Package: SelectionTools (via r-universe)

October 9, 2026

Version 26.5

Date 2026-09-20

Title Simulation and Data Analysis for Plant Breeders

Description Provides tools for simulation of plant breeding programs as described, for example, by Melchinger and Frisch (2023) <doi:10.1007/s00122-023-04446-3>, prediction of segregation variance (Osthushenrich, Frisch and Herzog (2017) <doi:10.1371/journal.pone.0188839>), genomic prediction (Hofheinz and Frisch (2014) <doi:10.1534/g3.113.010025>), linkage disequilibrium based haplotype construction, and planning of marker assisted back crossing programs. It provides an integrated framework for simulation and analysis of plant breeding programs.

Suggests knitr, rmarkdown

VignetteBuilder knitr

Depends R (>= 4.0.0)

License CC0

NeedsCompilation yes

Author Matthias Frisch [aut, cre], Hans Peter Maurer [ctb], Philipp Heilmann [ctb]

Maintainer Matthias Frisch <matthias.frisch@uni-giessen.de>

Repository https://cran.r-universe.dev

Date/Publication 2026-10-09 16:21:21 UTC

RemoteUrl https://github.com/cran/SelectionTools

RemoteRef HEAD

RemoteSha 459ed67694735ae1e9881c634d72cc4f4d55d6d1

Contents

append.population 9
be.quiet 10

concat.population . . . 10
copy.population . . . 11
cross . . . 12
data.params . . . 14
define.effects . . . 14
define.effects.df . . . 15
define.genome . . . 16
define.map . . . 17
deprecated . . . 19
dh . . . 19
divide.population . . . 20
DT_technow . . . 21
effect.weight.set.all . . . 21
effmap.remove . . . 22
effmap.remove.all . . . 22
evaluate.all.loci . . . 23
evaluate.allele . . . 23
evaluate.allele.freq . . . 24
evaluate.allele.freq2 . . . 25
evaluate.allele2 . . . 26
evaluate.genome . . . 27
evaluate.genotype . . . 28
evaluate.hap.calc . . . 30
evaluate.hap.calc.d . . . 30
evaluate.hap.free . . . 31
evaluate.hap.init . . . 31
evaluate.hap.return.genotype . . . 32
evaluate.hap.return.table . . . 32
evaluate.haplotype . . . 33
evaluate.ld . . . 33
evaluate.locus . . . 34
evaluate.mdp . . . 34
evaluate.population . . . 35
gd.allele.frequencies . . . 36
gd.allow.zero.frequencies . . . 36
gd.correct.missing . . . 37
gd.data.parameters . . . 37
gd.distance.similarity . . . 38
gd.dummy.populations . . . 38
gd.genetic.distance . . . 39
gd.list.irregular . . . 40
gd.list.missing . . . 41
gd.maize.aflp . . . 41
gd.maize.lines . . . 42
gd.maize.populations . . . 42
gd.mk.matr . . . 43
gd.pcoa . . . 44
gd.salad.aflp . . . 44

gd.similarity.coefficient . . . 45
gd.splitdot . . . 46
gd.status.zero.frequencies . . . 47
generate.effect.file . . . 47
generate.map.file . . . 48
generate.population . . . 48
genome.contribution . . . 49
genome.parameter.get . . . 50
genome.parameter.set . . . 50
genome.segments . . . 51
genotype.population . . . 52
get.genome.par . . . 53
get.map . . . 53
get.mdp . . . 54
get.population . . . 54
get.population.gvalue . . . 55
get.population.info . . . 55
get.population.pvalue . . . 56
get.population.size . . . 57
get.score . . . 57
gs.build.esvs . . . 58
gs.build.tsps . . . 59
gs.build.V . . . 60
gs.build.Z . . . 60
gs.check.pvals . . . 61
gs.compare.effects . . . 61
gs.cross.eval.es . . . 62
gs.cross.eval.gd . . . 64
gs.cross.eval.gd.fct . . . 65
gs.cross.eval.ma . . . 65
gs.cross.eval.mi . . . 66
gs.cross.eval.mu . . . 67
gs.cross.eval.va . . . 68
gs.cross.info . . . 69
gs.cross.info.gd . . . 70
gs.cross.validation . . . 70
gs.esteff.lsq . . . 71
gs.esteff.rmla . . . 72
gs.esteff.rmlc . . . 73
gs.esteff.rmlr . . . 74
gs.esteff.rmlv . . . 74
gs.esteff.rr . . . 75
gs.estimate.gv . . . 77
gs.get.im . . . 77
gs.get.info.level . . . 78
gs.get.V . . . 78
gs.get.y . . . 79
gs.get.Z . . . 79

gs.info . . . 80
gs.lambda.aov . . . 80
gs.lambda.const . . . 81
gs.lambda.emstep . . . 81
gs.lambda.hsq . . . 82
gs.lambda.reg . . . 83
gs.lambda.rmla . . . 83
gs.lambda.rmla.02 . . . 84
gs.lambda.rmlc . . . 85
gs.lambda.rmlr . . . 86
gs.lambda.rmlv . . . 86
gs.lambda.rrblup . . . 87
gs.mme.coeff . . . 88
gs.mme.invcoeff . . . 88
gs.mme.restcoeff . . . 89
gs.mme.restrhs . . . 89
gs.mme.rhs . . . 90
gs.mme.solve . . . 90
gs.mmet.coeff . . . 91
gs.mmet.coeff.3 . . . 92
gs.mmet.rhs . . . 92
gs.mmet.solve . . . 93
gs.neg.effall . . . 93
gs.plot.effects . . . 94
gs.plot.model.fit . . . 95
gs.plot.validation . . . 96
gs.pos.effall . . . 96
gs.predict.genotypes . . . 97
gs.reset . . . 98
gs.restrict.marker.data.01 . . . 98
gs.return.effects . . . 99
gs.return.pvals . . . 99
gs.set.all.info.levels . . . 100
gs.set.allele.codes . . . 100
gs.set.effects . . . 101
gs.set.info.level . . . 102
gs.set.lambda . . . 102
gs.set.num.threads . . . 103
gs.set.performance.data . . . 103
gs.single.marker.aov . . . 104
gs.single.marker.reg . . . 104
gs.start.timer . . . 105
gs.stop.timer . . . 105
gs.test.mp . . . 106
gs.testeff.lsq . . . 106
gs.testeff.rmlc . . . 107
gs.testeff.rmlv . . . 107
gs.vc.rrblup . . . 108

gs.w.aov . . . 109
gs.write.pseff . . . 109
homozygote . . . 111
il.eval.library . . . 111
il.eval.lines . . . 112
il.ideal.library . . . 112
il.overlapping.library . . . 113
info . . . 114
info.cat . . . 114
init.population . . . 115
lib00.map1 . . . 116
lib00.map1a . . . 117
lib00.map2 . . . 117
linkage.drag . . . 118
linkage.map.get . . . 118
linkage.map.load . . . 119
linkage.map.save . . . 119
list.effects . . . 120
list.populations . . . 121
load.effmap . . . 121
load.internal.effmap . . . 122
load.linkage.map . . . 123
mab.compare . . . 124
mab.df . . . 130
mab.Examples . . . 131
mab.Input.files . . . 131
mab.load.data . . . 132
mab.save.inds . . . 133
mab.simulate . . . 134
mab.tabulate . . . 138
masbc.input . . . 139
Md_technow . . . 140
Mf_technow . . . 140
optimize.population . . . 141
ph.linkage.map.create . . . 141
phenotype.population . . . 142
plabsim . . . 143
plabsim.init . . . 143
plabsim.R.date . . . 144
plabsim.R.version . . . 144
plabsim.version . . . 144
plant . . . 145
population.append . . . 145
population.concat . . . 146
population.copy . . . 147
population.divide . . . 147
population.exist . . . 148
population.individual.remove . . . 149

