Package ‘MoBPS’

January 20, 2025

Type Package

Title Modular Breeding Program Simulator

Version 1.6.64

Author Torsten Pook

Maintainer Torsten Pook <`torsten.pook@uni-goettingen.de`>

Description Framework for the simulation framework for the simulation of complex breeding programs and compare their economic and genetic impact. The package is also used as the background simulator for our a web-based interface <http:www.mobps.de>. 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

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

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

RoxygenNote 7.1.1

Date 2021-11-03

NeedsCompilation no

Repository CRAN

Date/Publication 2021-11-09 16:50:18 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 = 100)
population <- add.combi(population, trait = 3, combi.weights = c(1,5))
```

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.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 = "vanRaden", 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
)
```

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

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

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`	bitweise gespeicherte SNP-Sequenz
`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,
  mutation.rate = 10^-8,
  remutation.rate = 10^-8,
  recombination.rate = 1,
  selection.m = NULL,
  selection.f = NULL,
  new.selection.calculation = TRUE,
  selection.function.matrix = NULL,
  selection.size = 0,
  ignore.best = 0,
  breeding.size = 0,
  breeding.sex = NULL,
  breeding.sex.random = FALSE,
  relative.selection = FALSE,
  class.m = 0,
  class.f = 0,
  add.gen = 0,
  recom.f.indicator = NULL,
  duplication.rate = 0,
  duplication.length = 0.01,
  duplication.recombination = 1,
  new.class = 0L,
  bve = FALSE,
  sigma.e = NULL,
  sigma.g = 100,
  new.bv.child = NULL,
  phenotyping.child = NULL,
  relationship.matrix = "vanRaden",
  relationship.matrix.ogc = "kinship",
  computation.A = NULL,
  computation.A.ogc = NULL,
  delete.haplotypes = NULL,
  delete.individuals = NULL,
  fixed.breeding = NULL,
  fixed.breeding.best = NULL,
  max.offspring = Inf,
  max.litter = Inf,
  store.breeding.totals = FALSE,
  forecast.sigma.g = TRUE,
  multiple.bve = "add",
  store.bve.data = FALSE,
  fixed.assignment = FALSE,
  reduce.group = NULL,
  reduce.group.selection = "random",
  selection.highest = c(TRUE, TRUE),
  selection.criteria = NULL,
  same.sex.activ = FALSE,
  same.sex.sex = 0.5,
  same.sex.selfing = FALSE,
  selfing.mating = FALSE,
  selfing.sex = 0.5,
  praeimplantation = NULL,
  heritability = NULL,
  repeatability = NULL,
  save.recombination.history = FALSE,
  martini.selection = FALSE,
  BGLR.bve = FALSE,
  BGLR.model = "RKHS",
  BGLR.burnin = 500,
  BGLR.iteration = 5000,
  BGLR.print = FALSE,
  copy.individual = FALSE,
  copy.individual.m = FALSE,
  copy.individual.f = FALSE,
  dh.mating = FALSE,
  dh.sex = 0.5,
  n.observation = NULL,
  bve.0isNA = FALSE,
  phenotype.bv = FALSE,
  delete.same.origin = FALSE,
  remove.effect.position = FALSE,
  estimate.u = FALSE,
  new.phenotype.correlation = NULL,
  new.residual.correlation = NULL,
  new.breeding.correlation = NULL,
  estimate.add.gen.var = FALSE,
  estimate.pheno.var = FALSE,
  best1.from.group = NULL,
  best2.from.group = NULL,
  best1.from.cohort = NULL,
  best2.from.cohort = NULL,
  add.class.cohorts = TRUE,
  store.comp.times = TRUE,
  store.comp.times.bve = TRUE,
  store.comp.times.generation = TRUE,
  import.position.calculation = NULL,
  BGLR.save = "RKHS",
  BGLR.save.random = FALSE,
  ogc = FALSE,
  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,
  emmreml.bve = FALSE,
  rrblup.bve = FALSE,
  sommer.bve = FALSE,
  sommer.multi.bve = FALSE,
  nr.edits = 0,
  gene.editing.offspring = FALSE,
  gene.editing.best = FALSE,
  gene.editing.offspring.sex = c(TRUE, TRUE),
  gene.editing.best.sex = c(TRUE, TRUE),
  gwas.u = FALSE,
  approx.residuals = TRUE,
  sequenceZ = FALSE,
  maxZ = 5000,
  maxZtotal = 0,
  delete.sex = 1:2,
  gwas.group.standard = FALSE,
  y.gwas.used = "pheno",
  gen.architecture.m = 0,
  gen.architecture.f = NULL,
  add.architecture = NULL,
  ncore = 1,
  ncore.generation = 1,
  Z.integer = FALSE,
  store.effect.freq = FALSE,
  backend = "doParallel",
  randomSeed = NULL,
  randomSeed.generation = NULL,
  Rprof = FALSE,
  miraculix = NULL,
  miraculix.cores = 1,
  miraculix.mult = NULL,
  miraculix.chol = TRUE,
  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,
  bve.class = NULL,
  parallel.generation = FALSE,
  name.cohort = NULL,
  display.progress = TRUE,
  combine = FALSE,
  repeat.mating = NULL,
  repeat.mating.copy = NULL,
  repeat.mating.fixed = NULL,
  repeat.mating.overwrite = TRUE,
  time.point = 0,
  creating.type = 0,
  multiple.observation = FALSE,
  new.bv.observation = NULL,
  new.bv.observation.gen = NULL,
  new.bv.observation.cohorts = NULL,
  new.bv.observation.database = NULL,
  phenotyping = NULL,
  phenotyping.gen = NULL,
  phenotyping.cohorts = NULL,
