohort = "Cohort_1_M")
```

`ex_json` *ex_json*

Description

Exemplary json-data

Usage

```
ex_json
```

Author(s)

Torsten Pook <torsten.pook@wur.nl>

Source

Web-interface

`ex_pop` *ex_pop*

Description

Exemplary population-list

Usage

```
ex_pop
```

Author(s)

Torsten Pook <torsten.pook@wur.nl>

Source

MoBPS

find.chromo *Position detection (chromosome)*

Description

Internal function for the detection on which chromosome each marker is

Usage

```
find.chromo(position, length.total)
```

Arguments

position	position in the genome
length.total	Length of each chromosome

Value

Chromosome the marker is part of

find.snpbefore *Position detection (SNPs)*

Description

Internal function for the detection on which overall position each marker is

Usage

```
find.snpbefore(position, snp.position)
```

Arguments

position	Position on the genome
snp.position	Position of the SNPs on the genome

Value

SNP-position of the target position

`founder.simulation` *Founder simulation*

Description

Function to generate founder genotypes

Usage

```
founder.simulation(
  nindi = 100,
  sex.quota = 0.5,
  nsnp = 0,
  n.gen = 100,
  nfinal = NULL,
  sex.quota.final = NULL,
  big.output = FALSE,
  plot = TRUE,
  display.progress = TRUE,
  depth.pedigree = 7,
  dataset = NULL,
  vcf = NULL,
  chr.nr = NULL,
  bp = NULL,
  snp.name = NULL,
  hom0 = NULL,
  hom1 = NULL,
  bpcm.conversion = 0,
  freq = "beta",
  sex.s = "fixed",
  chromosome.length = NULL,
  length.before = 5,
  length.behind = 5,
  snps.equidistant = NULL,
  snp.position = NULL,
  position.scaling = FALSE,
  bit.storing = FALSE,
  nbits = 30,
  randomSeed = NULL,
  miraculix = TRUE,
  miraculix.dataset = TRUE,
  template.chip = NULL,
  beta.shape1 = 1,
  beta.shape2 = 1,
  map = NULL,
  verbose = TRUE,
  vcf.maxsnp = Inf,
```

```
  plot.ld = TRUE,
  plot.allele.freq = TRUE,
  xlim = NULL,
  ylim = NULL,
  nclass = 20,
  ylim.af = NULL,
  mutation.rate = 10^-8,
  remutation.rate = 10^-8,
  estimate.ne = TRUE,
  estimate.ld = TRUE
)
```

Arguments

nindi	number of individuals to generate in a random dataset
sex.quota	Share of newly added female individuals (deterministic if sex.s="fixed", alt: sex.s="random")
nsnp	number of markers to generate in a random dataset
n.gen	Number of generations to simulate (default: 100)
nfinal	Number of final individuals to include (default: nindi)
sex.quota.final	Share of female individuals in the final generation
big.output	Set to TRUE to export map, population list and pedigree relationship
plot	Set to FALSE to not generate LD-decay plot and allele frequency spectrum
display.progress	Set FALSE to not display progress bars. Setting verbose to FALSE will automatically deactive progress bars
depth.pedigree	Depth of the pedigree in generations (default: 7)
dataset	SNP dataset, use "random", "allhetero" "all0" when generating a dataset via nsnp,nindi
vcf	Path to a vcf-file used as input genotypes (correct haplotype phase is assumed!)
chr.nr	Vector containing the associated chromosome for each marker (default: all on the same)
bp	Vector containing the physical position (bp) for each marker (default: 1,2,3...)
snp.name	Vector containing the name of each marker (default ChrXSNPY - XY chosen accordingly)
hom0	Vector containing the first allelic variant in each marker (default: 0)
hom1	Vector containing the second allelic variant in each marker (default: 1)
bpcm.conversion	Convert physical position (bp) into a cM position (default: 0 - not done)
freq	frequency of allele 1 when randomly generating a dataset
sex.s	Specify which newly added individuals are male (1) or female (2)

`chromosome.length`
: Length of the newly added chromosome (default: 5)

`length.before`
: Length before the first SNP of the dataset (default: 5)

`length.behind`
: Length after the last SNP of the dataset (default: 5)

`snps.equidistant`
: Use equidistant markers (computationally faster! ; default: TRUE)

`snp.position`
: Location of each marker on the genetic map

`position.scaling`
: Manual scaling of snp.position

`bit.storing`
: Set to TRUE if the MoBPS (not-miraculix! bit-storing is used)

`nbits`
: Bits available in MoBPS-bit-storing

`randomSeed`
: Set random seed of the process

`miraculix`
: If TRUE use miraculix package for data storage, computations and dataset generation

`miraculix.dataset`
: Set FALSE to deactive miraculix package for dataset generation

`template.chip`
: Import genetic map and chip from a species ("cattle", "chicken", "pig")

`beta.shape1`
: First parameter of the beta distribution for simulating allele frequencies

`beta.shape2`
: Second parameter of the beta distribution for simulating allele frequencies

`map`
: map-file that contains up to 5 colums (Chromsome, SNP-id, M-position, Bp-position, allele freq - Everything not provides it set to NA). A map can be imported via MoBPSmaps::ensembl.map()

`verbose`
: Set to FALSE to not display any prints

`vcf.maxsnp`
: Maximum number of SNPs to include in the genotype file (default: Inf)

`plot.ld`
: Set FALSE to not generate the LD plot (default; TRUE)

`plot.allele.freq`
: Set FALSE to not generate the allele frequency spectrum plot (default: TRUE)

`xlim`
: Axis limits for the x-axis in the LD plot (default: NULL)

`ylim`
: Axis limits for the y-axis in the LD plot (default: NULL)

`nclass`
: Number of classes to consider in the allele frequency spectrum plot (default: 20)

`ylim.af`
: Axis limits for the allele frequency spectrum plot (default: NULL)

`mutation.rate`
: Mutation rate in each marker (default: 10^-8)

`remutation.rate`
: Remutation rate in each marker (default: 10^-8)

`estimate.ne`
: Set to FALSE to not estimate the effective population size (default: TRUE)

`estimate.ld`
: Set to FALSE to not estimate the ld decay (default: TRUE)

Examples

```
population <- founder.simulation(nindi=100, nsnp=1000, n.gen=5)
```

`generation.individual` *Function to generate a new individual*

Description

Function to generate a new individual

Usage

```
generation.individual(
  indexb,
  population,
  info_father_list,
  info_mother_list,
  copy.individual,
  mutation.rate,
  remutation.rate,
  recombination.rate,
  recom.f.indicator,
  duplication.rate,
  duplication.length,
  duplication.recombination,
  delete.same.origin,
  gene.editing,
  nr.edits,
  gen.architecture.m,
  gen.architecture.f,
  decodeOriginsU,
  current.gen,
  save.recombination.history,
  new.bv.child,
  dh.mating,
  share.genotyped,
  added.genotyped,
  genotyped.array,
  dh.sex,
  n.observation
)
```

Arguments

`indexb`	windows parallel internal test
`population`	windows parallel internal test
`info_father_list`	windows parallel internal test
`info_mother_list`	windows parallel internal test

copy.individual
: windows parallel internal test

mutation.rate
: windows parallel internal test

remutation.rate
: windows parallel internal test

recombination.rate
: windows parallel internal test

recom.f.indicator
: windows parallel internal test

duplication.rate
: windows parallel internal test

duplication.length
: windows parallel internal test

duplication.recombination
: windows parallel internal test

delete.same.origin
: windows parallel internal test

gene.editing
: windows parallel internal test

nr.edits
: windows parallel internal test

gen.architecture.m
: windows parallel internal test

gen.architecture.f
: windows parallel internal test

decodeOriginsU
: windows parallel internal test

current.gen
: windows parallel internal test

save.recombination.history
: windows parallel internal test

new.bv.child
: windows parallel internal test

dh.mating
: windows parallel internal test

share.genotyped
: windows parallel internal test

added.genotyped
: windows parallel internal test

genotyped.array
: windows parallel internal test

dh.sex
: windows parallel internal test

n.observation
: windows parallel internal test

Value

Offspring individual

`get.admixture` *Admixture Plot*

Description

Function to generate admixture plots

Usage

```
get.admixture(
  population,
  geno = NULL,
  gen = NULL,
  database = NULL,
  cohorts = NULL,
  id = NULL,
  d = NULL,
  verbose = TRUE,
  plot = TRUE,
  sort = FALSE,
  sort.cutoff = 0.01
)
```

Arguments

`population`	Population list
`geno`	Manually provided genotype dataset to use instead of gen/database/cohorts
`gen`	Quick-insert for database (vector of all generations to consider)
`database`	Groups of individuals to consider
`cohorts`	Quick-insert for database (vector of names of cohorts to consider)
`id`	Individual IDs to search/collect in the database
`d`	dimensions to consider in admixture plot (default: automatically estimate a reasonable number)
`verbose`	Set to FALSE to not display any prints
`plot`	Set to FALSE to not generate an admixture plot
`sort`	Set to TRUE to sort individuals according to contributes from the first dimension
`sort.cutoff`	Skip individuals with contributions under this threshold (and use next dimension instead) data(ex_pop) get.admixture(ex_pop, gen=4:6, d=2, sort=TRUE)

Value

Matrix with admixture proportion

`get.age.point` *Derive age point*

Description

Function to derive age point for each individual (Same as time.point unless copy.individual is used for aging)

Usage

```
get.age.point(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  use.id = TRUE
)
```

Arguments

`population`	Population list
`database`	Groups of individuals to consider for the export
`gen`	Quick-insert for database (vector of all generations to export)
`cohorts`	Quick-insert for database (vector of names of cohorts to export)
`id`	Individual IDs to search/collect in the database
`use.id`	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: TRUE)

Value

Time point selected gen/database/cohorts-individuals are born

Examples

```
data(ex_pop)
get.age.point(ex_pop, gen=2)
```

get.allele.freq *Calculate allele frequencies*

Description

Function to calculate allele frequencies

Usage

```
get.allele.freq(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL
)
```

Arguments

population	Population list
database	Groups of individuals to consider for the export
gen	Quick-insert for database (vector of all generations to export)
cohorts	Quick-insert for database (vector of names of cohorts to export)
id	Individual IDs to search/collect in the database

Value

Allele frequency of the alternative allele

Examples

```
data(ex_pop)
get.allele.freq(ex_pop, gen = 1)
```

get.bv *Export underlying true breeding values*

Description

Function to export underlying true breeding values

Usage

```
get.bv(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  use.id = TRUE
)
```

Arguments

population	Population list
database	Groups of individuals to consider for the export
gen	Quick-insert for database (vector of all generations to export)
cohorts	Quick-insert for database (vector of names of cohorts to export)
id	Individual IDs to search/collect in the database
use.id	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: TRUE)

Value

Genomic value of in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
get.bv(ex_pop, gen=2)
```

---

get.bve *Export estimated breeding values*

Description

Function to export estimated breeding values

Usage

```
get.bve(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  use.id = TRUE
)
```

Arguments

population	Population list
database	Groups of individuals to consider for the export
gen	Quick-insert for database (vector of all generations to export)
cohorts	Quick-insert for database (vector of names of cohorts to export)
id	Individual IDs to search/collect in the database
use.id	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: TRUE)

Value

Estimated breeding value of in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
get.bve(ex_pop, gen=2)
```

---

get.class *Derive class*

Description

Function to derive the class for each individual

Usage

```
get.class(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  use.id = TRUE
)
```

Arguments

population	Population list
database	Groups of individuals to consider for the export
gen	Quick-insert for database (vector of all generations to export)
cohorts	Quick-insert for database (vector of names of cohorts to export)
id	Individual IDs to search/collect in the database
use.id	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: TRUE)

Value

Class of in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
get.class(ex_pop, gen=2)
```

`get.cohorts` *Export Cohort-names*

Description

Function to export cohort names for the population list

Usage

```
get.cohorts(population, extended = FALSE)
```

Arguments

`population`	Population list
`extended`	extended cohorts

Value

List of all cohorts in the population-list

Examples

```
data(ex_pop)
get.cohorts(ex_pop)
```

`get.cohorts.individual`
Derive ID on an individual

Description

Function to derive the internal ID given to each individual

Usage

```
get.cohorts.individual(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  use.id = TRUE,
  keep.order = FALSE
)
```

Arguments

population	Population list
database	Groups of individuals to consider for the export
gen	Quick-insert for database (vector of all generations to export)
cohorts	Quick-insert for database (vector of names of cohorts to export)
id	Individual IDs to search/collect in the database
use.id	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names
keep.order	To not change order of individuals when ids are provided (default: TRUE)

Value

Individual ID for in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
get.cohorts.individual(ex_pop, gen=2)
```

---

get.computing.time *Derive computing time*

Description

Function to derive computing time required to generate input population list

Usage

```
get.computing.time(
  population,
  verbose = TRUE,
  extend = FALSE,
  per.call = FALSE
)
```

Arguments

population	Population list
verbose	Set to FALSE to not display any prints
extend	Set to TRUE to return computing times with detailled overview on generation and BVE (default: FALSE)
per.call	Set to TRUE to return computing times per call of breeding.diploid / creating.diploid() (default: FALSE)

Value

Computing times overview

Examples

```
data(ex_pop)
get.computing.time(ex_pop)
```

---

get.creating.type *Derive creating type*

Description

Function to derive creating type for each individual

Usage

```
get.creating.type(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  use.id = TRUE
)
```

Arguments

population	Population list
database	Groups of individuals to consider for the export
gen	Quick-insert for database (vector of all generations to export)
cohorts	Quick-insert for database (vector of names of cohorts to export)
id	Individual IDs to search/collect in the database
use.id	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: TRUE)

Value

Creating type of in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
get.creating.type(ex_pop, gen=2)
```