population.info.get . . . 149
population.info.set . . . 150
population.list . . . 150
population.matrix.load . . . 151
population.matrix.save . . . 151
population.name.swap . . . 152
population.optimize . . . 152
population.plabsim.load . . . 153
population.plabsim.save . . . 153
population.remove . . . 154
population.remove.all . . . 154
population.rename . . . 155
population.resize . . . 155
population.sample . . . 156
population.size.get . . . 157
population.sort . . . 157
population.swap.name . . . 158
population.transfer . . . 159
read.vcf . . . 159
remove.all.populations . . . 160
remove.effmaps . . . 161
remove.evaluate.population . . . 161
remove.genotype.population . . . 162
remove.map . . . 163
remove.population . . . 163
rename.population . . . 164
reset.all . . . 164
reset.mdp . . . 165
resize.population . . . 165
resources . . . 166
return.population . . . 166
rng.choose . . . 167
rng.info . . . 167
rng.init . . . 168
rng.list . . . 169
sample.population . . . 169
save.linkage.map . . . 170
sdev . . . 171
sel.target.define . . . 171
select.all.best . . . 172
select.all.best.intern . . . 173
select.genotypes . . . 174
select.n.best . . . 175
select.n.best.segments . . . 176
set.co.freq . . . 177
set.eff.weight . . . 177
set.genome.par . . . 178
set.info.level . . . 179

set.mdp . . . 179
set.NoLociInit . . . 180
set.population.gvalue . . . 180
set.population.info . . . 181
show.phase . . . 182
single.cross . . . 182
sm.start.timer . . . 183
sm.stop.timer . . . 183
splitdt . . . 184
ssd.mating . . . 184
st.calc.ld . . . 185
st.calc.ld.2 . . . 187
st.calc.q . . . 188
st.calc.rf . . . 189
st.calculate.enrichment . . . 189
st.chrom.stats . . . 191
st.compact.data.set . . . 192
st.copy.marker.data . . . 193
st.data.set.list . . . 194
st.data.set.remove . . . 195
st.data.set.remove.all . . . 196
st.datadir . . . 197
st.dataframe.to.STvcf . . . 197
st.dd . . . 199
st.def.hblocks . . . 200
st.destroy.phase . . . 201
st.genetic.distances . . . 202
st.genetic.distances.02 . . . 204
st.genetic.distances.fct . . . 205
st.get.info.level . . . 206
st.get.map . . . 206
st.get.num.threads . . . 207
st.get.simpop . . . 207
st.get.simpop.perfdata . . . 208
st.get.simpop2 . . . 209
st.id . . . 209
st.indir . . . 210
st.info . . . 210
st.keep.intersection.markers . . . 211
st.LDheatmap . . . 211
st.LDplot.ld . . . 215
st.LDplot.map . . . 216
st.load.performance.data . . . 216
st.load.vcf.data . . . 217
st.mark.alleles . . . 220
st.marker.data.statistics . . . 220
st.markerdata.to.STvcf . . . 222
st.mxd . . . 224

st.od . . . 225
st.outdir . . . 225
st.phase . . . 226
st.phase.v1 . . . 229
st.plot.corr . . . 236
st.plot.corr.l . . . 237
st.plot.gene.diversity . . . 238
st.plot.ggt . . . 239
st.plot.ggt.src . . . 241
st.read.map . . . 242
st.read.map.df . . . 243
st.read.marker.data . . . 244
st.read.marker.data.df . . . 245
st.read.performance.data . . . 246
st.recode.hbc . . . 247
st.recode.hil . . . 248
st.recode.ref . . . 249
st.recode.ref.2 . . . 250
st.reset . . . 251
st.restrict.marker.data . . . 251
st.return.performance.data . . . 253
st.select.phen . . . 253
st.set.hblocks . . . 254
st.set.info.level . . . 255
st.set.matrix.ops . . . 255
st.set.num.threads . . . 256
st.set.openblas.threads . . . 256
st.set.sim.ef . . . 257
st.set.sim.gp . . . 259
st.set.sim.mp . . . 259
st.set.sim.pp . . . 260
st.set.simpop . . . 261
st.start.timer . . . 262
st.stop.timer . . . 263
st.STvcf.to.dataframe . . . 263
st.switch.error . . . 264
st.write.map . . . 266
st.write.marker.data . . . 267
st_mixed . . . 268
summarize.gvalue . . . 269
swap.population.name . . . 270
talk.to.me . . . 270
v-tropmaize-map . . . 271
v-tropmaize-phe . . . 271
v-tropmaize-pop . . . 272
v-tropmaize-vcf . . . 273
write.vcf . . . 274
write.version.2 . . . 275

Index **276**

`append.population` *append.population()*

Description

Appends a copy of one population to another.

Usage

```
append.population(NameP1, NameP2)
```

Arguments

`NameP1`	Character string naming the destination population. If it does not exist, it is created.
`NameP2`	Character string naming the source population. This population is not modified.

Details

The individuals of `NameP2` are copied to the end of `NameP1`. Existing individuals in `NameP1` are retained, and `NameP2` is not modified.

The source population must exist and contain at least one individual. The source and destination population names must differ.

Value

Returns NameP1 and NameP2. Usually called for side effects.

See Also

`population.append`, which is called by this wrapper.

Examples

```
# Define the genome and linkage map
map <- data.frame(
   chrom = 1,
   pos = 0.5,
   name = "form",
   class = "trait1"
)
define.genome(map)

# Create two populations with different alleles
init.population("P1", homozygote(1, NoInd = 3))
init.population("P2", homozygote(2, NoInd = 2))
```

```
# Combine copies of both populations in a new population
append.population("Combined", "P1")
append.population("Combined", "P2")

# Combined contains three individuals with genotype 1/1
# and two individuals with genotype 2/2.
# The original populations P1 and P2 are retained.
evaluate.genotype("Combined", "form")
population.list()
```

---

be.quiet *Switch SelectionTools output to a quiet mode*

Description

Switch SelectionTools output to a quiet mode.

Usage

```
be.quiet()
```

Value

Returns the same invisible implementation-level `list` as `set.info.level(0)`; the main effect is switching package information output to quiet mode.

---

concat.population *concat.population()*

Description

Concatenates two populations and removes the second population.

Usage

```
concat.population(NameP1, NameP2)
```

Arguments

NameP1	Recipient population
NameP2	Name of the population to be concatenated

Details

The individuals of `NameP2` are appended to `NameP1`. After successful concatenation, `NameP2` is removed. Complete individual records are moved or copied as required so chromosome data and any valid per-individual derived information remain associated with the correct individuals.

Value

Returns the same invisible implementation-level `list` as `population.concat()`. The function is a compatibility wrapper and the meaningful result is the delegated population operation.

`copy.population` *copy.population()*

Description

Copies all or a contiguous part of a simulation population into a destination population.

Usage

```
copy.population(NameP1, NameP2, start=1, n=-1)
```

Arguments

`NameP1`	Name of the population to be created
`NameP2`	Name of the original population
`start`	Index of the first copied individual
`n`	Number of individuals to be copied. Default: n=-1 means all individuals

Details

Copying starts at the one-based position `start`. A negative `n` requests the remaining individuals through the end of the source; a positive value requests that many individuals subject to the source bounds. The source population is not modified. Raw chromosome data are copied, and genotype or genetic-value data are copied only when the corresponding source category is complete for all active source individuals.