  phenotyping.database = NULL,
  bve.gen = NULL,
  bve.cohorts = NULL,
  bve.database = NULL,
  sigma.e.gen = NULL,
  sigma.e.cohorts = NULL,
  sigma.e.database = NULL,
  sigma.g.gen = NULL,
  sigma.g.cohorts = NULL,
  sigma.g.database = NULL,
  gwas.gen = NULL,
  gwas.cohorts = NULL,
  gwas.database = NULL,
  bve.insert.gen = NULL,
  bve.insert.cohorts = NULL,
  bve.insert.database = NULL,
  reduced.selection.panel.m = NULL,
  reduced.selection.panel.f = NULL,
  breeding.all.combination = FALSE,
  depth.pedigree = 7,
  depth.pedigree.ogc = 7,
  copy.individual.keep.bve = TRUE,
  copy.individual.keep.pheno = TRUE,
  bve.avoid.duplicates = TRUE,
  report.accuracy = TRUE,
  share.genotyped = 1,
  singlestep.active = FALSE,
  remove.non.genotyped = TRUE,
  added.genotyped = 0,
  fast.uhat = TRUE,
  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,
  culling.gen = NULL,
  culling.database = NULL,
  culling.cohort = NULL,
  culling.time = Inf,
  culling.name = "Not_named",
  culling.bv1 = 0,
  culling.share1 = 0,
  culling.bv2 = NULL,
  culling.share2 = NULL,
  culling.index = 0,
  culling.single = TRUE,
  culling.all.copy = TRUE,
  calculate.reliability = FALSE,
  selection.m.gen = NULL,
  selection.f.gen = NULL,
  selection.m.database = NULL,
  selection.f.database = NULL,
  selection.m.cohorts = NULL,
  selection.f.cohorts = NULL,
  selection.m.miesenberger = FALSE,
  selection.f.miesenberger = NULL,
  selection.miesenberger.reliability.est = "estimated",
  miesenberger.trafo = 0,
  multiple.bve.weights.m = 1,
  multiple.bve.weights.f = NULL,
  multiple.bve.scale.m = "bv_sd",
  multiple.bve.scale.f = NULL,
  verbose = TRUE,
  bve.parent.mean = FALSE,
  bve.grandparent.mean = FALSE,
  bve.mean.between = "bvepheno",
  bve.direct.est = TRUE,
  bve.pseudo = FALSE,
  bve.pseudo.accuracy = 1,
  miraculix.destroyA = TRUE,
  mas.bve = FALSE,
  mas.markers = NULL,
  mas.number = 5,
  mas.effects = NULL,
  threshold.selection = NULL,
  threshold.sign = ">",
  input.phenotype = "own",
  bve.ignore.traits = NULL,
  bv.ignore.traits = NULL,
  genotyped.database = NULL,
  genotyped.gen = NULL,
  genotyped.cohorts = NULL,
  genotyped.share = 1,
  genotyped.array = 1,
  sex.s = NULL,
  bve.imputation = TRUE,
  bve.imputation.errorrate = 0,
  share.phenotyped = 1,
  avoid.mating.fullsib = FALSE,
  avoid.mating.halfsib = FALSE,
  max.mating.pair = Inf,
  bve.per.sample.sigma.e = TRUE,
  bve.solve = "exact"
)
```

Arguments

`population`	Population list
`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)
`selection.m`	Selection criteria for male individuals (Set to "random" to randomly select individuals - this happens automatically when no the input in selection.criteria has no input ((usually breeding values)))
`selection.f`	Selection criteria for female individuals (default: selection.m , alt: "random", function")
`new.selection.calculation`	If TRUE recalculate breeding values obtained by selection.function.matrix
`selection.function.matrix`	Manuel generation of a temporary selection function (Use BVs instead!)
`selection.size`	Number of selected individuals for breeding (default: c(0,0) - alt: positive numbers)
`ignore.best`	Not consider the top individuals of the selected individuals (e.g. to use 2-10 best individuals)
`breeding.size`	Number of individuals to generate

`breeding.sex`	Share of female animals (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)
`relative.selection`	Use best.selection.ratio instead!
`class.m`	Migrationlevels of male individuals to consider for mating process (default: 0)
`class.f`	Migrationlevels of female individuals to consider for mating process (default: 0)
`add.gen`	Generation you want to add the new individuals to (default: New generation)
`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)
`new.class`	Migration level of newly generated individuals (default: 0)
`bve`	If TRUE perform a breeding value estimation (default: FALSE)
`sigma.e`	Enviromental variance (default: 100)
`sigma.g`	Genetic variance (default: 100 - only used if not computed via estimate.sigma.g^2 in der Zuchtwertschaetzung (Default: 100)
`new.bv.child`	(OLD! - use phenotyping.child) Starting phenotypes of newly generated individuals (default: "mean" of both parents, "obs" - regular observation, "zero" - 0)
`phenotyping.child`	Starting phenotypes of newly generated individuals (default: "mean" of both parents, "obs" - regular observation, "zero" - 0)
`relationship.matrix`	Method to calculate relationship matrix for the breeding value estimation (Default: "vanRaden", alt: "kinship", "CE", "non_stand", "CE2", "CM")
`relationship.matrix.ogc`	Method to calculate relationship matrix for OGC (Default: "kinship", alt: "vanRaden", "CE", "non_stand", "CE2", "CM")
`computation.A`	(OLD! - use relationship.matrix) Method to calculate relationship matrix for the breeding value estimation (Default: "vanRaden", alt: "kinship", "CE", "non_stand", "CE2", "CM")
`computation.A.ogc`	(OLD! use relationship.matrix.ogc) Method to calculate pedigree matrix in OGC (Default: "kinship", alt: "vanRaden", "CE", "non_stand", "CE2", "CM")