`get.culling.time` *Derive culling time*

Description

Function to derive time point of culling for each individual

Usage

```
get.culling.time(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  use.id = TRUE,
  use.all.copy = TRUE
)
```

Arguments

`population`	Population list
`database`	Groups of individuals to consider for the export
`gen`	Quick-insert for database (vector of all generations to export)
`cohorts`	Quick-insert for database (vector of names of cohorts to export)
`id`	Individual IDs to search/collect in the database
`use.id`	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: TRUE)
`use.all.copy`	Set to TRUE to extract phenotyping

Value

Class of in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
get.class(ex_pop, gen=2)
```

`get.culling.type` *Derive culling type*

Description

Function to derive the culling type for each individual

Usage

```
get.culling.type(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  use.id = FALSE
)
```

Arguments

`population`	Population list
`database`	Groups of individuals to consider for the export
`gen`	Quick-insert for database (vector of all generations to export)
`cohorts`	Quick-insert for database (vector of names of cohorts to export)
`id`	Individual IDs to search/collect in the database
`use.id`	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: FALSE)

Value

Class of in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
get.class(ex_pop, gen=2)
```

`get.cullingtime` *Derive time of culling*

Description

Function to derive the time of culling for all individuals

Usage

```
get.cullingtime(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  use.id = FALSE
)
```

Arguments

`population`	Population list
`database`	Groups of individuals to consider for the export
`gen`	Quick-insert for database (vector of all generations to export)
`cohorts`	Quick-insert for database (vector of names of cohorts to export)
`id`	Individual IDs to search/collect in the database
`use.id`	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: FALSE)

Value

Time of death of in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
get.cullingtime(ex_pop, gen=2)
```

get.database *gen/database/cohorts conversion*

Description

Function to derive a database based on gen/database/cohorts

Usage

```
get.database(
  population,
  gen = NULL,
  database = NULL,
  cohorts = NULL,
  avoid.merging = FALSE,
  per.individual = FALSE,
  id = NULL,
  db.names = NULL,
  id.all.copy = FALSE,
  id.last = FALSE,
  keep.order = FALSE,
  class = NULL,
  genotyped = NULL,
  npheno = NULL,
  verbose = TRUE,
  sex.filter = 0
)
```

Arguments

population	Population list
gen	Quick-insert for database (vector of all generations to export)
database	Groups of individuals to consider for the export
cohorts	Quick-insert for database (vector of names of cohorts to export)
avoid.merging	Set to TRUE to avoid different cohorts to be merged in a joint group when possible
per.individual	Set TRUE to obtain a database with one row per individual instead of concatenating (default: FALSE)
id	Individual IDs to search/collect in the database
db.names	MoPBS internal names (SexNr_Generation)
id.all.copy	Set to TRUE to show all copies of an individual in the database (default: FALSE)
id.last	Set to TRUE to use the last copy of an individual for the database (default: FALSE - pick first copy)
keep.order	To not change order of individuals when ids are provided (default: FALSE)

class	Only include individuals of the following classes in the database (can also be vector with multiple classes; default: ALL)
genotyped	Only include individuals that are genotyped (TRUE) or not-genotyped (FALSE); default: NULL (all individuals)
npheno	Only include individuals with the certain number of phenotypes generated (default: NULL (all individuals))
verbose	Set to FALSE to not display any prints
sex.filter	Set to 1 to only include males and set 2 to only include females in database (default: 0)

Value

Matrix with combined gen/database/cohorts

Examples

```
data(ex_pop)
get.database(ex_pop, gen=2)
```

---

get.death.point *Derive death point*

Description

Function to derive the time of death for each individual (NA for individuals that are still alive))

Usage

```
get.death.point(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  use.id = TRUE
)
```

Arguments

population	Population list
database	Groups of individuals to consider for the export
gen	Quick-insert for database (vector of all generations to export)
cohorts	Quick-insert for database (vector of names of cohorts to export)
id	Individual IDs to search/collect in the database
use.id	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: TRUE)

Value

Time of death of in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
get.death.point(ex_pop, gen=2)
```

`get.dendrogram` *Dendrogram*

Description

Function calculate a dendrogram

Usage

```
get.dendrogram(
  population,
  path = NULL,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  method = NULL,
  individual.names = NULL
)
```

Arguments

`population`	Population list
`path`	provide a path if the dendrogram would be saved as a png-file
`database`	Groups of individuals to consider
`gen`	Quick-insert for database (vector of all generations to consider)
`cohorts`	Quick-insert for database (vector of names of cohorts to consider)
`id`	Individual IDs to search/collect in the database
`method`	Method used to calculate genetic distances (default: "Nei", alt: "Rogers", "Prevosti", "Modified Rogers"
`individual.names`	Names of the individuals in the database ((default are MoBPS internal names based on position))

Value

Dendrogram plot for genotypes

Examples

```
data(ex_pop)
get.dendrogram(ex_pop, gen=2)
```

`get.dendrogram.heatmap` *Dendrogram Heatmap*

Description

Function calculate a dendrogram heat

Usage

```
get.dendrogram.heatmap(
  population,
  path = NULL,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  method = NULL,
  individual.names = NULL,
  traits = NULL,
  type = "pheno"
)
```

Arguments

`population`	Population list
`path`	provide a path if the dendrogram would be saved as a png-file
`database`	Groups of individuals to consider
`gen`	Quick-insert for database (vector of all generations to consider)
`cohorts`	Quick-insert for database (vector of names of cohorts to consider)
`id`	Individual IDs to search/collect in the database
`method`	Method used to calculate genetic distances (default: "Nei", alt: "Rogers", "Prevosti", "Modified Rogers"
`individual.names`	Names of the individuals in the database ((default are MoBPS internal names based on position))
`traits`	Traits to include in the dendrogram (default: all traits)
`type`	Which traits values to consider (default: "pheno", alt: "bv", "bve")

Value

Dendrogram plot of genotypes vs phenotypes

Examples

```
population <- creating.diploid(nsnp=1000, nindi=40, n.additive = c(100,100,100),
          trait.cor = matrix(c(1,0.8,0.2,0.8,1,0.2,0.2,0.2,1), ncol=3), shuffle.traits = 1:3)
population <- breeding.diploid(population, phenotyping = "all", heritability = 0.5)
get.dendrogram.heatmap(population, gen=1, type="pheno")
```

---

get.dendrogram.trait *Dendrogram*

Description

Function calculate a dendogram for the traits

Usage

```
get.dendrogram.trait(
  population,
  path = NULL,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  traits = NULL,
  type = "pheno"
)
```

Arguments

population	Population list
path	provide a path if the dendrogram would be saved as a png-file
database	Groups of individuals to consider
gen	Quick-insert for database (vector of all generations to consider)
cohorts	Quick-insert for database (vector of names of cohorts to consider)
id	Individual IDs to search/collect in the database
traits	Traits to include in the dendrogram (default: all traits)
type	Which traits values to consider (default: "pheno", alt: "bv", "bve")

Value

Dendrogram plot for traits

Examples

```
population <- creating.diploid(nsnp=1000, nindi=100, n.additive = c(100,100,100),
          shuffle.cor = matrix(c(1,0.8,0.2,0.8,1,0.2,0.2,0.2,1), ncol=3), shuffle.traits = 1:3)
population <- breeding.diploid(population, phenotyping = "all", heritability = 0.5)
get.dendrogram.trait(population, gen=1, type="pheno")
```

---

get.distance *Calculate Nei distance between two or more population*

Description

Function to calculate Nei's distance between two or more population

Usage

```
get.distance(
  population,
  type = "nei",
  marker = "all",
  per.marker = FALSE,
  gen1 = NULL,
  database1 = NULL,
  cohorts1 = NULL,
  id1 = NULL,
  gen2 = NULL,
  database2 = NULL,
  cohorts2 = NULL,
  id2 = NULL,
  database.list = NULL,
  gen.list = NULL,
  cohorts.list = NULL
)
```

Arguments

population	population list
type	Chose type of distance to compute (default: Neis standard genetic distance "nei"). Alt: Reynolds distance ("reynold"), Cavalli-Sforza ("cavalli"), Neis distance ("nei_distance"), Neis minimum distance ("nei_minimum")
marker	Vector with SNPs to consider (Default: "all" - use of all markers)
per.marker	Set to TRUE to return per marker statistics on genetic distances
gen1	Quick-insert for database (vector of all generations to consider)
database1	First Groups of individuals to consider
cohorts1	Quick-insert for database (vector of names of cohorts to consider)
id1	Individual IDs to search/collect in the database

gen2	Quick-insert for database (vector of all generations to consider)
database2	Second Groups of individuals to consider
cohorts2	Quick-insert for database (vector of names of cohorts to consider)
id2	Individual IDs to search/collect in the database
database.list	List of databases to consider (use when working with more than 2 populations)
gen.list	Quick-insert for database (vector of all generations to consider)
cohorts.list	Quick-insert for database (vector of names of cohorts to consider)

Value

Population list

Examples

```
data(ex_pop)
get.distance(ex_pop, database1 = cbind(1,1), database2 = cbind(1,2))
```

---

get.effect.freq *Compute marker frequency in QTL-markers*

Description

Function to compute marker frequency in QTL-markers

Usage

```
get.effect.freq(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  sort = FALSE
)
```

Arguments

population	Population list
database	Groups of individuals to consider for the export
gen	Quick-insert for database (vector of all generations to export)
cohorts	Quick-insert for database (vector of names of cohorts to export)
id	Individual IDs to search/collect in the database
sort	Set to FALSE to not sort markers according to position on the genome

Value

Matrix with allele frequencies in the QTLs

Examples

```
data(ex_pop)
get.effect.freq(ex_pop, gen=1)
```

`get.effective.size` *Estimate effective population size*

Description

Function to estimate the effective population size

Usage

```
get.effective.size(
  population,
  gen = NULL,
  database = NULL,
  cohorts = NULL,
  id = NULL
)
```

Arguments

`population`	Population list
`gen`	Quick-insert for database (vector of all generations to export)
`database`	Groups of individuals to consider for the export
`cohorts`	Quick-insert for database (vector of names of cohorts to export)
`id`	Individual IDs to search/collect in the database

Value

Estimated effective population size

Examples

```
data(ex_pop)
get.effective.size(population=ex_pop, gen=5)
```

`get.fixed.effects.p` *Export parametrization of fixed effects*

Description

Function to export parametrization of the fixed effects

Usage

```
get.fixed.effects.p(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  use.id = TRUE
)
```

Arguments

`population`	Population list
`database`	Groups of individuals to consider for the export
`gen`	Quick-insert for database (vector of all generations to export)
`cohorts`	Quick-insert for database (vector of names of cohorts to export)
`id`	Individual IDs to search/collect in the database
`use.id`	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: FALSE)

Value

Estimated breeding value of in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
population <- add.fixed.effects(ex_pop, fixed.effects = cbind(1,5))
population <- breeding.diploid(population, heritability = 0.3,
fixed.effects.p = rbind(c(1,0), c(0,1)), phenotyping.gen=2)
get.fixed.effects.p(population, gen=2)
```

get.geno *Derive genotypes of selected individuals*

Description

Function to derive genotypes of selected individuals

Usage

```
get.geno(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  chromosome = "all",
  export.alleles = FALSE,
  non.genotyped.as.missing = FALSE,
  use.id = TRUE,
  array = NULL,
  remove.missing = TRUE
)
```

Arguments

population	Population list
database	Groups of individuals to consider for the export
gen	Quick-insert for database (vector of all generations to export)
cohorts	Quick-insert for database (vector of names of cohorts to export)
id	Individual IDs to search/collect in the database
chromosome	Limit the genotype output to a selected chromosome (default: "all")
export.alleles	If TRUE export underlying alleles instead of just 012
non.genotyped.as.missing	Set to TRUE to replace non-genotyped markers with NA
use.id	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: TRUE)
array	Use only markers available on the array
remove.missing	Remove markers not genotyped in any individual from the export

Value

Genotype data for in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
geno <- get.geno(ex_pop, gen=2)
```

`get.geno.time` *Derive genotyping timepoint*

Description

Function to derive genotyping timepoint

Usage

```
get.geno.time(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  use.id = TRUE,
  use.all.copy = TRUE
)
```

Arguments

`population`	Population list
`database`	Groups of individuals to consider for the export
`gen`	Quick-insert for database (vector of all generations to export)
`cohorts`	Quick-insert for database (vector of names of cohorts to export)
`id`	Individual IDs to search/collect in the database
`use.id`	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: TRUE)
`use.all.copy`	Set to TRUE to extract phenotyping

Value

genotyping timepoint

Examples

```
data(ex_pop)
get.class(ex_pop, gen=2)
```

`get.genotyped` *Derive genotyping status*

Description

Function to if selected individuals are genotyped

Usage

```
get.genotyped(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  use.id = TRUE,
  use.all.copy = FALSE
)
```

Arguments

`population`	Population list
`database`	Groups of individuals to consider for the export
`gen`	Quick-insert for database (vector of all generations to export)
`cohorts`	Quick-insert for database (vector of names of cohorts to export)
`id`	Individual IDs to search/collect in the database
`use.id`	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: TRUE)
`use.all.copy`	Set to TRUE to extract phenotyping

Value

Check if in gen/database/cohorts selected individuals are genotyped

Examples

```
data(ex_pop)
get.genotyped(ex_pop, gen=2)
```

`get.genotyped.snp` *Derive which markers are genotyped of selected individuals*

Description

Function to derive which markers are genotyped for the selected individuals

Usage

```
get.genotyped.snp(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  export.alleles = FALSE,
  use.id = TRUE,
  array = NULL
)
```