Value

Returns the same invisible implementation-level `list` as `population.copy()`. The function is a compatibility wrapper and the meaningful result is the delegated population operation.

cross *cross()*

Description

Generates a diploid progeny population by crossing individuals drawn from two parental populations.

Usage

```
cross(NamePg, NameP1, NameP2, NoPg = -1, self = 0, mode = 1,
      crossing.scheme = 0)
```

Arguments

NamePg	Character string naming the progeny population.
NameP1	Character string naming parental population 1.
NameP2	Character string naming parental population 2.
NoPg	Number of progeny to generate. A positive value creates that many individuals; -1 uses the allocation of an existing progeny population.
self	Probability of explicitly selfing a selected parent, between 0 and 1. With probability self/2, the selected individual from parental population 1 is selfed; with the same probability, the selected individual from parental population 2 is selfed. Otherwise, the two selected individuals are crossed. Thus, self=1 produces only selfed progeny.
mode	Integer controlling whether the same individual may be selected as both parents before explicit selfing is applied. With mode=0, the second parent is resampled if both selected parents are the same individual and the parental population contains more than one individual. With mode=1, this restriction is not applied.
crossing.scheme	Integer 0 to 3 selecting random or deterministic parent choice in the two parental populations.

Details

The simulation genome must be diploid. The progeny population must have a name different from both parental populations. Both parental population names may refer to the same population. Using the same population for both parents does not, by itself, ensure selfing. Existing contents under the progeny name are replaced when a new progeny population is constructed.

For positive NoPg, exactly that many progeny individuals are generated. The special value -1 uses the already allocated size of an existing progeny population; it is therefore meaningful only when such a destination population already exists with positive allocation.

The four crossing schemes control whether the parent chosen from each parental population is random or deterministic. Deterministic selection cycles through the active individuals of the corresponding parent population. For progeny number n (counted from 1), the one-based position of the deterministically selected parent in parental population P of size S_P is

$$i_P(n) = ((n-1) \bmod S_P) + 1.$$

Thus deterministic selection cycles through all active individuals and starts again with the first individual after the end of the parental population is reached. The rule is applied separately to each parental population, so the two populations may have different sizes. `self` controls selfing, and `mode` controls whether the mating logic applies self-incompatibility.

Each progeny receives newly simulated chromosome segments produced by meiosis. Cached marker genotypes and evaluated genetic values are not inherited as authoritative derived data; they can be generated from the new chromosomes when needed. If chromosome construction fails, the incomplete progeny population is removed rather than left partially generated.

Value

Invisibly returns a list containing the arguments passed to the compiled crossing routine. The function is primarily called for its side effect of creating the progeny population.

Note

Crossing scheme 0 selects both parents randomly; scheme 1 selects the first parent randomly and the second deterministically; scheme 2 does the reverse; scheme 3 selects both deterministically.

Examples

```
map <- data.frame(chrom = 1,
                  pos = 0.01,
                  name = "form")
define.genome(map)

init.population("P1", homozygote(1))
init.population("P2", homozygote(2))

# Example: Generate 100 F1 hybrids
cross("F1","P1", "P2", 100)
evaluate.genotype("P1","form")

# Example: Repeated selfing
cross("F2", "F1", "F1", 1000, self = 1)
cross("F3", "F2", "F2", 1000, self = 1)
cross("F4", "F3", "F3", 1000, self = 1)
evaluate.genotype("F4", "form")
```

data.params *Return or inspect parameters associated with a data object*

Description

Return or inspect parameters associated with a data object.

Usage

```
data.params(dta)
```

Arguments

dta	Input data object.

Value

A named list describing the supplied data object, with components:

colhead	Parsed column headings identifying populations and individuals.
rowhead	Parsed row headings identifying markers and alleles.
no.mar	Number of markers.
no.all	Number of alleles per marker.
no.pop	Number of populations.
no.ind	Numbers of individuals per population.
ind.list	Individual numbers within populations.
mar.list	Marker names.
pop.list	Population names.

define.effects *define.effects()*

Description

Defines effects that can be used for selection

Usage

```
define.effects(name, description)
```

Arguments

name	Name of the effect
description	Description of the effect

Details

The syntax for the `description` string is
Mean, Classname Allele [Allele] Distribution Parameter [,...]

Mean	The population mean
Classname	A class of Loci to which effects are assigned to
Allele	The allels to which the effects are assigned to
[Allele]	Optional. Is used to define dominance effects
[, ...]	An arbitrary number of effects can be defined separated by commas
Distribution	normal 0/0.01 or uniform 0

For `normal mean/variance`, the first number is the mean and the second number is the variance of the normal distribution from which an effect is drawn independently for every matching locus. Thus `normal 1/0.05` uses mean 1 and variance 0.05 (standard deviation $\sqrt{0.05}$). The variance must be nonnegative.

For `uniform value`, the supplied number is used as a fixed effect value at every matching locus; despite the historical keyword `uniform`, no random draw is made for this form.

If one allele is supplied, the specification defines an allele effect. If two alleles are supplied before the distribution keyword, the specification defines the effect of that allele combination and can be used for dominance effects. Several effect specifications can be included in one description, separated by commas.

Value

An integer status returned by the final `unlink()` call that removes the temporary effect-definition file. The meaningful result is the side effect of defining/loading the requested effects.

`define.effects.df` *define.effects.df()*

Description

Automatically generates effects for loci: effect is always 1 for allele 1.

Usage

```
define.effects.df ( LocNames )
```

Arguments

`LocNames`	Name of the loci for which standard effects of 1 are generated

Details

Population specified via PopName is accessed in the background and must therefore be available in the backend.

Value

NULL. The function is called for its side effect of defining effect maps for the locus names in the supplied data frame; invalid pre-existing definitions return NULL invisibly.

Examples

```
map <- data.frame(chrom = 1:2,
    pos = c(0.01,0.02),
    name = c("form","color"))
define.genome(map)

define.effects.df(map$name)
```

`define.genome` *Define a genome from a linkage map and optional chromosome lengths*

Description

Defines a diploid simulation genome and linkage map from a data frame of locus positions. The number of chromosomes is inferred from the map. Chromosome lengths can be supplied explicitly or inferred from the locus positions.

Usage

```
define.genome(map, length.vec=NULL)
```

Arguments

map	Linkage map or map object.
length.vec	Vector of chromosome lengths. If a chromosome length is zero, it is replaced by 0.01.

Details

Dataframe provided to the map function must have three columns and an optional fourth column. First four columns are interpreted as: *Chromosome, Position, Name, Class.* Column names do not need to be specified.

Chromosome Numeric chromosome identifier. Values must be whole numbers.

Position Numeric position of the locus in Morgans, greater than or equal to zero. When chromosome lengths are supplied explicitly, locus positions must lie within the corresponding chromosomes.

Name Character string identifying the locus. Locus names must be unique.

Class Optional character class label. Classes can be used to define effects for groups of loci. If this column is omitted, class names are generated by prefixing each locus name with "Class". For example, locus "form" receives class "Classform".

Calling this function replaces the simulation genome and linkage map. Existing simulation populations and effect definitions are removed.

Value

NULL on successful completion. The function is called for setting genome parameters and defining the linkage map in the backend.

See Also

This function is a wrapper for the more explicit [set.genome.par] and [define.map] that behave slightly differently.