`delete.haplotypes`	Generations for with haplotypes of founders can be deleted (only use if storage problem!)

`delete.individuals`	Generations for with individuals are completley deleted (only use if storage problem!)
`fixed.breeding`	Set of targeted matings to perform
`fixed.breeding.best`	Perform targeted matings in the group of selected individuals
`max.offspring`	Maximum number of offspring per individual (default: c(Inf,Inf) - (m,w))
`max.litter`	Maximum number of offspring per individual (default: c(Inf,Inf) - (m,w))
`store.breeding.totals`	If TRUE store information on selected animals in $info$breeding.totals
`forecast.sigma.g`	Set FALSE to not estimate sigma.g (Default: TRUE)
`multiple.bve`	Way to handle multiple traits in bv/selection (default: "add", alt: "ranking")
`store.bve.data`	If TRUE store information of bve in $info$bve.data
`fixed.assignment`	Set TRUE for targeted mating of best-best individual till worst-worst (of selected). set to "bestworst" for best-worst mating
`reduce.group`	(OLD! - use culling modules) Groups of animals 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")
`selection.highest`	If 0 individuals with lowest bve are selected as best individuals (default c(1,1) - (m,w))
`selection.criteria`	What to use in the selection proces (default: "bve", alt: "bv", "pheno")
`same.sex.activ`	If TRUE allow matings of individuals of same sex
`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
`selfing.mating`	If TRUE generate new individuals via selfing
`selfing.sex`	Share of female individuals used for selfing (default: 0.5)
`praeimplantation`	Only use matings the lead to a specific genotype in a specific marker
`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)
`save.recombination.history`	If TRUE store the time point of each recombination event
`martini.selection`	If TRUE use the group of non-selected individuals as second parent
`BGLR.bve`	If TRUE use BGLR to perform breeding value estimation

`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
`copy.individual`	If TRUE copy the selected father for a mating
`copy.individual.m`	If TRUE generate exactly one copy of all selected male in a new cohort (or more by setting breeding.size)
`copy.individual.f`	If TRUE generate exactly one copy of all selected female in a new cohort (or more by setting breeding.size)
`dh.mating`	If TRUE generate a DH-line in mating process
`dh.sex`	Share of DH-lines generated from selected female individuals
`n.observation`	Number of phenotypes generated per individuals (influences enviromental variance)
`bve.0isNA`	Individuals with phenotype 0 are used as NA in breeding value estimation
`phenotype.bv`	If TRUE use phenotype as estimated breeding value
`delete.same.origin`	If TRUE delete recombination points when genetic origin of adjacent segments is the same
`remove.effect.position`	If TRUE remove real QTLs in breeding value estimation
`estimate.u`	If TRUE estimate u in breeding value estimation (Y = Xb + Zu + e)
`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!)
`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
`best1.from.group`	(OLD!- use selection.m.database) Groups of individuals to consider as First Parent / Father (also female individuals are possible)
`best2.from.group`	(OLD!- use selection.f.database) Groups of individuals to consider as Second Parent / Mother (also male individuals are possible)

`best1.from.cohort`	(OLD!- use selection.m.cohorts) Groups of individuals to consider as First Parent / Father (also female individuals are possible)
`best2.from.cohort`	(OLD! - use selection.f.cohorts) Groups of individuals to consider as Second Parent / Mother (also male individuals are possible)
`add.class.cohorts`	Migration levels of all cohorts selected for reproduction are automatically added to class.m/class.f (default: TRUE)
`store.comp.times`	If TRUE store computation times in $info$comp.times (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)
`import.position.calculation`	Function to calculate recombination point into adjacent/following SNP
`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!)
`ogc`	If TRUE use optimal genetic contribution theory to perform selection (This requires the use of the R-package optiSel)
`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
`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 mulit-trait model in the R-package sommer for BVE
`nr.edits`	Number of edits to perform per individual
`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)
`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 (doesnt change order - only p-values)
`sequenceZ`	Split genomic matric into parts (relevent if high memory usage)
`maxZ`	Number of SNPs to consider in each part of sequenceZ
`maxZtotal`	Number of matrix entries to consider jointly (maxZ = maxZtotal/number of animals)
`delete.sex`	Remove all individuals from these sex from generation delete.individuals (default: 1:2 ; note:delete individuals=NULL)