Arguments

`population`	Population list
`database`	Groups of individuals to consider for the export
`gen`	Quick-insert for database (vector of all generations to export)
`cohorts`	Quick-insert for database (vector of names of cohorts to export)
`id`	Individual IDs to search/collect in the database
`export.alleles`	If TRUE export underlying alleles instead of just 012
`use.id`	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: TRUE)
`array`	Use only markers available on the array

Value

Binary Coded is/isnot genotyped level for in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
genotyped.snps <- get.genotyped.snp(ex_pop, gen=2)
```

get.haplo *Derive haplotypes of selected individuals*

Description

Function to derive haplotypes of selected individuals

Usage

```
get.haplo(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  chromosome = "all",
  export.alleles = FALSE,
  non.genotyped.as.missing = FALSE,
  use.id = TRUE,
  array = NULL,
  remove.missing = TRUE
)
```

Arguments

population	Population list
database	Groups of individuals to consider for the export
gen	Quick-insert for database (vector of all generations to export)
cohorts	Quick-insert for database (vector of names of cohorts to export)
id	Individual IDs to search/collect in the database
chromosome	Limit the genotype output to a selected chromosome (default: "all")
export.alleles	If TRUE export underlying alleles instead of just 012
non.genotyped.as.missing	Set to TRUE to replace non-genotyped markers with NA
use.id	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: TRUE)
array	Use only markers available on the array
remove.missing	Remove markers not genotyped in any individual from the export

Value

Haplotype data for in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
haplo <- get.haplo(ex_pop, gen=2)
```

`get.id` *Derive ID on an individual*

Description

Function to derive the internal ID given to each individual

Usage

```
get.id(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  use.id = TRUE,
  keep.order = FALSE
)
```

Arguments

`population`	Population list
`database`	Groups of individuals to consider for the export
`gen`	Quick-insert for database (vector of all generations to export)
`cohorts`	Quick-insert for database (vector of names of cohorts to export)
`id`	Individual IDs to search/collect in the database
`use.id`	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: TRUE)
`keep.order`	To not change order of individuals when ids are provided (default: FALSE)

Value

Individual ID for in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
get.id(ex_pop, gen=2)
```

get.index *Putting together indices for GxE / multi trait*

Description

Function to put together indices for GxE / multi trait

Usage

```
get.index(
  population,
  traits = NULL,
  locations = NULL,
  trait.weights = NULL,
  location.weights = NULL
)
```

Arguments

population	Population list
traits	Which traits to include in the index (all weight with factor 1)
locations	Which locations to include in the index (all weight weight factor 1)
trait.weights	Vector with a weight per trait
location.weights	Vector weight a weight per location

Value

Index

Examples

```
population = creating.diploid(nsnp =1000, nindi = 100)
population = creating.trait(population, n.additive = c(10,10), n.location=3, replace.traits = TRUE)
get.index(population, trait.weights = c(1,2), location.weights = c(1,2,3))
```

get.infos *Extract bv/pheno/geno of selected individuals*

Description

Function to extract bv/pheno/geno of selected individuals

Usage

```
get.infos(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  use.id = TRUE
)
```

Arguments

population	Population list
database	Groups of individuals to consider for the export
gen	Quick-insert for database (vector of all generations to export)
cohorts	Quick-insert for database (vector of names of cohorts to export)
id	Individual IDs to search/collect in the database
use.id	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: TRUE)

Value

Info list [[1]] phenotypes [[2]] genomic values [[3]] Z [[4/5/6]] additive/epistatic/dice marker effects

Examples

```
data(ex_pop)
get.infos(ex_pop, gen=2)
```

---

get.is.first *First copy*

Description

Function to derive if an individual is the first copy of itself in the database (based on position)

Usage

```
get.is.first(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  use.id = TRUE,
  keep.order = FALSE
)
```

Arguments

population	Population list
database	Groups of individuals to consider for the export
gen	Quick-insert for database (vector of all generations to export)
cohorts	Quick-insert for database (vector of names of cohorts to export)
id	Individual IDs to search/collect in the database
use.id	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: TRUE)
keep.order	To not change order of individuals when ids are provided (default: FALSE)

Value

Individual ID for in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
get.id(ex_pop, gen=2)
```

---

get.is.last *Last copy*

Description

Function to derive if an individual is the last copy of itself in the database (based on position)

Usage

```
get.is.last(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  use.id = TRUE,
  keep.order = FALSE
)
```

Arguments

population	Population list
database	Groups of individuals to consider for the export
gen	Quick-insert for database (vector of all generations to export)
cohorts	Quick-insert for database (vector of names of cohorts to export)

id	Individual IDs to search/collect in the database
use.id	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: TRUE)
keep.order	To not change order of individuals when ids are provided (default: FALSE)

Value

Logical vector

Examples

```
data(ex_pop)
get.is.last(ex_pop, gen=2)
```

`get.litter` *Export litter*

Description

Function to export litter ID of each individual

Usage

```
get.litter(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  use.id = TRUE
)
```

Arguments

population	Population list
database	Groups of individuals to consider for the export
gen	Quick-insert for database (vector of all generations to export)
cohorts	Quick-insert for database (vector of names of cohorts to export)
id	Individual IDs to search/collect in the database
use.id	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: TRUE)

Value

Litter ID for in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
get.litter(ex_pop, gen=2)
```

`get.litter.effect` *Export litter effect*

Description

Function to export the litter effect of selected individuals

Usage

```
get.litter.effect(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  use.id = TRUE
)
```

Arguments

`population`	Population list
`database`	Groups of individuals to consider for the export
`gen`	Quick-insert for database (vector of all generations to export)
`cohorts`	Quick-insert for database (vector of names of cohorts to export)
`id`	Individual IDs to search/collect in the database
`use.id`	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: TRUE)

Value

Litter effects for in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
get.litter.effect(ex_pop, gen=2)
```

get.maf *Calculate minor allele frequencies*

Description

Function to calculate minor allele frequencies

Usage

```
get.maf(population, database = NULL, gen = NULL, cohorts = NULL, id = NULL)
```

Arguments

population	Population list
database	Groups of individuals to consider for the export
gen	Quick-insert for database (vector of all generations to export)
cohorts	Quick-insert for database (vector of names of cohorts to export)
id	Individual IDs to search/collect in the database

Value

Allele frequency of the minor allele

Examples

```
data(ex_pop)
get.maf(ex_pop, gen = 1)
```

get.map *Map generation*

Description

Function to derive the genomic map for a given population list

Usage

```
get.map(population, use.snp.nr = FALSE, morgan.position.per.chromosome = TRUE)
```

Arguments

population	Population list
use.snp.nr	Set to TRUE to display SNP number and not SNP name
morgan.position.per.chromosome	Set to FALSE to Morgan position continuously over the genome

Value

Genomic map of the population list

Examples

```
data(ex_pop)
map <- get.map(ex_pop)
```

`get.ngen` *Number of generations*

Description

Function to calculate the number of generations in the population list

Usage

```
get.ngen(population)
```

Arguments

`population`	Population list

Value

Numeric value

Examples

```
data(ex_pop)
get.ngen(ex_pop)
```

`get.nindi` *Number of individuals*

Description

Function to calculate the number of individuals in the population list

Usage

```
get.nindi(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  extended = FALSE,
  count.copy = FALSE
)
```

Arguments

population	Population list
database	Groups of individuals to consider for the export
gen	Quick-insert for database (vector of all generations to export)
cohorts	Quick-insert for database (vector of names of cohorts to export)
id	Individual IDs to search/collect in the database
extended	Set to TRUE to export information on number of phenotyped, genotyped, and both pheno&genotyped individuals
count.copy	Set to TRUE to double count individuals if multiple copies of an individual are included in gen/database/cohorts

Value

Numeric value

Examples

```
data(ex_pop)
get.nindi(ex_pop, gen = 1)
```

---

get.npheno *Export underlying number of observations per phenotype*

Description

Function to export the number of observation of each underlying phenotype

Usage

```
get.npheno(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  use.all.copy = FALSE,
  use.id = TRUE
)
```

Arguments

population	Population list
database	Groups of individuals to consider for the export
gen	Quick-insert for database (vector of all generations to export)
cohorts	Quick-insert for database (vector of names of cohorts to export)
id	Individual IDs to search/collect in the database
use.all.copy	Set to TRUE to extract phenotyping
use.id	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: TRUE)

Value

Phenotypes for in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
get.pheno(ex_pop, gen=2)
```

---

get.ntrait *Number of traits*

Description

Function to calculate the number of traits in the population list

Usage

```
get.ntrait(population)
```

Arguments

population	Population list

Value

Numeric value

Examples

```
data(ex_pop)
get.ngen(ex_pop)
```

`get.pca` *Principle components analysis*

Description

Function to perform a principle component analysis

Usage

```
get.pca(
  population,
  path = NULL,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  coloring = "group",
  components = c(1, 2),
  plot = TRUE,
  pch = 1,
  export.color = FALSE,
  use.id = FALSE
)
```

Arguments

`population`	Population list
`path`	Location were to save the PCA-plot
`database`	Groups of individuals to consider for the export
`gen`	Quick-insert for database (vector of all generations to export)
`cohorts`	Quick-insert for database (vector of names of cohorts to export)
`id`	Individual IDs to search/collect in the database
`coloring`	Coloring by "group", "sex", "plain"
`components`	Default: c(1,2) for the first two principle components
`plot`	Set to FALSE to not generate a plot
`pch`	Point type in the PCA plot
`export.color`	Set to TRUE to export the per point coloring
`use.id`	Set TRUE to display IDs instead of dots in the PCA plot

Value

Principle components of gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
get.pca(ex_pop, gen=2)
```

`get.pedigree` *Derive pedigree*

Description

Derive pedigree for selected individuals

Usage

```
get.pedigree(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  founder.zero = TRUE,
  raw = FALSE,
  use.id = TRUE,
  id = NULL,
  use.first.copy = FALSE,
  include.error = FALSE,
  depth = 1
)
```

Arguments

`population`	Population list
`database`	Groups of individuals to consider for the export
`gen`	Quick-insert for database (vector of all generations to export)
`cohorts`	Quick-insert for database (vector of names of cohorts to export)
`founder.zero`	Parents of founders are displayed as "0" (default: TRUE)
`raw`	Set to TRUE to not convert numbers into Sex etc.
`use.id`	Set to TRUE to extract individual IDs
`id`	Individual IDs to search/collect in the database
`use.first.copy`	Set to TRUE to use database-position of the first copy of an individual (default: FALSE)
`include.error`	Set to TRUE to include errors simulated in the pedigree
`depth`	Depth (1) for parents, (2) for grandparents, (3) for grandgrandparents etc.

Value

Pedigree-file for in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
pedigree = get.pedigree(ex_pop, gen=2)
```

`get.pedigree.visual` *Draw pedigree*

Description

Draw a pedigree for selected individuals

Usage

```
get.pedigree.visual(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  depth.pedigree = 3,
  storage.save = 1.1,
  use.id = TRUE,
  cex = NULL,
  path = NULL,
  showgraph = TRUE,
  outline = FALSE,
  compact = FALSE
)
```

Arguments

`population`	Population list
`database`	Groups of individuals to consider for the export
`gen`	Quick-insert for database (vector of all generations to export)
`cohorts`	Quick-insert for database (vector of names of cohorts to export)
`id`	Individual IDs to search/collect in the database
`depth.pedigree`	Depth of the pedigree in generations
`storage.save`	The closer this is to 1 the more strict older animals will be filtered out of the pedigree (default: 1.1, min: 1)
`use.id`	Set to TRUE to extract individual IDs

cex	Size of individual labels
path	NULL or a character value means whether the pedigree graph will be saved in a pdf file. The graph in the pdf file is a legible vector drawing, and labels isn't overlapped especially when the number of individuals is big and width of the individual label is long in one generation. It is recommended that saving a pedigree graph in the pdf file. The default value is NULL (this is taken from visPedigree documentation).
showgraph	A logical value indicating whether a plot will be shown in the defaulted graphic device, such as the Plots panel of Rstudio. It is useful for quick viewing of the pedigree graph without opening the pdf file. However, the graph on the defaulted graphic device may be not legible, such as overlapped labels, aliasing lines due to the restricted width and height. It's a good choice to set showgraph = FALSE when the pedigree is large. The default value is TRUE (this is taken from visPedigree documentation).
outline	A logical value indicating whether shapes without label will be shown. A graph of the pedigree without individuals' label is shown when setting outline = TRUE. It is very useful for viewing the outline of the pedigree and finding the immigrant individuals in each generation when the width of a pedigree graph is longer than the maximum width (200 inches) of the pdf file. The defaulted value is FALSE (this is taken from visPedigree documentation).
compact	A logical value indicating whether IDs of full-sib individuals in one generation will be deleted and replaced with the number of full-sib individuals. For example, if there are 100 full-sib individuals in one generation, they will be deleted from the pedigree and be replaced with one individual label of "100" when compact = TRUE. The default value is FALSE (this is taken from visPedigree documentation).

Value

Pedigree visualization

Examples

```
population = creating.diploid(nsnp=100, nindi=10)
population = breeding.diploid(population, breeding.size=10)
population = breeding.diploid(population,  selection.m.database=cbind(c(1,2),1,1,5),
breeding.size=10)
population = breeding.diploid(population, selection.m.database=cbind(c(2,3),1,1,5),
 breeding.size=10)
get.pedigree.visual(population, gen=4)
```

---

get.pedigree2 *Derive pedigree including grandparents*

Description

Derive pedigree for selected individuals including grandparents

Usage