Examples

```
map <- data.frame(chrom = 1:2,
   pos = c(0.01,0.02),
   name = c("form","color"))
define.genome(map)

map <- data.frame(chrom = 1:2,
   pos = c(0.01,0.02),
   name = c("form","color"),
   class = paste0("trait",1:2))
define.genome(map)

map <- data.frame(chrom = 1:2,
   pos = c(0.01,0.02),
   name = c("form","color"),
   class = paste0("trait",1:2))
define.genome(map, length.vec=c(0.1,0.1))
```

---

define.map *define.map()*

Description

Defines and installs a simulation linkage map from a compact textual map description.

Usage

```
define.map(description)
```

Arguments

description	A character string or character vector containing locus specifications. Multiple specifications can be separated by commas or whitespace within a string, or supplied as separate elements of a character vector.

Details

Genome parameters must be defined before calling this function, for example using `set.genome.par`. Each call replaces the currently defined linkage map.

```
[*Number] Locusname Classname Position
```

where `*Number` is optional. `Locusname` is the locus-name prefix, `Classname` is the locus class, and `Position` controls placement.

For a single explicitly positioned locus, `Position` is written as `chromosome/position`. The chromosome is one-based and the position must lie between 0 and the chromosome length defined for the simulation genome. The position is in the same unit as the simulation chromosome lengths (Morgan in the standard SelectionTools simulation setup). A multiplicity prefix is not allowed for explicitly positioned loci.

Generated positions can be specified as follows:

`random` Generate `Number` loci at independently random positions over the complete genome. Chromosomes are sampled in proportion to their lengths, so these positions are genuinely random rather than equidistant.

`terminal` Distribute `Number` loci equally among the chromosomes and place them equidistantly on each chromosome including the telomeres. `Number` must be a multiple of the number of chromosomes.

`nonterminal` Distribute `Number` loci equally among the chromosomes and place them equidistantly in the chromosome interiors, excluding the telomeres. `Number` must be a multiple of the number of chromosomes.

For generated positions a running integer suffix is added to the supplied `Locusname` prefix to obtain unique locus names.

The description is first converted to a temporary map file by `generate.map.file` and is then loaded through the simulation linkage-map loader. The generated map is validated for chromosome bounds, position bounds, unique locus names, and complete specifications, and is ordered by chromosome and position before loading. The operation replaces the currently active simulation linkage map definition only after the description has been accepted by the map-generation/loading steps.

Value

Invisibly returns a list whose `retval` component indicates success or failure. The function is primarily called for its side effect of defining the simulation linkage map.

Examples

```
set.genome.par(no.chrom = 1,
               no.hom = 2,
               chrom.len = 0.05)

# Define a single locus
# Format: Name Class Chromosome/Position
define.map("form trait1 1/0")
```

```
# Replace the map with two loci supplied in one string
define.map("form trait1 1/0, color trait2 1/0.02")

# Define the same map using a character vector
define.map(c("form trait1 1/0",
             "color trait2 1/0.02"))

# Inspect the resulting map
linkage.map.get()
```

---

deprecated *Issue information for a deprecated function or object name*

Description

Issue information for a deprecated function or object name.

Usage

```
deprecated(NameOld, NameNew)
```

Arguments

NameOld	Previous function or object name.
NameNew	Replacement function or object name.

Value

NULL. This helper has no result object.

---

dh *dh()*

Description

Generates a diploid population of doubled-haploid lines from a parental population.

Usage

```
dh(NamePg, NameP1, NoPg=-1)
```

Arguments

NamePg	Character string naming the doubled-haploid progeny population.
NameP1	Character string naming the parental population.
NoPg	Number of progeny. A positive value creates that many lines; -1 uses the allocation of an existing destination population.

Details

For each progeny, a parental individual is chosen randomly from the active individuals of the parental population and a haploid meiotic product is generated. The selected haploid chromosome set is then doubled so that both homologues of each progeny carry the same recombined haplotype.

The simulation genome must be diploid. For positive `NoPg`, that many doubled-haploid progeny are generated. The special value `-1` uses the allocation of an existing progeny population and therefore requires an existing destination with positive allocation.

The destination population is constructed from raw chromosome segments. Cached genotype and evaluation data can subsequently be recalculated. If meiosis or chromosome construction fails, the incomplete progeny population is removed.

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of creating doubled-haploid progeny.

`divide.population` *divide.population()*

Description

Splits individuals from one simulation population into a second population.

Usage

```
divide.population(NameP1, NameP2, NoI=-1)
```

Arguments

`NameP1`	Name of the population to be created
`NameP2`	Name of the population to be divided
`NoI`	Number of individuals to be separated from population *NameP2*

Details

The first requested individuals are separated from `NameP2` and placed in `NameP1`; the remaining individuals stay in `NameP2`. Complete individual structures are moved, so chromosome segments, information fields, genotype data, and evaluations that belong to an individual move together.

Value

Returns the same invisible implementation-level `list` as `population.divide()`. The function is a compatibility wrapper and the meaningful result is the delegated population operation.

DT_technow *Technow Hybrid Phenotype Data*

Description

Phenotypic data for a maize hybrid data set used for examples of genomic prediction in SelectionTools. The records identify dent and flint parents, their hybrid, and grain-yield performance.

Usage

```
data(DT_technow)
```

Format

A data frame containing the variables `dent`, `flint`, `hy`, and `GY`. The first two variables identify the parental lines, `hy` identifies the hybrid, and `GY` contains grain-yield phenotypes.

Source

Data supplied with SelectionTools for genomic prediction examples. Original data published in Technow et al. (2014): doi:10.1534/genetics.114.165860.

effect.weight.set.all *Set effect weights for all effects*

Description

Set effect weights for all effects.

Usage

```
effect.weight.set.all(weight)
```

Arguments

`weight`	Weight value.

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of setting all effect weights.

effmap.remove *Remove a stored effect map*

Description

Remove a stored effect map.

Usage

```
effmap.remove(Name)
```

Arguments

Name	Name identifying the object to be processed.

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of removing the requested effect map.

effmap.remove.all *Remove all stored effect maps*

Description

Remove all stored effect maps.

Usage

```
effmap.remove.all()
```

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of removing all effect maps.

evaluate.all.loci *Evaluate all loci in a population*

Description

Evaluate all loci in a population.

Usage

```
evaluate.all.loci(PopName,
            loci = paste(as.vector(unlist(linkage.map.get()[,3])), collapse=" "))
```

Arguments

PopName	Name of the population.
loci	Locus or loci to process.

Value

The same data.frame structure as evaluate.locus(), evaluated for all loci in the current map: rows are loci and columns are homozygote, heterozygote, and missing.

evaluate.allele *evaluate.allele()*

Description

Calculates allele counts at one or more loci in a simulation population.

Usage

```
evaluate.allele(PopName, Loci, missing=0)
```

Arguments

PopName	Character string naming the population to evaluate.
Loci	A character string containing locus names separated by spaces.
missing	Integer controlling missing-allele handling. With 0, missing alleles are included. With 1, loci without any completely observed genotype are removed, and missing allele copies at the remaining loci are excluded from the counts.

Details

The population and linkage map must already be defined. For multiple loci, each row represents an allele at one locus. The other locus columns contain zero placeholders.

Value

A `data.frame` containing locus columns and `count`. Returns `NULL` invisibly when no result is available.

See Also

`evaluate.allele2` accepts a character vector of locus names and excludes missing allele copies without first removing loci lacking completely observed genotypes. `evaluate.allele.freq` also calculates relative frequencies.

Examples

```
map <- data.frame(
   chrom = 1:2,
   pos = c(0.01, 0.02),
   name = c("form", "color")
)
define.genome(map)
init.population("P1", homozygote(1, NoInd = 2))

evaluate.allele("P1", "form color")
```

---

evaluate.allele.freq *Calculate or return allele frequencies for selected loci*

Description

Calculate or return allele frequencies for selected loci.