`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")
`gen.architecture.m`	Genetic architecture for male animal (default: 0 - no transformation)
`gen.architecture.f`	Genetic architecture for female animal (default: gen.architecture.m - no transformation)
`add.architecture`	List with two vectors containing (A: length of chromosomes, B: position in cM of SNPs)
`ncore`	Cores used for parallel computing in compute.snps
`ncore.generation`	Number of cores to use in parallel generation
`Z.integer`	If TRUE save Z as a integer in parallel computing
`store.effect.freq`	If TRUE store the allele frequency of effect markers per generation
`backend`	Chose the used backend (default: "doParallel", alt: "doMPI")
`randomSeed`	Set random seed of the process

`randomSeed.generation`	Set random seed for parallel generation process
`Rprof`	Store computation times of each function
`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)
`best.selection.ratio.m`	Ratio of the frequency of the selection of the best best animal and the worst best animal (default=1)
`best.selection.ratio.f`	Ratio of the frequency of the selection of the best best animal and the worst best animal (default=1)
`best.selection.criteria.m`	Criteria to calculate this ratio (default: "bv", alt: "bve", "pheno")
`best.selection.criteria.f`	Criteria to calculate this ratio (default: "bv", alt: "bve", "pheno")
`best.selection.manual.ratio.m`	vector containing probability to draw from for every individual (e.g. c(0.1,0.2,0.7))
`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
`bve.class`	Consider only animals of those class classes in breeding value estimation (default: NULL - use all)
`parallel.generation`	Set TRUE to active parallel computing in animal generation
`name.cohort`	Name of the newly added cohort
`display.progress`	Set FALSE to not display progress bars. Setting verbose to FALSE will automatically deactive progress bars
`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
`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.individuals.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)
`time.point`	Time point at which the new individuals are generated
`creating.type`	Technique to generate new individuals (usage in web-based application)
`multiple.observation`	Set TRUE to allow for more than one phenotype observation per individual (this will decrease enviromental variance!)
`new.bv.observation`	(OLD! - use phenotyping) Quick acces to phenotyping for (all: "all", non-phenotyped: "non_obs", non-phenotyped male: "non_obs_m", non-phenotyped female: "non_obs_f")
`new.bv.observation.gen`	(OLD! use phenotyping.gen) Vector of generation from which to generate additional phenotypes
`new.bv.observation.cohorts`	(OLD! use phenotyping.cohorts)Vector of cohorts from which to generate additional phenotype
`new.bv.observation.database`	(OLD! use phenotyping.database) Matrix of groups from which to generate additional phenotypes
`phenotyping`	Quick acces to phenotyping for (all: "all", non-phenotyped: "non_obs", non-phenotyped male: "non_obs_m", non-phenotyped female: "non_obs_f")
`phenotyping.gen`	Vector of generation from which to generate additional phenotypes
`phenotyping.cohorts`	Vector of cohorts from which to generate additional phenotype
`phenotyping.database`	Matrix of groups from which to generate additional phenotypes
`bve.gen`	Generations of individuals to consider in breeding value estimation (default: NULL)
`bve.cohorts`	Cohorts of individuals to consider in breeding value estimation (default: NULL)
`bve.database`	Groups of individuals to consider in breeding value estimation (default: NULL)
`sigma.e.gen`	Generations to consider when estimating sigma.e when using hertability
`sigma.e.cohorts`	Cohorts to consider when estimating sigma.e when using hertability
`sigma.e.database`	Groups to consider when estimating sigma.e when using hertability
`sigma.g.gen`	Generations to consider when estimating sigma.g
`sigma.g.cohorts`	Cohorts to consider when estimating sigma.g

`sigma.g.database`	Groups to consider when estimating sigma.g
`gwas.gen`	Generations to consider in GWAS analysis
`gwas.cohorts`	Cohorts to consider in GWAS analysis
`gwas.database`	Groups to consider in GWAS analysis
`bve.insert.gen`	Generations of individuals to compute breeding values for (default: all groups in bve.database)
`bve.insert.cohorts`	Cohorts of individuals to compute breeding values for (default: all groups in bve.database)
`bve.insert.database`	Groups of individuals to compute breeding values for (default: all groups in bve.database)
`reduced.selection.panel.m`	Use only a subset of individuals of the potential selected ones ("Split in user-interface")
`reduced.selection.panel.f`	Use only a subset of individuals of the potential selected ones ("Split in user-interface")
`breeding.all.combination`	Set to TRUE to automatically perform each mating combination possible exactly ones.
`depth.pedigree`	Depth of the pedigree in generations (default: 7)
`depth.pedigree.ogc`	Depth of the pedigree in generations (default: 7)