```
get.pedigree2(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  shares = FALSE,
  founder.zero = TRUE,
  raw = FALSE,
  include.error = FALSE
)
```

Arguments

population	Population list
database	Groups of individuals to consider for the export
gen	Quick-insert for database (vector of all generations to export)
cohorts	Quick-insert for database (vector of names of cohorts to export)
shares	Determine actual inherited shares of grandparents
founder.zero	Parents of founders are displayed as "0" (default: TRUE)
raw	Set to TRUE to not convert numbers into Sex etc.
include.error	Set to TRUE to include errors simulated in the pedigree

Value

Pedigree-file (grandparents) for in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
get.pedigree2(ex_pop, gen=3)
```

---

get.pedigree3 *Derive pedigree parents and grandparents*

Description

Derive pedigree for selected individuals including parents/grandparents

Usage

```
get.pedigree3(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  founder.zero = TRUE,
  id = FALSE,
  raw = FALSE,
  include.error = FALSE
)
```

Arguments

population	Population list
database	Groups of individuals to consider for the export
gen	Quick-insert for database (vector of all generations to export)
cohorts	Quick-insert for database (vector of names of cohorts to export)
founder.zero	Parents of founders are displayed as "0" (default: TRUE)
id	Set to TRUE to extract individual IDs
raw	Set to TRUE to not convert numbers into Sex etc.
include.error	Set to TRUE to include errors simulated in the pedigree

Value

Pedigree-file (parents + grandparents) for in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
get.pedigree3(ex_pop, gen=3)
```

---

get.pedigree_old *Derive pedigree*

Description

Derive pedigree for selected individuals

Usage

```
get.pedigree_old(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  founder.zero = TRUE,
  raw = FALSE,
  use.id = TRUE,
  id = NULL,
  use.first.copy = FALSE,
  include.error = FALSE,
  depth = 1
)
```

Arguments

population	Population list
database	Groups of individuals to consider for the export
gen	Quick-insert for database (vector of all generations to export)
cohorts	Quick-insert for database (vector of names of cohorts to export)
founder.zero	Parents of founders are displayed as "0" (default: TRUE)
raw	Set to TRUE to not convert numbers into Sex etc.
use.id	Set to TRUE to extract individual IDs
id	Replaced by use.id ((consistency with all other get.xxx functions))
use.first.copy	Set to TRUE to use database-position of the first copy of an individual (default: FALSE)
include.error	Set to TRUE to include errors simulated in the pedigree
depth	Depth (1) for parents, (2) for grandparents, (3) for grandgrandparents etc.

Value

Pedigree-file for in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
get.pedigree_old(ex_pop, gen=2)
```

get.pedmap *Generate plink-file (pedmap)*

Description

Generate a ped and map file (PLINK format) for selected groups and chromosome

Usage

```
get.pedmap(
  population,
  path = NULL,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  non.genotyped.as.missing = FALSE,
  use.id = TRUE
)
```

Arguments

population	Population list
path	Location to save pedmap-file
database	Groups of individuals to consider for the export
gen	Quick-insert for database (vector of all generations to export)
cohorts	Quick-insert for database (vector of names of cohorts to export)
id	Individual IDs to search/collect in the database
non.genotyped.as.missing	Set to TRUE to replaced non-genotyped entries with "./."
use.id	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: TRUE)

Value

Ped and map-file for in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)

file_path <- tempdir()
get.pedmap(path=file_path, ex_pop, gen=2)
file.remove(paste0(file_path, ".ped"))
file.remove(paste0(file_path, ".map"))
```

get.pen *Export pen ID*

Description

Function to export pen ID

Usage

```
get.pen(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  use.id = TRUE
)
```

Arguments

population	Population list
database	Groups of individuals to consider for the export
gen	Quick-insert for database (vector of all generations to export)
cohorts	Quick-insert for database (vector of names of cohorts to export)
id	Individual IDs to search/collect in the database
use.id	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: TRUE)

Value

Pen ID for in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
get.pen.effect(ex_pop, gen=2)
```

`get.pen.effect` *Export pen-effects*

Description

Function to export pen-effects

Usage

```
get.pen.effect(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  use.id = TRUE
)
```

Arguments

`population`	Population list
`database`	Groups of individuals to consider for the export
`gen`	Quick-insert for database (vector of all generations to export)
`cohorts`	Quick-insert for database (vector of names of cohorts to export)
`id`	Individual IDs to search/collect in the database
`use.id`	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: TRUE)

Value

Phenotypes for in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
get.pen.effect(ex_pop, gen=2)
```

`get.pheno` *Export underlying phenotypes*

Description

Function to export underlying phenotypes

Usage

```
get.pheno(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  use.all.copy = FALSE,
  use.id = TRUE
)
```

Arguments

`population`	Population list
`database`	Groups of individuals to consider for the export
`gen`	Quick-insert for database (vector of all generations to export)
`cohorts`	Quick-insert for database (vector of names of cohorts to export)
`id`	Individual IDs to search/collect in the database
`use.all.copy`	Set to TRUE to extract phenotyping
`use.id`	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: TRUE)

Value

Phenotypes for in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
get.pheno(ex_pop, gen=2)
```

`get.pheno.off` *Export underlying offspring phenotypes*

Description

Function to export offspring phenotypes

Usage

```
get.pheno.off(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  use.id = TRUE
)
```

Arguments

`population`	Population list
`database`	Groups of individuals to consider for the export
`gen`	Quick-insert for database (vector of all generations to export)
`cohorts`	Quick-insert for database (vector of names of cohorts to export)
`id`	Individual IDs to search/collect in the database
`use.id`	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: TRUE)

Value

Avg. phenotype of the offspring of in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
get.pheno.off(ex_pop, gen=2)
```

`get.pheno.off.count` *Export underlying number of used offspring for offspring phenotypes*

Description

Function to export number of observations used for offspring phenotypes

Usage

```
get.pheno.off.count(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  use.id = TRUE
)
```

Arguments

`population`	Population list
`database`	Groups of individuals to consider for the export
`gen`	Quick-insert for database (vector of all generations to export)
`cohorts`	Quick-insert for database (vector of names of cohorts to export)
`id`	Individual IDs to search/collect in the database
`use.id`	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: TRUE)

Value

Number of offspring with phenotypes for in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
get.pheno.off.count(ex_pop, gen=2)
```

`get.pheno.single` *Export underlying phenotypes*

Description

Function to export underlying phenotypes

Usage

```
get.pheno.single(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  use.all.copy = FALSE,
  use.id = TRUE
)
```

Arguments

`population`	Population list
`database`	Groups of individuals to consider for the export
`gen`	Quick-insert for database (vector of all generations to export)
`cohorts`	Quick-insert for database (vector of names of cohorts to export)
`id`	Individual IDs to search/collect in the database
`use.all.copy`	Set to TRUE to extract phenotyping
`use.id`	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: TRUE)

Value

Phenotypes for in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
get.pheno.single(ex_pop, gen=2)
```

`get.pheno.time` *Derive phenotyping time point*

Description

Function to derive timepoint of phenotyping for each individual

Usage

```
get.pheno.time(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  use.id = TRUE,
  use.all.copy = TRUE
)
```

Arguments

`population`	Population list
`database`	Groups of individuals to consider for the export
`gen`	Quick-insert for database (vector of all generations to export)
`cohorts`	Quick-insert for database (vector of names of cohorts to export)
`id`	Individual IDs to search/collect in the database
`use.id`	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: TRUE)
`use.all.copy`	Set to TRUE to extract phenotyping

Value

Timepoint (of phenotyping) of in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
get.pheno.time(ex_pop, gen=2)
```

`get.phylogenetic.tree` *Phylogenetic Tree*

Description

Function calculate a phylogenetic tree

Usage

```
get.phylogenetic.tree(
  population,
  path = NULL,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  method = NULL,
  use.id = TRUE,
  circular = FALSE
)
```

Arguments

`population`	Population list
`path`	provide a path if the dendrogram would be saved as a png-file
`database`	Groups of individuals to consider
`gen`	Quick-insert for database (vector of all generations to consider)
`cohorts`	Quick-insert for database (vector of names of cohorts to consider)
`id`	Individual IDs to search/collect in the database
`method`	Method used to calculate genetic distances (default: "Nei", alt: "Rogers", "Prevosti", "Modified Rogers"
`use.id`	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: TRUE)
`circular`	Set to TRUE to generate a fan/circular layout tree

Value

Dendrogram plot for traits

Examples

```
data(ex_pop)
get.phylogenetic.tree(ex_pop, gen=1, circular=TRUE)
```

get.plink *Generate binary plink-file*

Description

Generate a binary plink-file for selected groups and chromosome

Usage

```
get.plink(
  population,
  path = NULL,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  chromosome = "all",
  non.genotyped.as.missing = FALSE,
  fam.id = FALSE,
  type = 0,
  use.id = TRUE,
  bve.pedigree.error = TRUE
)
```

Arguments

population	Population list
path	Location to save vcf-file
database	Groups of individuals to consider for the export
gen	Quick-insert for database (vector of all generations to export)
cohorts	Quick-insert for database (vector of names of cohorts to export)
id	Individual IDs to search/collect in the database
chromosome	Limit the genotype output to a selected chromosome (default: "all")
non.genotyped.as.missing	Set to TRUE to replaced non-genotyped entries with "./."
fam.id	Set TRUE to set the fam-ID to the individual ID
type	Set 1 to only write paternal haplotype, set 2 to only write maternal haplotype, set 0 for both (default)
use.id	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: TRUE)
bve.pedigree.error	Set to FALSE to ignore/correct for any pedigree errors

Value

binary plink-file for in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
data(ex_pop)

file_path <- tempdir()
get.vcf(path=file_path, ex_pop, gen=2)
file.remove(paste0(file_path, ".vcf"))
```

---

get.pool *Export founder pool*

Description

Function to export founder pool

Usage

```
get.pool(
  population,
  gen = NULL,
  database = NULL,
  cohorts = NULL,
  id = NULL,
  plot = FALSE,
  import.position.calculation = NULL,
  decodeOriginsU = decodeOriginsR,
  use.id = TRUE
)
```

Arguments

population	Population list
gen	Quick-insert for database (vector of all generations to export)
database	Groups of individuals to consider for the export
cohorts	Quick-insert for database (vector of names of cohorts to export)
id	Individual IDs to search/collect in the database
plot	Set TRUE to generate a visualization of genetic origins
import.position.calculation	Function to calculate recombination point into adjacent/following SNP
decodeOriginsU	Used function for the decoding of genetic origins [[5]]/[[6]]
use.id	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: TRUE)

Value

Founder pool of in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
get.pool(ex_pop, gen=2)
```

`get.pool.founder` *Derive founder pool*

Description

Function to derive the founder pool of individuals

Usage

```
get.pool.founder(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  use.id = TRUE
)
```

Arguments

`population`	Population list
`database`	Groups of individuals to consider for the export
`gen`	Quick-insert for database (vector of all generations to export)
`cohorts`	Quick-insert for database (vector of names of cohorts to export)
`id`	Individual IDs to search/collect in the database
`use.id`	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: TRUE)

Value

Founder pool of in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
get.pool.founder(ex_pop, gen=2)
```

get.qtl *QTL extraction*

Description

Function to the position of QTLs (for snp/chr use get.qtl.effects()

Usage

```
get.qtl(population)
```

Arguments

population	Population list

Value

Vector of SNP positions

Examples

```
data(ex_pop)
positions <- get.qtl(ex_pop)
```

get.qtl.effects *QTL effect extraction*

Description

Function to extract QTL effect sizes

Usage

```
get.qtl.effects(population)
```

Arguments

population	Population list

Value

List with [[1]] single SNP QTLs [[2]] epistatic SNP QTLs [[3]] dice QTL

Examples

```
data(ex_pop)
effects <- get.qtl.effects(ex_pop)
```

get.qtl.variance *QTL effect variance extraction*

Description

Function to extract QTL effect variance for single SNP QTLs in a given gen/database/cohort

Usage

```
get.qtl.variance(
  population,
  gen = NULL,
  database = NULL,
  cohorts = NULL,
  id = NULL
)
```

Arguments

population	Population list
gen	Quick-insert for database (vector of all generations to consider)
database	Groups of individuals to consider
cohorts	Quick-insert for database (vector of names of cohorts to consider)
id	Individual IDs to search/collect in the database

Value

matrix with SNP / Chr / estimated effect variance

Examples

```
data(ex_pop)
effects <- get.qtl.variance(ex_pop)
```

get.recombi *Derive genetic origins*

Description

Function to derive genetic origin

Usage

```
get.recombi(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  use.id = TRUE
)
```

Arguments

population	Population list
database	Groups of individuals to consider for the export
gen	Quick-insert for database (vector of all generations to export)
cohorts	Quick-insert for database (vector of names of cohorts to export)
id	Individual IDs to search/collect in the database
use.id	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: TRUE)

Value

Recombination points for in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
get.recombi(ex_pop, gen=2)
```

---

get.reliability *Export underlying reliabilities*

Description

Function to export underlying reliabilities

Usage

```
get.reliability(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  use.id = TRUE
)
```

Arguments

population	Population list
database	Groups of individuals to consider for the export
gen	Quick-insert for database (vector of all generations to export)
cohorts	Quick-insert for database (vector of names of cohorts to export)
id	Individual IDs to search/collect in the database
use.id	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: TRUE)

Value

Estimated reliability for BVE for in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
get.reliability(ex_pop, gen=2)
```

---

get.selectionbve *Export derived breeding values based on the selection index*

Description

Function to export last breeding values based on the selection index

Usage

```
get.selectionbve(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  use.id = TRUE
)
```

Arguments

population	Population list
database	Groups of individuals to consider for the export
gen	Quick-insert for database (vector of all generations to export)
cohorts	Quick-insert for database (vector of names of cohorts to export)
id	Individual IDs to search/collect in the database
use.id	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: TRUE)

Value

Last applied selection index for in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
get.selectionbve(ex_pop, gen=2)
```