Usage

```
evaluate.allele.freq(PopName, Loci, missing=0)
```

Arguments

`PopName`	Name of the population.
`Loci`	A character string containing one locus name or multiple locus names separated by spaces. Character vectors of length greater than one are not supported.
`missing`	Integer controlling missing-allele handling. With `0`, missing alleles are included. With `1`, missing alleles are excluded.

Value

A `data.frame` containing the allele counts and relative frequencies.

See Also

For vector based input of loci, see `evaluate.allele.freq2`.

Examples

```
map <- data.frame(chrom = 1,
                  pos = 0.01,
                  name = "form")
define.genome(map)

init.population("P1", homozygote(1))
init.population("P2", homozygote(2))

# Example: Generate 100 F1 hybrids
cross("F1","P1", "P2", 100)
evaluate.genotype("P1","form")

# Example: Repeated selfing
cross("F2", "F1", "F1", 1000, self = 1)
cross("F3", "F2", "F2", 1000, self = 1)
cross("F4", "F3", "F3", 1000, self = 1)

evaluate.genotype("F4", "form")
evaluate.allele.freq("F4","form")
```

---

evaluate.allele.freq2 *evaluate.allele.freq2()*

Description

Calculates allele counts and relative frequencies at one or more loci in a simulation population.

Usage

```
evaluate.allele.freq2(PopName, Loci, missing=0)
```

Arguments

PopName	Character string naming the population to evaluate.
Loci	Character vector of locus names.
missing	Integer controlling missing-allele handling. With 0, missing alleles are included. With 1, missing allele copies are excluded from the counts and frequency calculation.

Details

The population and linkage map must already be defined. Each element of Loci is interpreted as one locus name. Space-separated names within a single string are not split.

For multiple loci, each row represents an allele at one locus; the other locus columns contain zero placeholders. Frequencies are calculated by dividing each count by the sum of all returned counts. They therefore sum to one across the complete table, not separately within each locus. To obtain frequencies within a locus, evaluate that locus separately.

Value

A `data.frame` containing locus columns, `count`, and `freq`.

See Also

`evaluate.allele.freq`, an alternative implementation that accepts space-separated locus names and differs in its handling of missing alleles.

Examples

```
map <- data.frame(
   chrom = 1:2,
   pos = c(0.01, 0.02),
   name = c("form", "color"),
   class = paste0("trait", 1:2)
)
define.genome(map)

init.population("P1", homozygote(1))
init.population("P2", homozygote(2))
cross("F1", "P1", "P2", 10)
cross("F2", "F1", "F1", 1000, self = 1)

# Allele frequencies at a single locus
evaluate.allele.freq2("F2", "form")

# Allele counts at both loci, with frequencies normalised
# across the complete table
evaluate.allele.freq2("F2", map$name)

# Evaluate each locus separately for within-locus frequencies
evaluate.allele.freq2("F2", "form")
evaluate.allele.freq2("F2", "color")
```

`evaluate.allele2` *evaluate.allele2()*

Description

Calculates allele counts at one or more loci in a simulation population.

Usage

```
evaluate.allele2(PopName, Loci, missing=0)
```

Arguments

PopName	Character string naming the population to evaluate.
Loci	Character vector of locus names. Each element specifies one locus.
missing	Integer controlling missing-allele handling. With 0, missing alleles are included. With 1, missing allele copies are excluded, while known copies are retained even if another allele of the same individual is missing.

Details

The population and linkage map must already be defined. For multiple loci, each row represents an allele at one locus. The other locus columns contain zero placeholders.

Value

A data.frame containing locus columns and count. Returns NULL invisibly when the request cannot be evaluated.

See Also

evaluate.allele accepts space-separated locus names. evaluate.allele.freq2 also calculates relative frequencies.

Examples

```
map <- data.frame(
   chrom = 1:2,
   pos = c(0.01, 0.02),
   name = c("form", "color")
)
define.genome(map)
init.population("P1", homozygote(1, NoInd = 2))

evaluate.allele2("P1", c("form", "color"))
```

---

evaluate.genome *evaluate.genome()*

Description

Computes the frequency distribution of an allele across the whole genome.

Usage

```
evaluate.genome(PopName, allele, chromosome=NULL, begin=NULL, end=NULL,
                verbose=FALSE)
```

Arguments

PopName	Name of the population to be evaluated
allele	Function argument.
chromosome	Chromosome
begin	Start of chromosome region to investigate
end	End of the chromosome region
verbose	Function argument.

Details

Analysis is done on the level of the chromosomes, independently of the loaded marker map.

Value

A data.frame with one row per individual. It always contains ratio, the evaluated allele proportion. With verbose = TRUE it additionally contains length, total, and noBlocks, as returned by the genome-evaluation routine.

Note

See the lib00.example1 for an application of the command.

---

evaluate.genotype *evaluate.genotype()*

Description

Calculates genotype frequencies

Usage

```
evaluate.genotype(PopName, Loci, alleles=0, mode=2, bg=0)
```

Arguments

PopName	Name of the population to be evaluated
Loci	A List of Loci
alleles	A list of different Alleles
mode	Considering gametic phase or not
bg	Integer controlling missing-allele handling. With 0, genotype combinations containing missing alleles are included. With 1, combinations containing the missing allele indicator are excluded and frequencies are recalculated among the remaining individuals.

Details

The population and linkage map must already be defined. Counts represent numbers of individuals, and frequencies are calculated relative to the total number of included individuals. Names must only consist of letters and numbers (no spaces, dots, etc.).

Value

A `data.frame` containing the genotype combinations and the count/frequency information returned by the compiled genotype evaluator. If a specific allele combination is requested the result can be reduced to the matching row. Returns `NULL` invisibly if no result is available.

Examples

```
# Simple functionalty:
map <- data.frame(chrom = 1,
                  pos = 0.0,
                  name = "form",
                  class = "trait1")
define.genome(map)

init.population("P1", homozygote(1))
init.population("P2", homozygote(2))

cross("F1","P1", "P2", 100)
cross("BC1","F1", "P2", 100)

evaluate.genotype("BC1","form")

# Complex functionality: Two linked loci
reset.all()

map <- data.frame(
  chrom = c(1, 1),
  pos = c(0.2, 0.8),
  name = c("A", "B")
)
define.genome(map, length.vec = 1)

# Individual 1 carries haplotypes 1-1 and 2-2.
# Individual 2 carries haplotypes 1-2 and 2-1.
pop <- data.frame(
  ind   = c(1, 1, 1, 1, 2, 2, 2, 2),
  chrom = c(1, 1, 1, 1, 1, 1, 1, 1),
  hom   = c(1, 1, 2, 2, 1, 1, 2, 2),
  pos = c(0.2, 0.8, 0.2, 0.8, 0.2, 0.8, 0.2, 0.8),
  all   = c(1, 2, 1, 2, 1, 2, 2, 1)
)
init.population("PhaseExample", pop)

# Preserve phase: the two individuals form separate categories,
# each with count 1 and frequency 0.5.
evaluate.genotype("PhaseExample", c("A", "B"), mode = 0)
```

```
# Ignore phase: both individuals are heterozygous at both loci,
# giving one category with count 2 and frequency 1.
evaluate.genotype("PhaseExample", c("A", "B"), mode = 2)

# Loci can also be supplied as a space-separated string.
evaluate.genotype("PhaseExample", "A B", mode = 2)

# Retrieve one observed phased combination.
evaluate.genotype("PhaseExample", c("A", "B"),
                  alleles = c(1, 2, 1, 2), mode = 0)
```

`evaluate.hap.calc` *Calculate haplotype information for a pair of loci*

Description

Calculate haplotype information for a pair of loci.