`copy.individual.keep.bve`	Set to FALSE to not keep estimated breeding value in case of use of copy.individuals
`copy.individual.keep.pheno`	Set to FALSE to not keep estimated breeding values in case of use of copy.individuals
`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)
`report.accuracy`	Report the accuracy of the breeding value estimation
`share.genotyped`	Share of individuals newly generated individuals that are genotyped
`singlestep.active`	Set TRUE to use single step in breeding value estimation (only implemented for vanRaden- G matrix and without use sequenceZ) (Legarra 2014)
`remove.non.genotyped`	Set to FALSE to manually include non-genotyped individuals in genetic BVE, single-step will deactive this as well
`added.genotyped`	Share of individuals that is additionally genotyped (only for copy.individuals)

`fast.uhat`	Set to FALSE to derive inverse of A in rrBLUP
`offspring.bve.parents.gen`	Generations to consider to derive phenotype from offspring phenotypes
`offspring.bve.parents.database`	Groups to consider to derive phenotype from offspring phenotypes
`offspring.bve.parents.cohorts`	Cohorts to consider to derive phenotype from offspring phenotypes
`offspring.bve.offspring.gen`	Active generations for import of offspring phenotypes
`offspring.bve.offspring.database`	Active groups for import of offspring phenotypes
`offspring.bve.offspring.cohorts`	Active cohorts for import of offspring phenotypes
`culling.gen`	Generations to consider to culling
`culling.database`	Groups to consider to culling
`culling.cohort`	Cohort to consider to culling
`culling.time`	Age of the individuals at culling
`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
`calculate.reliability`	Set TRUE to calculate a reliability when performing Direct-Mixed-Model BVE
`selection.m.gen`	Generations available for selection of paternal parent
`selection.f.gen`	Generations available for selection of maternal parent
`selection.m.database`	Groups available for selection of paternal parent
`selection.f.database`	Groups available for selection of maternal parent
`selection.m.cohorts`	Cohorts available for selection of paternal parent
`selection.f.cohorts`	Cohorts available for selection of maternal parent

`selection.m.miesenberger`	Use Weighted selection index according to Miesenberger 1997 for paternal selection
`selection.f.miesenberger`	Use Weighted selection index according to Miesenberger 1997 for maternal selection
`selection.miesenberger.reliability.est`	If available reliability estimated are used. If not use default:"estimated" (SD BVE / SD Pheno), alt: "heritability", "derived" (cor(BVE,BV)^2) as replacement
`miesenberger.trafo`	Ignore all eigenvalues below this threshold and apply dimension reduction (default: 0 - use all)
`multiple.bve.weights.m`	Weighting between traits when using "add" (default: 1)
`multiple.bve.weights.f`	Weighting between traits when using "add" (default: same as multiple.bve.weights.m)
`multiple.bve.scale.m`	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
`multiple.bve.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
`verbose`	Set to FALSE to not display any prints
`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.direct.est`	If TRUE predict BVEs in direct estimation according to vanRaden 2008 method 2 (default: TRUE)
`bve.pseudo`	If set to TRUE the breeding value estimation will be simulated with resulting accuracy bve.pseudo.accuracy (default: 1)
`bve.pseudo.accuracy`	The accuracy to be obtained in the "pseudo" - breeding value estimation
`miraculix.destroyA`	If FALSE A will not be destroyed in the process of inversion (less computing / more memory)
`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())
`threshold.selection`	Minimum value in the selection index selected individuals have to have
`threshold.sign`	Pick all individuals above (">") the threshold. Alt: ("<", "=", "<=", ">=")
`input.phenotype`	Select what to use in BVE (default: own phenotype ("own"), offspring phenotype ("off"), their average ("mean") or a weighted average ("weighted"))
`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 multiple.bve.weights.m/.w)
`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!)
`genotyped.database`	Groups to generate genotype data (that can be used in a BVE)
`genotyped.gen`	Generations to generate genotype data (that can be used in a BVE)
`genotyped.cohorts`	Cohorts 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
`sex.s`	Specify which newly added individuals are male (1) or female (2)
`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)
`share.phenotyped`	Share of the individuals to phenotype
`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
`max.mating.pair`	Set to the maximum number of matings between two individuals (default: Inf)
`bve.per.sample.sigma.e`	Set to FALSE to deactivate the use of a heritablity based on the number of observations generated per sample
`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)

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
)
```