`get.selectionindex` *Export underlying last used selection index*

Description

Function to export last used selection index (mostly relevant for Miesenberger 1997 stuff)

Usage

```
get.selectionindex(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  use.id = TRUE
)
```

Arguments

`population`	Population list
`database`	Groups of individuals to consider for the export
`gen`	Quick-insert for database (vector of all generations to export)
`cohorts`	Quick-insert for database (vector of names of cohorts to export)
`id`	Individual IDs to search/collect in the database
`use.id`	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: TRUE)

Value

Last applied selection index for in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
get.selectionindex(ex_pop, gen=2)
```

get.sex *Extraction of individual sex*

Description

Function to extract the sex of selected individuals

Usage

```
get.sex(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  use.id = TRUE,
  male.female.coding = F
)
```

Arguments

population	Population list
database	Groups of individuals to consider for the export
gen	Quick-insert for database (vector of all generations to export)
cohorts	Quick-insert for database (vector of names of cohorts to export)
id	Individual IDs to search/collect in the database
use.id	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names
male.female.coding	Set to TRUE to display male/female instead of 1/2

Examples

```
data(ex_pop)
get.sex(ex_pop, gen=2)
```

get.size *Number of individuals in each generation*

Description

Function to calculate the number of individuals per generation

Function to calculate the number of individuals per generation

Usage

```
get.size(population)

get.size(population)
```

Arguments

population	Population list

Value

matrix with numeric values

matrix with numeric values

Examples

```
data(ex_pop)
get.size(ex_pop)
data(ex_pop)
get.size(ex_pop)
```

---

get.snapshot *Derive snapshot of selected individuals*

Description

Function to derive snapshot of genotyping/phenotyping state of selected individuals

Usage

```
get.snapshot(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  phenotype.data = FALSE,
  gain.data = FALSE,
  digits = 3,
  use.all.copy = TRUE,
  time.diff = NA
)
```

Arguments

population	Population list
database	Groups of individuals to consider for the export
gen	Quick-insert for database (vector of all generations to export)
cohorts	Quick-insert for database (vector of names of cohorts to export)
id	Individual IDs to search/collect in the database
phenotype.data	Set to TRUE to include information of number of phenotypes generated
gain.data	Set to TRUE to add information on changes in genetic level between cohorts (default: FALSE)
digits	Number of digits provided for the gain.data output (default: 3)
use.all.copy	Set to TRUE to extract phenotyping
time.diff	Set to a target time interval to receive information between transitions to other cohorts (default: NA)

Value

Snapshot-matrix

Examples

```
data(ex_pop)
get.snapshot(ex_pop, gen = 2)
```

---

get.snapshot.single *Derive snapshot of selected individuals*

Description

Function to derive snapshot of genotyping/phenotyping state of selected individuals

Usage

```
get.snapshot.single(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  phenotype.data = FALSE,
  gain.data = FALSE,
  digits = 3,
  time.diff = 1,
  min.time = -Inf,
  max.time = Inf,
```

```
  use.all.copy = TRUE,
  verbose = TRUE,
  time.points = NULL,
  include.culled = FALSE
)
```

Arguments

population	Population list
database	Groups of individuals to consider for the export
gen	Quick-insert for database (vector of all generations to export)
cohorts	Quick-insert for database (vector of names of cohorts to export)
id	Individual IDs to search/collect in the database
phenotype.data	Set to TRUE to include information of number of phenotypes generated
gain.data	Set to TRUE to add information on changes in genetic level between cohorts (default: FALSE)
digits	Number of digits provided for the gain.data output (default: 3)
time.diff	Set to a target time interval to receive information between transitions to other cohorts (default: NA)
min.time	Earliest time point relevant for output results (default: -Inf)
max.time	Latest time point relevant for output results (default: Inf)
use.all.copy	Set to TRUE to extract phenotyping
verbose	Set to FALSE to not display any prints (default: TRUE)
time.points	Use this parameter to manual provide a vector of time points on results should be generated for (default: NULL)
include.culled	Set to TRUE to also include culled individuals in the statistics provided

Value

Snapshot Matrix

Examples

```
data(ex_pop)
get.snapshot.single(ex_pop, cohorts = "Cohort_2_M")
```

`get.time.point` *Derive time point of generation*

Description

Function to derive time point of generation for each individual

Usage

```
get.time.point(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  use.id = TRUE
)
```

Arguments

`population`	Population list
`database`	Groups of individuals to consider for the export
`gen`	Quick-insert for database (vector of all generations to export)
`cohorts`	Quick-insert for database (vector of names of cohorts to export)
`id`	Individual IDs to search/collect in the database
`use.id`	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: TRUE)

Value

Time point of generation for in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
get.time.point(ex_pop, gen=2)
```

`get.trafo.p` *Export parametrization of fixed effects*

Description

Function to export parametrization of the fixed effects in transformation function

Usage

```
get.trafo.p(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  use.id = TRUE,
  include = NULL
)
```

Arguments

`population`	Population list
`database`	Groups of individuals to consider for the export
`gen`	Quick-insert for database (vector of all generations to export)
`cohorts`	Quick-insert for database (vector of names of cohorts to export)
`id`	Individual IDs to search/collect in the database
`use.id`	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: FALSE)
`include`	Which observations to include in the calculation (default: NULL (all))

Value

Parameter input used in the transformation function

Examples

```
data(ex_pop)
get.trafo.p(ex_pop, gen = 2)
```

`get.trafo.p.single` *Export parametrization of fixed effects*

Description

Function to export parametrization of the fixed effects in transformation function

Usage

```
get.trafo.p.single(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  use.id = TRUE
)
```

Arguments

`population`	Population list
`database`	Groups of individuals to consider for the export
`gen`	Quick-insert for database (vector of all generations to export)
`cohorts`	Quick-insert for database (vector of names of cohorts to export)
`id`	Individual IDs to search/collect in the database
`use.id`	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: FALSE)

Value

List with transformation parameters per observation

Examples

```
data(ex_pop)
get.trafo.p.single(ex_pop, gen = 2)
```

get.trait.name *Name of traits*

Description

Function to export trait names in the population list

Usage

```
get.trait.name(population)
```

Arguments

population	Population list

Value

Numeric value

Examples

```
data(ex_pop)
get.ngen(ex_pop)
```

get.uhat *Export estimated SNP effects*

Description

Function to export estimated SNP effects

Usage

```
get.uhat(population, extend = FALSE, plot = TRUE, trait.plot = 1)
```

Arguments

population	Population list
extend	Set to TRUE to export u_hat estimates from all breeding values instead of just the last (default: FALSE)
plot	Set to FALSE to not display overview of estimated SNP effects (default: TRUE)
trait.plot	Select trait for which to generate the visualization (default: 1)

Value

matrix with estimated marker effects

Examples

```
data(ex_pop)
population = breeding.diploid(ex_pop, bve.gen = 2:3, genotyped.gen = 2:3,
 bve = TRUE, estimate.u = TRUE)
get.uhat(population)
```

`get.variance` *Derive variances components (add/dom)*

Description

Function to derive underlying variance components (add/dom)

Usage

```
get.variance(
  population,
  gen = NULL,
  database = NULL,
  cohorts = NULL,
  id = NULL
)
```

Arguments

`population`	Population list
`gen`	Quick-insert for database (vector of all generations to export)
`database`	Groups of individuals to consider for the export
`cohorts`	Quick-insert for database (vector of names of cohorts to export)
`id`	Individual IDs to search/collect in the database

Value

Table with realized narrow/broad-sense heritability, sigma_g,a,d

Examples

```
data(ex_pop)
get.variance(ex_pop, gen = 2)
```

get.variance.components *Derive variance components*

Description

Function to derive variance components

Usage

```
get.variance.components(population)
```

Arguments

population	Population list

Value

Estimated variance components

Examples

```
data(ex_pop)
population = breeding.diploid(ex_pop, bve = TRUE, bve.gen = 2)
get.variance.components(population)
```

get.vcf *Generate vcf-file*

Description

Generate a vcf-file for selected groups and chromosome

Usage

```
get.vcf(
  population,
  path = NULL,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  id = NULL,
  chromosome = "all",
  non.genotyped.as.missing = FALSE,
  use.id = FALSE,
  file.append = FALSE
)
```

Arguments

population	Population list
path	Location to save vcf-file
database	Groups of individuals to consider for the export
gen	Quick-insert for database (vector of all generations to export)
cohorts	Quick-insert for database (vector of names of cohorts to export)
id	Individual IDs to search/collect in the database
chromosome	Limit the genotype output to a selected chromosome (default: "all")
non.genotyped.as.missing	Set to TRUE to replaced non-genotyped entries with "./."
use.id	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names
file.append	Set extend an existing vcf-file ((without writting a header))

Value

VCF-file for in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)

file_path <- tempdir()
get.vcf(path=file_path, ex_pop, gen=2)
file.remove(paste0(file_path, ".vcf"))
```

---

group.diff *Exclude individuals from a database*

Description

Function to exclude individuals from a database

Usage

```
group.diff(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  remove.gen = NULL,
  remove.database = NULL,
  remove.cohorts = NULL
)
```

Arguments

population	Population list
database	Groups of individuals to consider for the export
gen	Quick-insert for database (vector of all generations to export)
cohorts	Quick-insert for database (vector of names of cohorts to export)
remove.gen	Generations of individuals to remove from the database (same IDs!)
remove.database	Groups of individuals to remove from the database (same IDs!)
remove.cohorts	Cohorts of individuals to remove from the database (same IDs!)

Value

Database excluding removals

Examples

```
data(ex_pop)
database <- group.diff(ex_pop, gen=1, remove.database=cbind(1,1))
```

---

inbreeding.emp *Empirical kinship*

Description

Function to compute empirical kinship for a set of individuals)

Usage

```
inbreeding.emp(
  population = NULL,
  animals = NULL,
  gen = NULL,
  database = NULL,
  cohorts = NULL,
  use.id = TRUE
)
```

Arguments

population	Population list
animals	List of animals to compute kinship for
gen	Quick-insert for database (vector of all generations to export)
database	Groups of individuals to consider for the export
cohorts	Quick-insert for database (vector of names of cohorts to export)
use.id	Set TRUE to use animal IDs for column/row-names in the output matrix (default: TRUE)

Value

Empirical kinship matrix (IBD-based since Founders)

Examples

```
data(ex_pop)
inbreeding <- inbreeding.emp(population=ex_pop, database=cbind(3,1,1,25))
```

`inbreeding.exp` *Expected inbreeding*

Description

Function to derive pedigree based inbreeding

Usage

```
inbreeding.exp(
  population,
  gen = NULL,
  database = NULL,
  cohorts = NULL,
  depth.pedigree = 7,
  start.kinship = NULL,
  elements = NULL,
  storage.save = 1.5,
  verbose = TRUE
)
```

Arguments

`population`	Population list
`gen`	Quick-insert for database (vector of all generations to export)
`database`	Groups of individuals to consider for the export
`cohorts`	Quick-insert for database (vector of names of cohorts to export)
`depth.pedigree`	Depth of the pedigree in generations
`start.kinship`	Relationship matrix of the individuals in the first considered generation
`elements`	Vector of individuals from the database to include in pedigree matrix
`storage.save`	Lower numbers will lead to less memory but slightly higher computing time (default: 1.5, min: 1)
`verbose`	Set to FALSE to not display any prints

Value

Pedigree-based inbreeding in gen/database/cohort selected individuals

Examples

```
data(ex_pop)
inbreeding <- inbreeding.exp(population=ex_pop, gen=5)
```

`insert.bv` *Manually enter breeding values*

Description

Function to manually enter breeding values

Usage

```
insert.bv(
  population,
  bvs,
  na.override = FALSE,
  count = 1,
  count.only.increase = TRUE
)
```

Arguments

`population`	Population list
`bvs`	Matrix of phenotypes to enter (one row per individual with 1 element coding individual name)
`na.override`	Set to TRUE to also enter NA values (Default: FALSE - those entries will be skipped)
`count`	Counting for economic cost calculation (default: 1 - (one observation (for "pheno"), one genotyping (for "bve")))
`count.only.increase`	Set to FALSE to reduce the number of observation for a phenotype to "count" (default: TRUE)

Value

Population-List with newly entered breeding values

Examples

```
data(ex_pop)
bv <- get.bv(ex_pop, gen=2, use.id = FALSE)
new.bve <- cbind( colnames(bv), bv[,1]) ## Unrealistic but you do not get better than this!
ex_pop <- insert.bv(ex_pop, bvs=new.bve)
```

`insert.bve` *Manually enter estimated breeding values*

Description

Function to manually enter estimated breeding values

Usage

```
insert.bve(
  population,
  bves,
  type = "bve",
  na.override = FALSE,
  count = 1,
  count.only.increase = TRUE
)
```

Arguments

`population`	Population list
`bves`	Matrix of breeding values to enter (one row per individual with 1 element coding individual name)
`type`	which time of values to input (default: "bve", alt: "bv", "pheno")
`na.override`	Set to TRUE to also enter NA values (Default: FALSE - those entries will be skipped)
`count`	Counting for economic cost calculation (default: 1 - (one observation (for "pheno"), one genotyping (for "bve")))
`count.only.increase`	Set to FALSE to reduce the number of observation for a phenotype to "count" (default: TRUE)

Value

Population-List with newly entered estimated breeding values

Examples

```
data(ex_pop)
bv <- get.bv(ex_pop, gen=2, use.id = FALSE)
new.bve <- cbind( colnames(bv), bv[,1]) ## Unrealistic but you do not get better than this!
ex_pop <- insert.bve(ex_pop, bves=new.bve)
```

`insert.pheno` *Manually enter phenotypes*

Description

Function to manually enter phenotypes

Usage