Usage

```
evaluate.hap.calc(locusA, locusB)
```

Arguments

`locusA`	First locus.
`locusB`	Second locus.

Value

A numeric vector of length five, in the order `noTI`, `noI`, `nodH`, `dLocA`, and `dLocB`, containing the values returned by the compiled pairwise haplotype calculation.

`evaluate.hap.calc.d` *Calculate a haplotype statistic for a pair of loci*

Description

Calculate a haplotype statistic for a pair of loci.

Usage

```
evaluate.hap.calc.d(locusA, locusB)
```

Arguments

locusA	First locus.
locusB	Second locus.

Value

An invisible numeric vector of length three, in the order dd, dp, and rr, containing the values returned by the compiled haplotype-statistic calculation.

---

evaluate.hap.free *Release or reset temporary haplotype-evaluation state*

Description

Release or reset temporary haplotype-evaluation state.

Usage

```
evaluate.hap.free()
```

Value

An invisible list returned by the underlying .C routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of releasing haplotype evaluation state.

---

evaluate.hap.init *Initialize haplotype evaluation for a population*

Description

Initialize haplotype evaluation for a population.

Usage

```
evaluate.hap.init(PopName, doubleHetero=0, missing=0)
```

Arguments

PopName	Name of the population.
doubleHetero	Control for handling double heterozygotes.
missing	Value or code used for missing data.

Value

An invisible list returned by the underlying .C routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of initializing haplotype evaluation state.

`evaluate.hap.return.genotype`
Return genotype results from the current haplotype evaluation

---

Description

Return genotype results from the current haplotype evaluation.

Usage

```
evaluate.hap.return.genotype()
```

Value

A `data.frame` containing the genotype table from the current initialized haplotype evaluation.

`evaluate.hap.return.table`
Return the table produced by the current haplotype evaluation

---

Description

Return the table produced by the current haplotype evaluation.

Usage

```
evaluate.hap.return.table()
```

Value

A `data.frame` containing the table from the current initialized haplotype evaluation.

evaluate.haplotype *Evaluate haplotypes for selected loci in a population*

Description

Evaluate haplotypes for selected loci in a population.

Usage

```
evaluate.haplotype(PopName, Loci, missing=0,
                   bg=plabsim$allele.missing.indicator)
```

Arguments

PopName	Name of the population.
Loci	Locus or loci to process.
missing	Value or code used for missing data.
bg	Background or missing-data code used by the evaluator.

Value

A data.frame containing the haplotype-evaluation table returned for the requested population and loci.

evaluate.ld *Evaluate linkage disequilibrium for a population and genomic position*

Description

Evaluate linkage disequilibrium for a population and genomic position.

Usage

```
evaluate.ld(PopName, allele, chrom, pos)
```

Arguments

PopName	Name of the population.
allele	Allele specification.
chrom	Chromosome identifier.
pos	Position on the chromosome.

Value

A one-column data.frame named length, containing the linkage-disequilibrium segment lengths returned for the requested population, allele and position.

evaluate.locus *Evaluate selected loci in a population*

Description

Evaluate selected loci in a population.

Usage

```
evaluate.locus(PopName, eLoci)
```

Arguments

PopName	Name of the population.
eLoci	Locus or loci to evaluate.

Value

A data.frame with one row for each requested locus and columns homozygote, heterozygote, and missing, containing the corresponding genotype counts.

evaluate.mdp *Evaluate marker-assisted donor-parent information for populations*

Description

Evaluate marker-assisted donor-parent information for populations.

Usage

```
evaluate.mdp(NamePg, NameP1, NameP2, effectfile=NULL)
```

Arguments

NamePg	Name of the population to evaluate.
NameP1	Name of the first parental population.
NameP2	Name of the second parental population.
effectfile	Effect file or effect specification.

Value

When a specific effectfile is supplied, the result is the invisible implementation-level list returned by the compiled MDP evaluator. When effectfile = NULL, the function evaluates all active effects for side effects and returns NULL.

`evaluate.population` *Evaluate genetic values for simulation populations*

Description

Evaluates genetic values for all active individuals in one or more simulation populations using loaded effect definitions.

Usage

```
evaluate.population(PopNames, effName=NULL)
```

Arguments

`PopNames`	Population name or population specification accepted by the simulation routines.
`effName`	Optional character string naming the effect used as the selection index.

Details

The population must have a complete genotype representation for all active individuals. Evaluation calculates the effect-specific genetic values for all loaded effects. Element 0 is first calculated as the weighted sum of these values. If `effName` is supplied, element 0 is then replaced by the genetic value of the named effect, preserving the existing selection behavior.

Evaluation is population-wide. Allocation failure clears the genetic-value category rather than leaving only some individuals evaluated.

Any stored phenotypic values for a population being re-evaluated are discarded before the new genetic values are calculated.

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of calculating/storing evaluation values for the requested populations.

See Also

`genotype.population`, `phenotype.population`, `get.population.gvalue`

gd.allele.frequencies *gd.allele.frequencies()*

Description

Calculates the allele frequencies of the populations in a data set consisting of codominant molecular marker data.

Usage

```
gd.allele.frequencies(dta)
```

Arguments

dta	Name of the dataset contaning the observed marker genotypes. The format of the dataset is illustrated by the datasets gd.dummy.populations and gd.maize.populations.

Value

A `data.frame` of allele frequencies, with marker/allele combinations in rows and populations in columns.

gd.allow.zero.frequencies *gd.allow.zero.frequencies()*

Description

If for a marker no allele was observed, this is a missing value if codominant markers are used. However, for dominant markers where each band is regarded as a marker with only one allele, markers where no allele are observed are not missing data.

Usage

```
gd.allow.zero.frequencies(allow=0)
```

Arguments

allow	Codominant marker data (SSR or RFLP) is analyzed like the datasets `gd.maize.lines` or `gd.maize.populations`. Dominant marker data (AFLP) is analyzed like the dataset `gd.maize.aflp`.

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of setting whether zero allele frequencies are allowed in the genetic-diversity routines.

gd.correct.missing *gd.correct.missing()*

Description

The function *gd.correct.missing* corrects missing marker data points in a data set. Two rules are applied: (1) If at least one allele was observed at a marker, missing values for this marker at other alleles are changed to zeros. (2) If at a marker no allele was observed but for all alleles only zeros are in the data set, the zeros are changed to missing data.

Usage

```
gd.correct.missing(dta)
```

Arguments

dta	Name of the dataset contaning the observed marker genotypes. The format of the dataset is illustrated by the datasets gd.dummy.populations and gd.maize.populations.

Value

A `data.frame` containing the corrected data, with the same row and column structure as the supplied data object.

gd.data.parameters *gd.data.parameters()*

Description

Determines the structure of a data set containing molecular marker data.

Usage

```
gd.data.parameters(dta)
```

Arguments

dta	Name of the dataset contaning the observed marker genotypes. The format of the dataset is illustrated by the datasets gd.dummy.populations and gd.maize.populations.

Value

A named `list` describing the genetic-diversity data object, with components `colhead`, `rowhead`, `no.mar`, `no.all`, `no.pop`, `no.ind`, `ind.list`, `mar.list`, and `pop.list`. These contain parsed column and row headings, marker/allele/population counts, individual indices, marker names, and population names, respectively.

gd.distance.similarity *Gd distance similarity*

Description

Calculate genetic distance or similarity measures, with optional bootstrap inputs.