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]]

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)
```

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 separte 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 = 100,
  var.target = 10,
  gen = NULL,
  database = NULL,
  cohorts = NULL,
  adapt.bve = FALSE,
  adapt.pheno = FALSE,
  verbose = FALSE
)
```

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: FALSE)
`adapt.pheno`	Modify previous phenotypes by scaling (default: FALSE)
`verbose`	Set to TRUE to display prints

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@uni-goettingen.de`>

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)
```

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)

Value

logical with TRUE if relatedness does not excced 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@uni-goettingen.de`>

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)
```

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

Description

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

Usage

```
compute.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 geneation of a genotype
`fix.costs`	one time occuring fixed costs
`fix.costs.annual`	annually occuring 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)
compute.costs(ex_pop, gen=1:5)
```

---

compute.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

```
compute.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 geneation of a genotype
`housing.costs`	Costs for housing
`fix.costs`	one time occuring fixed costs
`fix.costs.annual`	annually occuring 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)
compute.costs.cohorts(ex_pop, gen=1:5, genotyping.costs=25, json=FALSE)
```

`compute.snps` *Compute genotype/haplotype*

Description

Internal function for the computation of genotypes & haplotypes

Usage

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

---

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

Description

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

Usage

```
compute.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(
  dataset = NULL,
  vcf = NULL,
  chr.nr = NULL,
  bp = NULL,
  snp.name = NULL,
  hom0 = NULL,
  hom1 = NULL,
  bpcm.conversion = 0,
  nsnp = 0,
  nindi = 0,
  freq = "beta",
  population = NULL,
  sex.s = "fixed",
  add.chromosome = FALSE,
  generation = 1,
  class = 0L,
  sex.quota = 0.5,
  chromosome.length = NULL,
  length.before = 5,
  length.behind = 5,
  real.bv.add = NULL,
  real.bv.mult = NULL,
  real.bv.dice = NULL,
  snps.equidistant = NULL,
  change.order = FALSE,
  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.equal.additive = 0,
  n.dominant = 0,
  n.equal.dominant = 0,
  n.qualitative = 0,
  n.quantitative = 0,
  dominant.only.positive = FALSE,
  var.additive.l = NULL,
  var.dominant.l = NULL,
  var.qualitative.l = NULL,
  var.quantitative.l = NULL,
  effect.size.equal.add = 1,
  effect.size.equal.dom = 1,
  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,
  share.genotyped = 1,
  genotyped.s = NULL,
  map = NULL,
  remove.invalid.qtl = TRUE,
  verbose = TRUE,
  bv.standard = FALSE,
  mean.target = NULL,
  var.target = NULL,
  is.maternal = NULL,
  is.paternal = NULL,
  vcf.maxsnp = Inf,
  internal = FALSE
)
```

Arguments

`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 assosiated 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
`nindi`	number of inidividuals to generate in a random dataset
`freq`	frequency of allele 1 when randomly generating a dataset
`population`	Population list

`sex.s`	Specify which newly added individuals are male (1) or female (2)
`add.chromosome`	If TRUE add an additional chromosome to the dataset
`generation`	Generation of the newly added individuals (default: 1)
`class`	Migration level of the newly added individuals
`sex.quota`	Share of newly added female individuals (deterministic if sex.s="fixed", alt: sex.s="random")
`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)
`change.order`	If TRUE sort markers according to given marker positions
`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 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 n.equal.dominant QTL with equal effect size
`n.qualitative`	Number of qualitative epistatic QTL
`n.quantitative`	Number of quantitative epistatic QTL
`dominant.only.positive`	Set to TRUE to always asign the heterozygous variant with the higher of the two homozygous effects (e.g. hybrid breeding); default: FALSE
`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
`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
`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 shuffeled 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

`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
`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)
`internal`	Dont touch!

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
)
```

Arguments

`population`	Population list
`phenotypic.transform.function`	Phenotypic transformation to apply
`trait`	Trait for which a transformation is to be applied data(ex_pop) trafo <- function(x) return(x^2) ex_pop <- creating.phenotypic.transform(ex_pop, phenotypic.transform.function=traf

Value

Population-list with a new phenotypic transformation function

---

`creating.trait` *Generation of genomic traits*

Description

Generation of the trait in a starting population

Usage

```
creating.trait(
  population,
  real.bv.add = NULL,
  real.bv.mult = NULL,
  real.bv.dice = NULL,
  bv.total = 0,
  polygenic.variance = 100,
  bve.mult.factor = NULL,
  bve.poly.factor = NULL,
  base.bv = NULL,
  new.phenotype.correlation = NULL,
  new.residual.correlation = NULL,
  new.breeding.correlation = NULL,
  n.additive = 0,
  n.equal.additive = 0,
  n.dominant = 0,
  n.equal.dominant = 0,
  n.qualitative = 0,
  n.quantitative = 0,
  dominant.only.positive = FALSE,
  var.additive.l = NULL,
  var.dominant.l = NULL,
  var.qualitative.l = NULL,
  var.quantitative.l = NULL,
  effect.size.equal.add = 1,
  effect.size.equal.dom = 1,
  exclude.snps = NULL,
  randomSeed = NULL,
  shuffle.traits = NULL,
  shuffle.cor = NULL,
  replace.traits = FALSE,
  trait.name = NULL,
  remove.invalid.qtl = TRUE,
  bv.standard = FALSE,
  mean.target = NULL,
  var.target = NULL,
  verbose = TRUE,
  is.maternal = NULL,
  is.paternal = NULL
)
```

Arguments

`population`	Population list
`real.bv.add`	Single Marker effects
`real.bv.mult`	Two Marker effects
`real.bv.dice`	Multi-marker effects
`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
`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!
`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 n.equal.dominant QTL with equal effect size
`n.qualitative`	Number of qualitative epistatic QTL

`n.quantitative`	Number of quantitative epistatic QTL
`dominant.only.positive`	Set to TRUE to always asign the heterozygous variant with the higher of the two homozygous effects (e.g. hybrid breeding); default: FALSE
`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
`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
`exclude.snps`	Marker were no QTL are simulated on
`randomSeed`	Set random seed of the process
`shuffle.traits`	Combine different traits into a joined trait
`shuffle.cor`	Target Correlation between shuffeled traits
`replace.traits`	If TRUE delete the simulated traits added before
`trait.name`	Name of the trait generated
`remove.invalid.qtl`	Set to FALSE to deactive the automatic removal of QTLs on markers that do not exist
`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
`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)

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` *Add a genotyping array*

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

`ex_json` *ex_json*

Description

Exemplary json-data

Usage