```
insert.pheno(
  population,
  phenos,
  na.override = FALSE,
  count = 1,
  count.only.increase = TRUE
)
```

Arguments

`population`	Population list
`phenos`	Matrix of phenotypes to enter (one row per individual with 1 element coding individual name)
`na.override`	Set to TRUE to also enter NA values (Default: FALSE - those entries will be skipped)
`count`	Counting for economic cost calculation (default: 1 - (one observation (for "pheno"), one genotyping (for "bve")))
`count.only.increase`	Set to FALSE to reduce the number of observation for a phenotype to "count" (default: TRUE)

Value

Population-List with newly entered phenotypes

Examples

```
data(ex_pop)
bv <- get.bv(ex_pop, gen=2, use.id = FALSE)
new.bve <- cbind( colnames(bv), bv[,1]) ## Unrealistic but you do not get better than this!
ex_pop <- insert.pheno(ex_pop, phenos=new.bve)
```

insert.pheno.single *Manually enter phenotypes*

Description

Function to manually enter phenotypes (with repeated records)

Usage

```
insert.pheno.single(population, phenos, count.only.increase = TRUE)
```

Arguments

population	Population list
phenos	Matrix of breeding values to enter (one row per individual: 1st column: individual, 2nd column: trait nr, 3rd-nth column records)
count.only.increase	Set to FALSE to reduce the number of observation for a phenotype to "count" (default: TRUE)

Value

Population-List with newly entered phenotypes

Examples

```
data(ex_pop)
bv <- get.bv(ex_pop, gen=2, use.id = FALSE)
new.bve <- cbind( colnames(bv), 1, bv[,1]) ## Unrealistic but you do not get better than this!
ex_pop <- insert.pheno.single(ex_pop, phenos=new.bve)
```

json.simulation *Simulation of a breeding program based on a JSON-file from MoBPSweb*

Description

Function to simulate a breeding program based on a JSON-file from MoBPSweb

Usage

```
json.simulation(
  file = NULL,
  log = NULL,
  total = NULL,
  fast.mode = FALSE,
  progress.bars = FALSE,
  size.scaling = NULL,
  rep.max = 1,
  verbose = TRUE,
  miraculix.cores = NULL,
  miraculix.chol = NULL,
  skip.population = FALSE,
  time.check = FALSE,
  time.max = 7200,
  export.population = FALSE,
  export.gen = NULL,
  export.timepoint = NULL,
  export.cor = FALSE,
  fixed.generation.order = NULL,
  generation.cores = NULL,
  manual.select.check = FALSE
)
```

Arguments

file	Path to a json-file generated by the user-interface
log	Provide Path where to write a log-file of your simulation (or false to not write a log-file)
total	Json-file imported via jsonlite::read_json
fast.mode	Set to TRUE work on a small genome with few markers
progress.bars	Set to TRUE to display progress bars
size.scaling	Scale the size of nodes by this factor (especially for testing smaller examples)
rep.max	Maximum number of repeats to use in fast.mode
verbose	Set to FALSE to not display any prints
miraculix.cores	Number of cores used in miraculix applications (default: 1)
miraculix.chol	Set to FALSE to manually deactive the use of miraculix for any cholesky decompostion even though miraculix is active
skip.population	Set to TRUE to not execute breeding actions (only cost/time estimation will be performed)
time.check	Set to TRUE to automatically check simulation run-time before executing breeding actions
time.max	Maximum length of the simulation in seconds when time.check is active

`export.population`
: Path were to export the population to (at state selected in export.gen/timepoint)

`export.gen`
: Last generation to simulate before exporting population to file

`export.timepoint`
: Last timepoint to simulate before exporting population to file

`export.cor`
: Set TRUE to export correlation matrices

`fixed.generation.order`
: Vector containing the order of cohorts to generate (Advanced // Testing Parameter!)

`generation.cores`
: Number of cores used for the generation of new individuals (This will only be active when generating more than 500 individuals)

`manual.select.check`
: Set to FALSE to not automatically remove cohorts from Manual select with they lead to an invalid breeding scheme

Value

Population-list

Examples

```
data(ex_json)
population <- json.simulation(total=ex_json)
```

---

kinship.development *Development of genetic/breeding value*

Description

Function to plot genetic/breeding values for multiple generation/cohorts

Usage

```
kinship.development(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  json = FALSE,
  ibd.obs = 50,
  hbd.obs = 10,
  display.cohort.name = FALSE,
  display.time.point = FALSE,
  equal.spacing = FALSE,
  time_reorder = FALSE,
  display.hbd = FALSE
)
```

Arguments

population	population list
database	Groups of individuals to consider for the export
gen	Quick-insert for database (vector of all generations to export)
cohorts	Quick-insert for database (vector of names of cohorts to export)
json	If TRUE extract which cohorts to plot according to the json-file used in json.simulation
ibd.obs	Number of Individual pairs to sample for IBD estimation
hbd.obs	Number of Individuals to sample for HBD estimation
display.cohort.name	Set TRUE to display the name of the cohort in the x-axis
display.time.point	Set TRUE to use time point of generated to sort groups
equal.spacing	Equal distance between groups (independent of time.point)
time_reorder	Set TRUE to order cohorts according to the time point of generation
display.hbd	Set to TRUE to also display HBD in plot

Value

Estimated of avg. kinship/inbreeding based on IBD/HBD

Examples

```
data(ex_pop)
kinship.development(ex_pop,gen=1:5)
```

---

kinship.emp *Empirical kinship*

Description

Function to compute empirical kinship for a set of individuals

Usage

```
kinship.emp(
  population = NULL,
  animals = NULL,
  gen = NULL,
  database = NULL,
  cohorts = NULL,
  sym = FALSE,
  use.id = TRUE
)
```

Arguments

population	Population list
animals	List of animals to compute kinship for
gen	Quick-insert for database (vector of all generations to export)
database	Groups of individuals to consider for the export
cohorts	Quick-insert for database (vector of names of cohorts to export)
sym	If True derive matrix entries below principle-diagonal
use.id	Set TRUE to use animal IDs for column/row-names in the output matrix (default: TRUE)

Value

Empirical kinship matrix (IBD-based since Founders)

Examples

```
data(ex_pop)
kinship <- kinship.emp(population=ex_pop, database=cbind(2,1,1,25))
```

---

kinship.emp.fast *Approximate empirical kinship*

Description

Function to compute empirical kinship for a set of individuals (not all pairs of individuals are evaluated)

Usage

```
kinship.emp.fast(
  population = NULL,
  animals = NULL,
  gen = NULL,
  database = NULL,
  cohorts = NULL,
  ibd.obs = 200,
  hbd.obs = 50
)
```

Arguments

population	Population list
animals	List of animals to compute kinship for
gen	Quick-insert for database (vector of all generations to export)
database	Groups of individuals to consider for the export
cohorts	Quick-insert for database (vector of names of cohorts to export)
ibd.obs	Number of Individual pairs to sample for IBD estimation
hbd.obs	Number of Individuals to sample for HBD estimation

Value

Empirical kinship matrix (IBD-based since Founders) per gen/database/cohort

Examples

```
data(ex_pop)
kinship.emp.fast(population=ex_pop,gen=2)
```

---

kinship.emp.fast.between

Approximate empirical kinship

Description

Function to compute empirical kinship for a set of individuals (not all pairs of individuals are evaluated)

Usage

```
kinship.emp.fast.between(
  population = NULL,
  gen1 = NULL,
  database1 = NULL,
  cohorts1 = NULL,
  gen2 = NULL,
  database2 = NULL,
  cohorts2 = NULL,
  ibd.obs = 50
)
```

Arguments

population	Population list
gen1	Quick-insert for database1 (vector of all generations to export)
database1	First Groups of individuals to consider for the export
cohorts1	Quick-insert for database1 (vector of names of cohorts to export)
gen2	Quick-insert for database2 (vector of all generations to export)
database2	Second Groups of individuals to consider for the export
cohorts2	Quick-insert for database2 (vector of names of cohorts to export)
ibd.obs	Number of Individual pairs to sample for IBD estimation

Value

Empirical kinship matrix (IBD-based since Founders) per gen/database/cohort

Examples

```
data(ex_pop)
kinship.emp.fast(population=ex_pop,gen=2)
```

---

kinship.exp *Derive expected kinship*

Description

Function to derive expected kinship

Usage

```
kinship.exp(
  population,
  gen = NULL,
  database = NULL,
  cohorts = NULL,
  depth.pedigree = 7,
  start.kinship = NULL,
  elements = NULL,
  mult = 1,
  storage.save = 1.05,
  verbose = TRUE,
  include.error = TRUE,
  elements.copy = FALSE
)
```

Arguments

population	Population list
gen	Quick-insert for database (vector of all generations to export)
database	Groups of individuals to consider for the export
cohorts	Quick-insert for database (vector of names of cohorts to export)
depth.pedigree	Depth of the pedigree in generations
start.kinship	Relationship matrix of the individuals in the first considered generation
elements	Vector of individuals from the database to include in pedigree matrix
mult	Multiplicator of kinship matrix (default: 1; set to 2 for a pedigree relationship matrix)
storage.save	Lower numbers will lead to less memory but slightly higher computing time (default: 1.5, min: 1)
verbose	Set to FALSE to not display any prints
include.error	Set to FALSE to ignore/correct any errors in the pedigree
elements.copy	Set to TRUE to automatically remove duplicated entries for individuals/database from the output matrix (default: FALSE)

Value

Pedigree-based kinship matrix for in gen/database/cohort selected individuals

Examples

```
data(ex_pop)
kinship <- kinship.exp(population=ex_pop, gen=2)
```

---

ld.decay *Generate LD plot*

Description

Generate LD pot

Usage

```
ld.decay(
  population,
  genotype.dataset = NULL,
  chromosome = 1,
  dist = NULL,
  step = 5,
  max = 500,
  max.cases = 100,
  database = NULL,
```

```
  gen = NULL,
  cohorts = NULL,
  type = "snp",
  plot = FALSE,
  xlim = NULL,
  ylim = NULL
)
```

Arguments

population	Population list
genotype.dataset	Genotype dataset (default: NULL - just to save computation time when get.geno was already run)
chromosome	Only consider a specific chromosome in calculations (default: 1)
dist	Manuel input of marker distances to analyze
step	Stepsize to calculate LD
max	Maximum distance between markers to consider for LD-plot
max.cases	Maximum number of marker pairs to consider of each distance (default: 100; randomly sampled!)
database	Groups of individuals to consider for the export
gen	Quick-insert for database (vector of all generations to export)
cohorts	Quick-insert for database (vector of names of cohorts to export)
type	Compute LD decay according to following distance measure between markers (default: "snp", alt: "bp", "cM")
plot	Set to FALSE to not generate an LD plot
xlim	Axis limits for the x-axis in the LD plot
ylim	Axis limits for the y-axis in the LD plot

Value

LD-decay plot for in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
ld.decay(population=ex_pop, gen=5)
```

maize_chip *maize chip*

Description

Genome for maize according to Lee et al.

Usage

```
maize_chip
```

Author(s)

Torsten Pook <torsten.pook@wur.nl>

Source

Lee et al 2002

matrix.posdef *Projection positive definite*

Description

Function to project a matrix in the space of positive definite matrices

Usage

```
matrix.posdef(A, verbose = TRUE, matrix.name = "Matrix", epsilon = 1e-07)
```

Arguments

A	Input matrix to project
verbose	Set to FALSE to not display any prints
matrix.name	This is just for internal prints
epsilon	This factor is added to the diagonal to avoid numerical issues with semi-definit matrices

Value

Positive definite matrix

merging.cohorts *Merging of cohorts*

Description

Function to merge cohorts in a population list

Usage

```
merging.cohorts(population, cohorts, name.cohort = NULL)
```

Arguments

population	Population list
cohorts	Quick-insert for database (vector of names of cohorts to export)
name.cohort	Name of the newly added cohort

Examples

```
data(ex_pop)
population <- breeding.diploid(ex_pop, add.gen = 7, breeding.size = 50)
population <- breeding.diploid(population, add.gen = 7, breeding.size = 50,
 selection.m.database = cbind(6,1), selection.f.database = cbind(6,2))
population <- merging.cohorts(population, cohorts = c("Cohort_7_M", "Cohort_8_M"))
```

merging.trait *Generation of genomic traits*

Description

Generation of the trait in a starting population

Usage

```
merging.trait(
  population,
  merge = NULL,
  trait.name = NULL,
  bv.standard = FALSE,
  mean.target = NULL,
  var.target = NULL,
  verbose = TRUE,
  set.zero = FALSE,
  new.phenotype.correlation = NULL,
  new.residual.correlation = NULL
)
```

Arguments

population	Population list
merge	Vector containing the traits to merge (e.g. c(2,3))
trait.name	Name of the trait generated
bv.standard	Set TRUE to standardize trait mean and variance via bv.standardization()
mean.target	Target mean
var.target	Target variance
verbose	Set to FALSE to not display any prints
set.zero	Set to TRUE to have no effect on the 0 genotype (or 00 for QTLs with 2 underlying SNPs)
new.phenotype.correlation	(OLD! - use new.residual.correlation) Correlation of the simulated enviromental variance
new.residual.correlation	Correlation of the simulated enviromental variance

Value

Population-list with one or more additional new traits

Examples

```
population <- creating.diploid(nsnp=1000, nindi=100)
population <- creating.trait(population, n.additive=c(100,100))
population <- merging.trait(population, merge = c(1,2))
```