Usage

```
gd.distance.similarity(dta, measure, par, boot=0, no.pop.u=0, pop.u=0,
                       no.ind.u=0, ind.u=0, no.mar.u=0, mar.u=0)
```

Arguments

dta	Input data object.
measure	Distance or similarity measure.
par	Parameters used by the selected method.
boot	Bootstrap control.
no.pop.u	Number of populations used for bootstrap sampling.
pop.u	Population bootstrap indices.
no.ind.u	Numbers of individuals used for bootstrap sampling.
ind.u	Individual bootstrap indices.
no.mar.u	Number of markers used for bootstrap sampling.
mar.u	Marker bootstrap indices.

Value

A data.frame with columns OTU1, OTU2, and Measure, containing pairwise population comparisons. When resampling-based uncertainty is requested an additional SDev column contains the corresponding standard deviations.

gd.dummy.populations *Molecular marker data for several dummy populations*

Description

This dataset illustrates how missing values are handled in the calculation of allele freuquencies.

Usage

```
data(gd.dummy.populations)
```

gd.genetic.distance *gd.genetic.distance()*

Description

Calculates pairwise genetic distances between all pairs of populations in the dataset.

Usage

```
gd.genetic.distance(dta, measure="euc", par="dist", boot=0, no.pop.u=0,
                    pop.u=0, no.ind.u=0, ind.u=0, no.mar.u=0, mar.u=0)
```

Arguments

dta	Name of the dataset containing the observed marker genotypes. The format of the dataset is illustrated by the datasets gd.dummy.populations and gd.maize.populations.
measure	"mrd" is the Modified Rogers Distance according to Wright (1978, p. 78). "rd" is the Rogers Distance (Rogers (1972). "euc" is the Euclidean Distance with respect to the allele frequencies, sometimes this distance measure is referred to as Gowers Distance. If measure is ommitted the Euclidean distance is computed.
par	Parameters to be calculated:

"dist"	Genetic distance measure
"sdev/jackm"	Jackkniving over markers
"sdev/bootm"	Bootstrapping over markers
"sdev/booti"	Bootstrapping over individuals
"sdev/bootmi"	Bootstrapping over markers and individuals

boot	Number of bootstrap runs for estimating the standard deviation of the estimates.
no.pop.u	Function argument.
pop.u	Function argument.
no.ind.u	Function argument.
ind.u	Function argument.
no.mar.u	Function argument.
mar.u	Function argument.

Value

A data.frame with columns OTU1, OTU2, and Measure, containing the requested pairwise genetic distances; an additional SDev column is included when resampling-based standard deviations are requested. Returns NULL invisibly on validation failure.

Note

1.) The Rogers distance and the Modified Rogers distance are standardized measures in the interval [0,1]. The standardization is achieved by the divisions by m and by 2 in the respective equations. This requests codominant markers! These measures can't be used with dominant markers. However the Euclidean distance is not standardized and can also be used with dominant marker data (such as the AFLP data in the dataset `gd.maize.aflp`).

2.) For homozygous genotypes Rogers' Distance (Rogers 1972) equals the Nei-Li Distance (Nei and Li 1979), which is the same as 1 minus the Dice coefficient (Dice 1945), further the Modified Rogers' Distance is the square root of the Rogers' Distance.

3.) For calculating pairwise genetic distances between individuals you can use populations consisting of one single individual each. The column names could be e.g., `p1.1 p2.1 p3.1 p4.1`

References

Dice, L.R. 1945. Measures of the amount of ecologic association between species. Ecology 26:197–302.

Nei, M. and W.H. Li. 1979. Mathematical Models for studying genetic variation in terms of restriction endonucleases. Proc. Natl. Acad. Sci. USA. 76:5268–5371.

Rogers, J.S. 1972. Measures of genetic similarity and genetic distance. VII. Univ. Tex. Publ. 7213:145–153.

Wright, S. 1978. Evolution and the Genetics of Populations. Volume 4: Variability Within and Among Natural Populations. University of Chicago Press, Chicago, Illinois.

`gd.list.irregular` *gd.list.irregular()*

Description

The function *gd.list.irregular* lists individuals for which at one marker more than two alleles were observed.

Usage

```
gd.list.irregular(dta1)
```

Arguments

`dta1`	Name of the dataset containing the observed marker genotypes. The format of the dataset is illustrated by the datasets gd.dummy.populations and gd.maize.populations.

Value

A `data.frame` with columns `ind` and `marker`, identifying individual/marker combinations for which more than two alleles were observed.

gd.list.missing *gd.list.missing()*

Description

The function *gd.list.missing* lists the missing marker data points in a data set containing molecular marker data.

Usage

```
gd.list.missing(dta)
```

Arguments

dta	Name of the dataset containing the observed marker genotypes. The format of the dataset is illustrated by the datasets gd.dummy.populations and gd.maize.populations.

Value

A `data.frame` with columns `ind` and `marker`, identifying individual/marker combinations with missing observations.

gd.maize.aflp *51 maize lines analyzed with 462 AFLP markers.*

Description

This data set consists of AFLP data from 51 maize lines. 462 markers are used. This is dominant data. Each band is a marker, which has only one allele. For analyzing this type of dominant marker use `gd.allow.zero.frequencies(1)`

Usage

```
data(gd.maize.aflp)
```

Format

The first five rows and columns look like:

	l1.1	l2.1	l3.1	l4.1	l5.1
E35M50+M001.1	1	1	1	0	1
E35M50+M002.1	0	0	0	0	1
E35M50+M003.1	0	1	0	0	0
E35M50+M004.1	1	1	1	0	1
E35M50+M005.1	0	0	1	0	1

Each column contains the marker genotype of an individual. The identifier for an individual (the column name) contains the name of the population to which the individual belongs and the name of the individual, separated by a dot. For band there is a row in the dataset, the rowname contains the name of the marker and seperated by a dot a 1, indicating that each marker has only one allele.

The data matrix consists of zeros and ones, denoting whether a certain band was observed at an individual or not. Missing data are denoted with NA.

For AFLPs the primer combination and the band is separated by a +.

See Also

gd.genetic.distance, gd.similarity.coefficient, gd.maize.populations, gd.maize.lines, gd.salad.aflp

---

gd.maize.lines *RFLP data from 50 maize lines*

Description

This data set consists of the RFLP data from 50 maize lines.

Usage

```
data(gd.maize.lines)
```

Format

The format is described for the data set gd.maize.populations.

See Also

gd.genetic.distance, gd.similarity.coefficient, gd.maize.populations, gd.maize.aflp, gd.salad.aflp

---

gd.maize.populations *SSR data from seven maize populations*

Description

This data set consists of the SSR marker data of seven tropical maize populations. From each populations 48 individuals were samples analyzed with 85 polymorphic Single Sequence Repeat markers.

Usage

```
data(gd.maize.populations)
```

Format

The first five rows and columns look like:

	POL24.01	POL24.02	POL24.03	POL24.04	POL25.01
nc130.139	0	0	1	1	NA
nc130.142	1	1	0	1	1
nc130.145	0	1	0	0	0
nc133.110	1	0	0	1	1
nc133.148	0	0	0	0	0

Each column contains the marker genotype of an individual. The identifier for an individual (the column name) contains the name of the population to which the individual belongs and the name of the individual, separated by a dot. For each allel of a marker there is a row in the dataset, the rowname contains the name of the marker and the name of the allele seperated by a dot.

The data matrix consists of zeros and ones, denoting whether a certain allale was observed at an individual or not. Missing data are denoted with NA.