```
ex_json
```

Author(s)

Torsten Pook <`torsten.pook@uni-goettingen.de`>

Source

Web-interface

`ex_pop` *ex_pop*

Description

Exemplary population-list

Usage

```
ex_pop
```

Author(s)

Torsten Pook <`torsten.pook@uni-goettingen.de`>

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 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,
  change.order = FALSE,
  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
)
```

Arguments

`nindi`	number of inidividuals 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 assosiated 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)
`change.order`	If TRUE sort markers according to given 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
`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)

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,
  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)
`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 devide 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,
  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)
use.id	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: FALSE)

Value

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

Examples

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

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, 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)
use.id	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: FALSE)

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

`get.class` *Derive class*

Description

Function to devide the class for each individual

Usage

```
get.class(
  population,
  database = NULL,
  gen = NULL,
  cohorts = 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)
`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.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.creating.type` *Derive creating type*

Description

Function to devide creating type for each individual

Usage

```
get.creating.type(
  population,
  database = NULL,
  gen = NULL,
  cohorts = 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)
`use.id`	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: FALSE)

Value

Creating type of in gen/database/cohorts selected individuals

Examples

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

`get.cullingtime` *Derive time of culling*

Description

Function to devide the time of culling for all individuals

Usage

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

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

Value

Combine gen/database/cohorts to a joined database

Examples

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

get.death.point *Derive death point*

Description

Function to devide 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,
  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)
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.death.point(ex_pop, gen=2)
```

get.dendrogram *Dendrogram*

Description

Function calculate a dendogram

Usage

```
get.dendrogram(
  population,
  path = NULL,
  database = NULL,
  gen = NULL,
  cohorts = 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)
`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 dendogram

Usage

```
get.dendrogram.heatmap(
  population,
  path = NULL,
  database = NULL,
  gen = NULL,
  cohorts = 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)
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),
          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.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,
  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)
`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,
  gen2 = NULL,
  database2 = NULL,
  cohorts2 = 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)
`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)
`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,
  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)
`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)
```

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)

Value

Estimated effective population size

Examples

```
data(ex_pop)
get.effective.size(population=ex_pop, gen=5)
```

`get.geno` *Derive genotypes of selected individuals*

Description

Function to devide genotypes of selected individuals

Usage

```
get.geno(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  chromosomen = "all",
  export.alleles = FALSE,
  non.genotyped.as.missing = FALSE,
  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)
`chromosomen`	Beschraenkung des Genotypen auf bestimmte Chromosomen (default: 1)
`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: FALSE)

Value

Genotype data for in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
geno <- get.geno(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,
  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)
`use.id`	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: FALSE)

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 devide which markers are genotyped for the selected individuals

Usage

```
get.genotyped.snp(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  export.alleles = FALSE,
  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)
`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: FALSE)

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 devide haplotypes of selected individuals

Usage

```
get.haplo(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  chromosomen = "all",
  export.alleles = FALSE,
  non.genotyped.as.missing = FALSE,
  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)
`chromosomen`	Beschraenkung der Haplotypen auf bestimmte Chromosomen (default: 1)
`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: FALSE)

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, 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)
use.id	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names

Value

Individual ID for in gen/database/cohorts selected individuals

Examples

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

get.individual.loc — *Export location of individuals from the population list*

Description

Export location of individuals from the population list

Usage

```
get.individual.loc(population, database = NULL, gen = NULL, 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)

Value

Storage Position for in gen/database/cohorts selected individuals (Generation/Sex/IndividualNr)

Examples

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

---

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,
  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)
`use.id`	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: FALSE)

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.map *Map generation*

Description

Function to derive the genomic map for a given population list

Usage

```
get.map(population, use.snp.nr = FALSE)
```

Arguments

`population`	Population list
`use.snp.nr`	Set to TRUE to display SNP number and not SNP name

Value

Genomic map of the population list

Examples

```
data(ex_pop)
map <- get.map(ex_pop)
```

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,
  use.all.copy = FALSE,
  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)
`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: FALSE)

Value

Phenotypes for in gen/database/cohorts selected individuals

Examples

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

`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,
  coloring = "group",
  components = c(1, 2),
  plot = TRUE,
  pch = 1,
  export.color = 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)
`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

Value

Genotype data for in 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,
  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)
`founder.zero`	Parents of founders are displayed as "0" (default: TRUE)
`raw`	Set to TRUE to not convert numbers into Sex etc.
`id`	Set to TRUE to extract individual IDs

Value

Pedigree-file for in gen/database/cohorts selected individuals

Examples

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

`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
)
```

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)

Value

Pedigree-file (grandparents) for in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
get.pedigree2(ex_pop, gen=2)
```

---

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
)
```

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)

Value

Pedigree-file (parents + grandparents) for in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
get.pedigree3(ex_pop, gen=3)
```

`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,
  non.genotyped.as.missing = FALSE,
  use.id = FALSE
)
```

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)
`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

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.pheno — *Export underlying phenotypes*

Description

Function to export underlying phenotypes

Usage

```
get.pheno(
  population,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  use.all.copy = FALSE,
  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)
`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: FALSE)

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,
  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)
`use.id`	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: FALSE)