---

miesenberger.index *Miesenberger Index*

Description

Function to selection index weights according to Miesenberger 1997

Usage

```
miesenberger.index(V, G, V1 = NULL, RG = NULL, r, w, zw = NULL)
```

Arguments

V	Phenotypic covariance matrix
G	Genomic covariance matrix
V1	Inverted phenotypic covariance matrix
RG	Genomic correlation matrix
r	reliability for the breeding value estimation
w	relative weighting of each trait (per genetic SD)
zw	Estimated breeding value

Value

weights of the selection index

`miraculix` *Add miraculix-coding for genotypes*

Description

Internal function to store genotypes bit-wise

Usage

```
miraculix(population)
```

Arguments

`population`	Population list

Value

Population list

Examples

```
# This is only relevant with the package miraculix is installed and used
population <- creating.diploid(nsnp=100, nindi=50, miraculix=FALSE)
population <- miraculix(population)
```

`mutation.intro` *Mutation intro*

Description

Function to change the base-pair in a specific loci

Usage

```
mutation.intro(
  population,
  gen = NULL,
  database = NULL,
  cohorts = NULL,
  qtl.posi,
  target.variant = NULL,
  haplo.set = 1
)
```

Arguments

population	Population list
gen	Quick-insert for database (vector of all generations to export)
database	Groups of individuals to consider for the export
cohorts	Quick-insert for database (vector of names of cohorts to export)
qtl.posi	Marker number to mutate
target.variant	target variant to obtain ((if haplotype already is correct do not introduce a mutation))
haplo.set	Select chromosome set (default: 1 , alt: 2, 1:2 (to edit both))

Value

Population-List with mutated marker for the selected individual

Examples

```
data(ex_pop)
ex_pop <- mutation.intro(ex_pop, database = cbind(1,1,1), qtl.posi=100)
```

---

new.base.generation *Set new base generation*

Description

Function to set a new base generation for the population

Usage

```
new.base.generation(
  population,
  base.gen = NULL,
  base.database = NULL,
  base.cohorts = NULL,
  delete.previous.gen = FALSE,
  delete.breeding.totals = FALSE,
  delete.bve.data = FALSE,
  add.chromosome.ends = TRUE,
  founder.pool = 1
)
```

Arguments

population	Population list
base.gen	Vector containing all new base generations
base.database	Matrix containing all database entries to be used as new base generation
base.cohorts	Vector containing all cohorts to be used as new base generations
delete.previous.gen	Delete all data before base.gen (default: FALSE)
delete.breeding.totals	Delete all breeding totals before base.gen (default: FALSE)
delete.bve.data	Delete all previous bve data (default: FALSE)
add.chromosome.ends	Add chromosome ends as recombination points
founder.pool	AAA

Value

Population-List with mutated marker for the selected individual

Examples

```
data(ex_pop)
ex_pop <- new.base.generation(ex_pop, base.gen=2)
```

OGC *Optimal genetic contribution*

Description

In this function the OGC selection according to Meuwissen 1997 is performed

Usage

```
OGC(
  A,
  u,
  Q,
  cAc = NA,
  single = TRUE,
  verbose = FALSE,
  max_male = Inf,
  max_female = Inf
)
```

Arguments

A	relationship matrix
u	breeding values
Q	sex indicator
cAc	target gain in inbreeding
single	If FALSE multiple individuals can be removed at the same type (this is faster but potentially inaccurate!)
verbose	Set to FALSE to not display any prints
max_male	maximum number of male with positive contributions
max_female	maximum number of females with positive contributions

Value

[[1]] Contributions [[2]] expected inbreeding gain

---

ogc.mobps *Breeding function*

Description

Function to simulate a step in a breeding scheme

Usage

```
ogc.mobps(
  population,
  animallist,
  relationship.matrix.ogc,
  depth.pedigree.ogc,
  bve.pedigree.error,
  ogc.target = "min.sKin",
  ogc.uniform = NULL,
  ogc.lb = NULL,
  ogc.ub = NULL,
  ogc.ub.sKin = NULL,
  ogc.lb.BV = NULL,
  ogc.ub.BV = NULL,
  ogc.eq.BV = NULL,
  ogc.ub.sKin.increase = NULL,
  ogc.lb.BV.increase = NULL,
  bve.p_i.list = NULL,
  miraculix = FALSE,
  miraculix.mult = FALSE,
  import.position.calculation = NULL,
```

```
  decodeOriginsU = decodeOriginsR,
  nbits = NULL,
  store.sparse = FALSE,
  verbose = TRUE,
  bit.storing = FALSE
)
```

Arguments

population	Population list
animallist	List of individuals to include in ogc
relationship.matrix.ogc	Method to calculate relationship matrix for OGC (Default: "pedigree", alt: "vanRaden", "CE", "non_stand", "CE2", "CM")
depth.pedigree.ogc	Depth of the pedigree in generations (default: 7)
bve.pedigree.error	Set to FALSE to ignore/correct for any pedigree errors
ogc.target	Target of OGC (default: "min.sKin" - minimize inbreeding; alt: "max.BV" / "min.BV" - maximize genetic gain; both under constrains selected below)
ogc.uniform	This corresponds to the uniform constrain in optiSel
ogc.lb	This corresponds to the lb constrain in optiSel
ogc.ub	This corresponds to the ub constrain in optiSel
ogc.ub.sKin	This corresponds to the ub.sKin constrain in optiSel
ogc.lb.BV	This corresponds to the lb.BV constrain in optiSel
ogc.ub.BV	This corresponds to the ub.BV constrain in optiSel
ogc.eq.BV	This corresponds to the eq.BV constrain in optiSel
ogc.ub.sKin.increase	This corresponds to the upper bound (current sKin + ogc.ub.sKin.increase) as ub.sKin in optiSel
ogc.lb.BV.increase	This corresponds to the lower bound (current BV + ogc.lb.BV.increase) as lb.BV in optiSel
bve.p_i.list	XXX
miraculix	XXX
miraculix.mult	XXX
import.position.calculation	XXX
decodeOriginsU	XXX
nbits	XXX
store.sparse	XXX
verbose	Set to FALSE to not display any prints
bit.storing	Set to TRUE if the MoBPS (not-miraculix! bit-storing is used)

Value

contributions of each individual in selection

Examples

```
population <- creating.diploid(nsnp=1000, nindi=100, n.additive = 10)
population <- breeding.diploid(population, breeding.size=100, selection.size=c(25,25),
ogc = TRUE)
```

`optimize.cores` *Optimize generation cores*

Description

Function to optimize the number of cores used in the generation of new individuals

Usage

```
optimize.cores(
  population = NULL,
  test.size = 2500,
  max.cores = 10,
  verbose = TRUE,
  plot = TRUE
)
```

Arguments

`population`	Population list
`test.size`	Number of individuals to generate for each core number (default: 2500)
`max.cores`	Maximum number of cores to test (default: 10)
`verbose`	Set to FALSE to not display any prints
`plot`	Set to FALSE to not generate a plot of computing times per core

Value

Population-list with one or more additional new traits

Examples

```
population = optimize.cores(max.cores=1, test.size=500)
```

`pedigree.matrix` *Derive pedigree relationship matrix*

Description

Function to derive pedigree matrix

Usage

```
pedigree.matrix(
  population,
  gen = NULL,
  database = NULL,
  cohorts = NULL,
  depth.pedigree = 7,
  start.kinship = NULL,
  elements = NULL,
  storage.save = 1.05,
  verbose = TRUE
)
```

Arguments

`population`	Population list
`gen`	Quick-insert for database (vector of all generations to export)
`database`	Groups of individuals to consider for the export
`cohorts`	Quick-insert for database (vector of names of cohorts to export)
`depth.pedigree`	Depth of the pedigree in generations
`start.kinship`	Relationship matrix of the individuals in the first considered generation
`elements`	Vector of individuals from the database to include in pedigree matrix
`storage.save`	Lower numbers will lead to less memory but slightly higher computing time (default: 1.5, min: 1)
`verbose`	Set to FALSE to not display any prints

Value

Pedigree-based kinship matrix for in gen/database/cohort selected individuals

Examples

```
data(ex_pop)
pedigree_matrix <- pedigree.matrix(population=ex_pop, gen=2)
```

`pedigree.simulation` *Simulation of a given pedigree*

Description

Function to simulate a given pedigree

Usage

```
pedigree.simulation(
  pedigree,
  keep.ids = FALSE,
  plot = TRUE,
  dataset = NULL,
  vcf = NULL,
  chr.nr = NULL,
  bp = NULL,
  snp.name = NULL,
  hom0 = NULL,
  hom1 = NULL,
  bpcm.conversion = 0,
  nsnp = 0,
  freq = "beta",
  sex.s = "fixed",
  chromosome.length = NULL,
  length.before = 5,
  length.behind = 5,
  real.bv.add = NULL,
  real.bv.mult = NULL,
  real.bv.dice = NULL,
  snps.equidistant = NULL,
  bv.total = 0,
  polygenic.variance = 100,
  bve.mult.factor = NULL,
  bve.poly.factor = NULL,
  base.bv = NULL,
  add.chromosome.ends = TRUE,
  new.phenotype.correlation = NULL,
  new.residual.correlation = NULL,
  new.breeding.correlation = NULL,
  add.architecture = NULL,
  snp.position = NULL,
  position.scaling = FALSE,
  bit.storing = FALSE,
  nbits = 30,
  randomSeed = NULL,
  miraculix = TRUE,
```

```
  miraculix.dataset = TRUE,
  n.additive = 0,
  n.dominant = 0,
  n.qualitative = 0,
  n.quantitative = 0,
  var.additive.l = NULL,
  var.dominant.l = NULL,
  var.qualitative.l = NULL,
  var.quantitative.l = NULL,
  exclude.snps = NULL,
  replace.real.bv = FALSE,
  shuffle.traits = NULL,
  shuffle.cor = NULL,
  skip.rest = FALSE,
  enter.bv = TRUE,
  name.cohort = NULL,
  template.chip = NULL,
  beta.shape1 = 1,
  beta.shape2 = 1,
  time.point = 0,
  creating.type = 0,
  trait.name = NULL,
  trait.cor = NULL,
  trait.cor.include = NULL,
  share.genotyped = 1,
  genotyped.s = NULL,
  map = NULL,
  remove.invalid.qtl = TRUE,
  verbose = TRUE,
  bv.standard = FALSE,
  mean.target = NULL,
  var.target = NULL,
  progress.bar = TRUE,
  is.maternal = NULL,
  is.paternal = NULL,
  vcf.maxsnp = Inf,
  halffounder = TRUE,
  output.extended = FALSE
)
```

Arguments

pedigree	Pedigree-file (matrix with 3 columns (Individual ID, Father ID, Mother ID), optional forth columns with earliest generations to generate an individual)
keep.ids	Set to TRUE to keep the IDs from the pedigree-file instead of the default MoBPS ids
plot	Set to FALSE to not generate an overview of inbreeding and number of individuals over time

`dataset` SNP dataset, use "random", "allhetero" "all0" when generating a dataset via nsnp,nindi

`vcf` Path to a vcf-file used as input genotypes (correct haplotype phase is assumed!)

`chr.nr` Vector containing the associated chromosome for each marker (default: all on the same)

`bp` Vector containing the physical position (bp) for each marker (default: 1,2,3...)

`snp.name` Vector containing the name of each marker (default ChrXSNPY - XY chosen accordingly)

`hom0` Vector containing the first allelic variant in each marker (default: 0)

`hom1` Vector containing the second allelic variant in each marker (default: 1)

`bpcm.conversion`
Convert physical position (bp) into a cM position (default: 0 - not done)

`nsnp` number of markers to generate in a random dataset

`freq` frequency of allele 1 when randomly generating a dataset

`sex.s` Specify which newly added individuals are male (1) or female (2)

`chromosome.length`
Length of the newly added chromosome (default: 5)

`length.before` Length before the first SNP of the dataset (default: 5)

`length.behind` Length after the last SNP of the dataset (default: 5)

`real.bv.add` Single Marker effects

`real.bv.mult` Two Marker effects

`real.bv.dice` Multi-marker effects

`snps.equidistant`
Use equidistant markers (computationally faster! ; default: TRUE)

`bv.total` Number of traits (If more than traits via real.bv.X use traits with no directly underlying QTL)

`polygenic.variance`
Genetic variance of traits with no underlying QTL

`bve.mult.factor`
Multiplicate trait value times this

`bve.poly.factor`
Potency trait value over this

`base.bv` Average genetic value of a trait

`add.chromosome.ends`
Add chromosome ends as recombination points

`new.phenotype.correlation`
(OLD! - use new.residual.correlation) Correlation of the simulated enviromental variance

`new.residual.correlation`
Correlation of the simulated enviromental variance

`new.breeding.correlation`
Correlation of the simulated genetic variance (child share! heritage is not influenced!

`add.architecture`
: Add genetic architecture (marker positions)

`snp.position`
: Location of each marker on the genetic map

`position.scaling`
: Manual scaling of snp.position

`bit.storing`
: Set to TRUE if the MoBPS (not-miraculix! bit-storing is used)

`nbits`
: Bits available in MoBPS-bit-storing

`randomSeed`
: Set random seed of the process

`miraculix`
: If TRUE use miraculix package for data storage, computations and dataset generation

`miraculix.dataset`
: Set FALSE to deactive miraculix package for dataset generation

`n.additive`
: Number of additive QTL

`n.dominant`
: Number of dominante QTL

`n.qualitative`
: Number of qualitative epistatic QTL

`n.quantitative`
: Number of quantitative epistatic QTL

`var.additive.l`
: Variance of additive QTL

`var.dominant.l`
: Variance of dominante QTL

`var.qualitative.l`
: Variance of qualitative epistatic QTL

`var.quantitative.l`
: Variance of quantitative epistatic QTL

`exclude.snps`
: Marker were no QTL are simulated on

`replace.real.bv`
: If TRUE delete the simulated traits added before

`shuffle.traits`
: Combine different traits into a joined trait

`shuffle.cor`
: Target Correlation between shuffled traits

`skip.rest`
: Internal variable needed when adding multipe chromosomes jointly

`enter.bv`
: Internal parameter

`name.cohort`
: Name of the newly added cohort

`template.chip`
: Import genetic map and chip from a species ("cattle", "chicken", "pig")

`beta.shape1`
: First parameter of the beta distribution for simulating allele frequencies

`beta.shape2`
: Second parameter of the beta distribution for simulating allele frequencies

`time.point`
: Time point at which the new individuals are generated

`creating.type`
: Technique to generate new individuals (usage in web-based application)

`trait.name`
: Name of the trait generated

`trait.cor`
: Target correlation between QTL-based traits (underlying true genomic values)

`trait.cor.include`
: Vector of traits to be included in the modelling of corrlated traits (default: all - needs to match with trait.cor)

share.genotyped
: Share of individuals genotyped in the founders

genotyped.s
: Specify with newly added individuals are genotyped (1) or not (0)

map
: map-file that contains up to 5 colums (Chromsome, SNP-id, M-position, Bp-position, allele freq - Everything not provides it set to NA). A map can be imported via MoBPSmaps::ensembl.map()

remove.invalid.qtl
: Set to FALSE to deactive the automatic removal of QTLs on markers that do not exist

verbose
: Set to FALSE to not display any prints

bv.standard
: Set TRUE to standardize trait mean and variance via bv.standardization() - automatically set to TRUE when mean/var.target are used

mean.target
: Target mean

var.target
: Target variance

progress.bar
: Set to FALSE to not use progress bars in the simulation (Keep log-files lean!)

is.maternal
: Vector coding if a trait is caused by a maternal effect (Default: all FALSE)

is.paternal
: Vector coding if a trait is caused by a paternal effect (Default: all FALSE)

vcf.maxsnp
: Maximum number of SNPs to include in the genotype file (default: Inf)

halffounder
: Set to FALSE to make individuals with only one known parent founders (default: TRUE; will additional generate a missing unrelated parent)

output.extended
: Set TRUE to export more information about the population list

add.chromosome
: If TRUE add an additional chromosome to the dataset

Value

Population-list

Examples

```
pedigree <- matrix(c(1,0,0,
2,0,0,
3,0,0,
4,1,2,
5,1,3,
6,1,3,
7,1,3,
8,4,6,
9,4,7), ncol=3, byrow=TRUE)
population = pedigree.simulation(pedigree, nsnp = 100)
```

pedmap.to.phasedbeaglevcf *Internal function to perform imputing/phasing*

Description

Internal function to perform imputing/phasing (path chosen for the web-based application)

Usage

```
pedmap.to.phasedbeaglevcf(
  ped_path = NULL,
  map_path = NULL,
  vcf_path = NULL,
  beagle_jar = "/home/nha/beagle.03Jul18.40b.jar",
  plink_dir = "/home/nha/Plink/plink",
  db_dir = "/home/nha/Plink/DB/",
  verbose = TRUE
)
```

Arguments

ped_path	Directory of the ped-file
map_path	Directory of the map-file
vcf_path	Directory of the vcf-file (this will override any ped/map-file input)
beagle_jar	Directory of BEAGLE
plink_dir	Directory of Plink
db_dir	Directory to save newly generated files (ped/map will be stored in the original folder)
verbose	Set to FALSE to not display any prints

Value

Phased vcf file in vcf_path

pig_chip *pig chip*

Description

Genome for pig according to Rohrer et al.

Usage

```
pig_chip
```

Author(s)

Torsten Pook <torsten.pook@wur.nl>

Source

Rohrer et al 1994

---

plot.population *Plot Population*

Description

Basic plot of the population list

Usage

```
## S3 method for class 'population'
plot(x, type = "bve", gen = NULL, database = NULL, cohorts = NULL, ...)
```

Arguments

x	Population-list
type	Default "bve" - bv.development, alt: "kinship" - kinship.development(), "pca" - get.pca()
gen	generations to consider
database	groups to consider
cohorts	cohorts to consider
...	remaining stuff

Value

Summary of the population list including number of individuals, genome length and trait overview

Examples

```
data(ex_pop)
plot(ex_pop)
```

`recalculate.bv` *Recalculate genomic values*

Description

Function to recalculate genomic values

Usage

```
recalculate.bv(
  population,
  gen = NULL,
  database = NULL,
  cohorts = NULL,
  bv.ignore.traits = NULL,
  store.comp.times = TRUE
)
```

Arguments

`population`	Population list
`gen`	Quick-insert for database (vector of all generations to export)
`database`	Groups of individuals to consider for the export
`cohorts`	Quick-insert for database (vector of names of cohorts to export)
`bv.ignore.traits`	Vector of traits to ignore in the calculation of the genomic value (default: NULL; Only recommended for high number of traits and experienced users!)
`store.comp.times`	If TRUE store computation times in $info$comp.times.general (default: TRUE)

Value

Population list

Examples

```
data(ex_pop)
population <- recalculate.bv(ex_pop, gen=2)
```

`recalculate.manual` *Recalculate genomic values*

Description

Function to recalculate genomic values

Usage

```
recalculate.manual(
  population,
  gen = NULL,
  database = NULL,
  cohorts = NULL,
  e0 = NULL,
  e1 = NULL,
  e2 = NULL,
  store.comp.times = TRUE
)
```

Arguments

`population`	Population list
`gen`	Quick-insert for database (vector of all generations to export)
`database`	Groups of individuals to consider for the export
`cohorts`	Quick-insert for database (vector of names of cohorts to export)
`e0`	Effect matrix for 0 genotype (default: Will be automatically extracted)
`e1`	Effect matrix for 1 genotype (default: Will be automatically extracted)
`e2`	Effect matrix for 2 genotype (default: Will be automatically extracted)
`store.comp.times`	If TRUE store computation times in $info$comp.times.general (default: TRUE)

Value

Population list

Examples

```
data(ex_pop)
population <- recalculate.manual(ex_pop, gen = 1)
```

recombination.function.haldane *Haldane recombination function*

Description

Standard haldane function

Usage

```
recombination.function.haldane(noc, length.genome)
```

Arguments

noc	number of recombination points
length.genome	Length of the genome

Value

Vector of recombination events

Examples

```
recombination.function.haldane(5, 10)
```

renaming.cohort *Rename a cohort*

Description

Function to rename a cohort

Usage

```
renaming.cohort(population, old.name, new.name, verbose = TRUE)
```

Arguments

population	Population list
old.name	Quick-insert for database (vector of names of cohorts to export)
new.name	Groups of individuals to remove from the database (same IDs!)
verbose	Set to FALSE to not display any prints

Value

population list

Examples

```
data(ex_pop)
population <- renaming.cohort(ex_pop, old.name="Cohort_4", new.name = "NewName")
```

`rowMedian` *Row-wise Median*

Description

Function to calculate row-wise median values

Usage

```
rowMedian(A)
```

Arguments

A	Matrix

Value

row-wise median values of a matrix

Matrix with modified diagonal entries

Examples

```
A <- matrix(c(1,2,3,4), ncol=2)
x <- rowMedian(A)
```

`scaling.relationship` *scaling.relationship*

Description

Function to scale relationship matricies

Usage

```
scaling.relationship(A, Z, p)
```

Arguments

A	Population list
Z	genotype matrix
p	allele frequency

Value

scaled genomic relationship matrix

`set.age.point` *Set age points*

Description

Function to overwrite age.points of individuals

Usage

```
set.age.point(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  time.point = 0
)
```

Arguments

`population`	Population list
`database`	Groups of individuals to consider for the export
`gen`	Quick-insert for database (vector of all generations to export)
`cohorts`	Quick-insert for database (vector of names of cohorts to export)
`time.point`	Input value for the age point (time of birth) of an individual (default: 0)

Value

Population-List with newly entered class values

Examples

```
data(ex_pop)
population <- set.age.point(ex_pop, database=cbind(1,1), time.point = 2)
```

set.class *Set class*

Description

Function to overwrite class values for individuals

Usage

```
set.class(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  new.class = 0
)
```

Arguments

population	Population list
database	Groups of individuals to consider for the export
gen	Quick-insert for database (vector of all generations to export)
cohorts	Quick-insert for database (vector of names of cohorts to export)
new.class	Class to change to (either single character or vector for each individual when just a single group is selected)

Value

Population-List with newly entered class values

Examples

```
data(ex_pop)
population <- set.class(ex_pop, database=cbind(1,1), new.class = 2)
```

set.default *Set defaults*

Description

Set default parameter values in breeding.diploid

Usage

```
set.default(
  population,
  parameter.name = NULL,
  parameter.value = NULL,
  parameter.remove = NULL,
  reset.all = FALSE
)
```

Arguments

population	Population list
parameter.name	Number of traits (If more than traits via real.bv.X use traits with no directly underlying QTL)
parameter.value	Genetic variance of traits with no underlying QTL
parameter.remove	Remove a specific previously generated parameter default
reset.all	Set to TRUE to remove all prior parameter values

Value

Population-list with one or more additional new traits

Examples

```
data(ex_pop)
population <- set.default(ex_pop, parameter.name="heritability", parameter.value=0.3)
```

---

set.mean.pool *Set differences between founder pool*

Description

Function to scale trait for genetic differences based on founder pools

Usage

```
set.mean.pool(
  population,
  pool = NULL,
  mean = NULL,
  trait = NULL,
  gen = NULL,
  database = NULL,
  cohorts = NULL,
```

```
  reference = "pool",
  max.effects = Inf
)
```

Arguments

population	Population list
pool	Vector with pools considered (default: 1:length(mean))
mean	Vector with the target mean for the different pools
trait	Which trait to set the new mean for
gen	Quick-insert for database (vector of all generations to export) (THIS CAN ONLY BE APPLIED ON FOUNDERS, if empty -> default: 1)
database	Groups of individuals to consider for the export (THIS CAN ONLY BE APPLIED ON FOUNDERS)
cohorts	Quick-insert for database (vector of names of cohorts to export) (THIS CAN ONLY BE APPLIED ON FOUNDERS)
reference	Target mean is compared again the reference (default: "pool" - average genomic value in the respective pool, alt: "all")
max.effects	Maximum number of locations in the genome that will be assigned an effect for pool-based correction

Value

Class of in gen/database/cohorts selected individuals

Examples

```
population = creating.diploid(nsnp = 100, nindi = 10, n.additive = 100, founder.pool = 1)
population = creating.diploid(population=population, nindi = 10,
  founder.pool = 2)
population = set.mean.pool(population, mean = c(100,110))
```

---

set.time.point *Set time point*

Description

Function to set time point of birth for individuals

Usage

```
set.time.point(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  time.point = 0
)
```

Arguments

population	Population list
database	Groups of individuals to consider for the export
gen	Quick-insert for database (vector of all generations to export)
cohorts	Quick-insert for database (vector of names of cohorts to export)
time.point	Input value for the time point (time the cohort of the individual was generated) (default: 0)

Value

Population-List with newly entered time points

Examples

```
data(ex_pop)
population <- set.time.point(ex_pop, database=cbind(1,1), time.point = 2)
```

---

sheep_chip *sheep chip*

Description

Genome for sheep according to Prieur et al.

Usage

```
sheep_chip
```

Author(s)

Torsten Pook <torsten.pook@wur.nl>

Source

Prieur et al 2017

sortd *Apply sort and unique*

Description

Efficient function to perform sort(unique(v))

Usage

```
sortd(v)
```

Arguments

v	Vector

Value

numerical sorted vector without duplicates

Examples

```
v <- c(1,1,4,5)
sortd(v)
```

ssGBLUP *Single Step GBLUP*

Description

Function to perform single step GBLUP according to Legarra 2014

Usage

```
ssGBLUP(A11, A12, A22, G)
```

Arguments

A11	pedigree relationship matrix of non-genotyped individuals
A12	pedigree relationship matrix between non-genotyped and genotyped individuals
A22	pedigree relationship matrix of genotyped individuals
G	genomic relationship matrix of genotyped individuals

Value

Single step relationship matrix

summary.population *Summary Population*

Description

Summary of the population list

Usage

```
## S3 method for class 'population'
summary(object, ...)
```

Arguments

object	Population-list
...	additional arguments affecting the summary produced

Value

Summary of the population list including number of individuals, genome length and trait overview

Examples

```
data(ex_pop)
summary(ex_pop)
```

vlist *Generation of a sublist*

Description

Internal function to write a couple of list entries in a new list

Usage

```
vlist(list, skip = NULL, first = NULL, select = NULL)
```

Arguments

list	list you want to print details of
skip	Skip first that many list-elements
first	Only display first that many list-elements
select	Display only selected list-elements

Value

Selected elements of a list

Examples

```
data(ex_pop)
vlist(ex_pop$breeding[[1]], select=3:10)
```

`write.pedigree` *write.pedigree.mixblup*

Description

write.pedigree.mixblup

Usage

```
write.pedigree(
  population,
  path,
  gen = NULL,
  database = NULL,
  cohorts = NULL,
  id = NULL,
  depth.pedigree = 7,
  storage.save = 1.5,
  verbose = TRUE,
  mixblup.reliability = FALSE,
  blupf90 = FALSE,
  include.error = TRUE
)
```

Arguments

`population`	Population list
`path`	AA
`gen`	AA
`database`	AA
`cohorts`	AA
`id`	AA
`depth.pedigree`	AA
`storage.save`	AA
`verbose`	AA
`mixblup.reliability`	AA
`blupf90`	FALSE for mixblup; TRUE for MixBLUP
`include.error`	AA

Value

pedigree table

Index