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)
```

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)

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.phylogenetic.tree *Phylogenetic Tree*

Description

Function calculate a phylogenetic tree

Usage

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

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)

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,
  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)
`use.id`	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: FALSE)

Value

Recombination points for in gen/database/cohorts selected individuals

Examples

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

`get.reliabilities` *Export underlying reliabilities*

Description

Function to export underlying reliabilities

Usage

```
get.reliabilities(
  population,
  database = NULL,
  gen = NULL,
  cohorts = 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)
`use.id`	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: FALSE)

Value

Estimated reliability for BVE for in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
get.reliabilities(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,
  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)
`use.id`	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: FALSE)

Value

Last applied selection index for in gen/database/cohorts selected individuals

Examples

```
data(ex_pop)
get.selectionindex(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,
  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)
use.id	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: FALSE)

Value

Last applied selection index for in gen/database/cohorts selected individuals

Examples

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

---

get.time.point   *Derive time point*

Description

Function to devide time point for each individual

Usage

```
get.time.point(
  population,
  database = NULL,
  gen = NULL,
  cohorts = 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)
use.id	Set to TRUE to use MoBPS ids instead of Sex_Nr_Gen based names (default: FALSE)

Value

Time point of generation for in gen/database/cohorts selected individuals

Examples

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

---

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,
  chromosomen = "all",
  non.genotyped.as.missing = FALSE,
  use.id = 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)
`chromosomen`	Beschraenkung des Genotypen auf bestimmte Chromosomen (default: 1)
`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

Value

VCF-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"))
```

---

group.diff *Function to 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))
```

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)
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)
```

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,
  fixed.generation.order = NULL
)
```

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 actived

`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
`fixed.generation.order`	Vector containing the order of cohorts to generate (Advanced // Testing Parameter!)

Value

Population-list

Examples

```
data(ex_json)
population <- json.simulation(total=ex_json)
```

`kinship.development` *Devolopment 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(
  animals = NULL,
  population = NULL,
  gen = NULL,
  database = NULL,
  cohorts = NULL,
  sym = FALSE
)
```

Arguments

`animals`	List of animals to compute kinship for
`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)
`sym`	If True derive matrix entries below principle-diagonal

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(
  animals = NULL,
  population = NULL,
  gen = NULL,
  database = NULL,
  cohorts = NULL,
  sym = FALSE,
  ibd.obs = 50,
  hbd.obs = 10
)
```

Arguments

`animals`	List of animals to compute kinship for
`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)
`sym`	If True derive matrix entries below principle-diagonal
`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.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 = 2,
  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

mult	Multiplicator of kinship matrix (default: 2)
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)
kinship <- kinship.exp(population=ex_pop, gen=2)
```

---

ld.decay *Generate LD plot*

Description

Generate LD pot

Usage

```
ld.decay(
  population,
  genotype.dataset = NULL,
  chromosomen = 1,
  dist = NULL,
  step = 5,
  max = 500,
  max.cases = 100,
  database = NULL,
  gen = NULL,
  cohorts = NULL,
  type = "snp",
  plot = FALSE
)
```

Arguments

population	Population list
genotype.dataset	Genotype dataset (default: NULL - just to save computation time when get.geno was already run)
chromosomen	Only consider a specific chromosome in calculations (default: 1)
dist	Manuel input of marker distances to analyse

`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

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@uni-goettingen.de`>

Source

Lee et al 2002

`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 covarianz matrix
`G`	Genomic covarianz matrix
`V1`	Inverted phenotypic covarianz 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, sex, individual.nr, qtl.posi, haplo.set = 1)
```

Arguments

`population`	Population list
`gen`	Generation of the individual to introduce a mutation in
`sex`	Sex of the individual to introduce a mutation in
`individual.nr`	Individual Nr. of the individual to introduce a mutation in
`qtl.posi`	Marker number to mutate
`haplo.set`	Select chromosome set (default: 1 , alt: 2)

Value

Population-List with mutated marker for the selected individual

Examples

```
data(ex_pop)
ex_pop <- mutation.intro(ex_pop, 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,
  delete.previous.gen = FALSE,
  delete.breeding.totals = FALSE,
  delete.bve.data = FALSE,
  add.chromosome.ends = TRUE
)
```

Arguments

`population`	Population list
`base.gen`	Vector containing all 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`	Deleta all previous bve data (default: FALSE)
`add.chromosome.ends`	Add chromosome ends as recombination points

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

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,
  change.order = FALSE,
  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,
  share.genotyped = 1,
  genotyped.s = NULL,
  map = NULL,
  remove.invalid.qtl = TRUE,
  verbose = TRUE,
  bv.standard = FALSE,
  mean.target = NULL,
  var.target = NULL,
  is.maternal = NULL,
  is.paternal = NULL,
  vcf.maxsnp = Inf
)
```

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 assosiated 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)
`change.order`	If TRUE sort markers according to given marker positions
`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 shuffeled 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
`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
`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)
`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=1000)
```

`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@uni-goettingen.de`>

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, genone length and trait overview

Examples

```
data(ex_pop)
plot(ex_pop)
```

set.class *Export estimated breeding values*

Description

Function to export estimated breeding values

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)
```

`sheep_chip` *sheep chip*

Description

Genome for sheep according to Prieur et al.

Usage

```
sheep_chip
```

Author(s)

Torsten Pook <`torsten.pook@uni-goettingen.de`>

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, genone 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)
```

Index