See Also

gd.genetic.distance, gd.similarity.coefficient, gd.maize.lines, gd.maize.aflp, gd.salad.aflp

---

gd.mk.matr *Construct a matrix from genetic-distance results*

Description

Construct a matrix from genetic-distance results.

Usage

```
gd.mk.matr(dta, distance)
```

Arguments

dta	Input data object.
distance	Distance specification or matrix.

Value

A numeric matrix representation of the supplied pairwise distance results, with population labels taken from the data object.

gd.pcoa *gd.pcoa()*

Description

Calculates a principal coordinate analysis for marker data. It is a wrapper for the `cmdscale` function.

Usage

```
gd.pcoa(dta, k=3)
```

Arguments

`dta`	Name of the dataset containing the observed marker genotypes
`k`	Number of the principal coordinates that should be returned

Value

A numeric `matrix` of principal-coordinate scores returned by `cmdscale()`, with rows corresponding to populations and columns to the first `k` principal coordinates.

gd.salad.aflp *Jasminas 44 salads analyzed with 108 AFLP markers.*

Description

This data set consists of AFLP data from 44 salad lines. 108 markers are used. This is dominant data. Each band is a marker, which has only one allele. For analyzing this type of dominant marker use `gd.allow.zero.frequencies(1)`

Usage

```
data(gd.salad.aflp)
```

Format

The first five rows and columns look like:

	l34.1	l44.1	l45.1	l38.1	l33.1
p1+6.1	1	0	0	1	1
p2+1.1	1	1	1	1	1
p2+6.1	1	1	1	1	1
p2+7.1	1	1	1	1	1
p2+8.1	1	1	1	1	1

Each column contains the marker genotype of an individual. The identifier for an individual (the column name) contains the name of the population to which the individual belongs and the name of the individual, separated by a dot. For band there is a row in the dataset, the rowname contains the name of the marker and seperated by a dot a 1, indicating that each marker has only one allele.

The data matrix consists of zeroes and ones, denoting whether a certain band was observed at an individual or not. Missing data are denoted with NA.

For AFLPs the primer combination and the band is separated by a +.

See Also

gd.genetic.distance, gd.similarity.coefficient, gd.maize.populations, gd.maize.lines, gd.maize.aflp

---

`gd.similarity.coefficient` *gd.similarity.coefficient()*

Description

Calculates matching coefficients between individuals.

Usage

```
gd.similarity.coefficient(dta, measure="jac", par="dist", boot=0, no.pop.u=0,
                          pop.u=0, no.ind.u=0, ind.u=0, no.mar.u=0, mar.u=0,
                          resample.primers=FALSE)
```

Arguments

`dta`	Name of the dataset containing the observed marker genotypes. The format of the dataset is illustrated by the dataset gd.salad.aflp.
`measure`	`"dic"` is the Dice coefficient, `"jac"` the Jaccard similarity, and `"sma"` is the simple matching coefficient. Default is the Jaccard similarity.
`par`	Parameters to be calculated:

`"dist"`	Genetic similarity measure
`"sdev/jackm"`	Jackkniving over markers
`"sdev/bootm"`	Bootstrapping over markers

`boot`	Number of bootstrap runs for estimating the standard deviation of the estimates.
`no.pop.u`	Function argument.
`pop.u`	Function argument.
`no.ind.u`	Function argument.
`ind.u`	Function argument.

`no.mar.u` Function argument.

`mar.u` Function argument.

`resample.primers`
if set to `TRUE` Resampling is done over primers, instead of markers. Usefull for AFLP data. Primers must be separated in the dataset from the markers by a `+`.

Value

A `data.frame` with columns `OTU1`, `OTU2`, and `Measure`, containing the requested pairwise similarity coefficients; an additional `SDev` column is included when resampling-based standard deviations are requested. Returns `NULL` invisibly on validation failure.

Note

Each population should consist of one indivdiual only.

`gd.splitdot` *Split or transform marker labels used by genetic-diversity routines*

Description

Split or transform marker labels used by genetic-diversity routines.

Usage

```
gd.splitdot(m.a)
```

Arguments

`m.a` Marker or matrix input.

Value

A character `matrix` with two columns and one row for each input string. The first column contains the part before the first dot (or underscore if no dot is present), and the second column contains the remaining part; if no separator is present the second field is empty.

gd.status.zero.frequencies

Return status information concerning zero allele frequencies

Description

Return status information concerning zero allele frequencies.

Usage

```
gd.status.zero.frequencies()
```

Value

An integer scalar giving the current compiled status value for whether zero allele frequencies are allowed.

generate.effect.file *generate.effect.file()*

Description

Generates a file with effect descriptions that can be used for selection. Effects can be loaded. Additive and dominance effects can be

Usage

```
generate.effect.file(fName, description)
```

Arguments

fName	Name of the file
description	Description of the effect

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of generating the effect file.

Note

See `define.effects` for the syntax of the descripton and for an example

generate.map.file *generate.map.file()*

Description

Generates a simulation linkage-map text file from a compact map description without itself installing the map.

Usage

```
generate.map.file(fName, description)
```

Arguments

fName	The name of the map file that is generated
description	Description of the linkage map

Details

The description syntax is the same syntax consumed by `define.map`. The function validates the file name and description, normalizes a multi-element description to one comma-separated specification, and asks the simulation backend to write the map file.

The generated file can subsequently be loaded as a simulation linkage map. Generating the file does not by itself replace the currently loaded map.

Value

An invisible `list` whose `retval` component is the compiled map-generation status. Invalid map dimensions return `list(retval = -2L)` invisibly. The main effect is writing the generated map file.

generate.population *generate.population()*

Description

Generates one or more simulation populations from the marker-incidence matrix representation used by the genetic-distance routines.

Usage

```
generate.population(dta, backcross=FALSE)
```

Arguments

dta	Data frame or matrix in the marker-incidence representation expected by the genetic-distance population interface.
backcross	Logical flag selecting backcross-oriented generation in the backend.

Details

The function interprets the row and column descriptors of dta to determine marker names, allele codes, population names, and individual membership. Allele coding must be numeric integer coding compatible with the simulation routines.

The R layer derives the population and marker dimensions and passes the matrix together with allele and missing-value information to the C population generator. The resulting objects are simulation populations represented by chromosome data, not SelectionTools marker-data sets.

The backcross flag selects the corresponding population-generation mode in the simulation backend. Existing genome and linkage-map definitions determine how generated marker information is represented in simulated chromosomes.

Value

An invisible implementation-level list returned by either the population-generation routine or the information routine used to report invalid allele encoding. The meaningful result is the side effect of generating the requested population when the input is valid.

---

genome.contribution *genome.contribution()*

Description

Calculates the distribution of one allele in populations.

Usage

```
genome.contribution(pops, allele, chromosome=NULL, begin=NULL, end=NULL)
```

Arguments

pops	Names of the populations separated by blanks
allele	Allele which is considered
chromosome	Chromosome
begin	Start of chromosome region to investigate
end	End of the chromosome region

Details

Analysis is done on the level of the chromosomes, independently of the loaded marker map. Summarizes the results of evaluate.genome.

Value

A numeric `matrix` with one row per population and columns `Obs`, `Mean`, `SDev`, `Min`, `Q10`, `Med`, and `Max`, summarizing genome-contribution values.

Note

See the `lib00.example1` for an application of the command.

---

genome.parameter.get *Return the current genome parameters*

Description

Returns the chromosome structure currently installed in the simulation backend.

Usage

```
genome.parameter.get()
```

Details

The result describes the simulation-wide genome parameters used to interpret chromosome segments and linkage-map positions. It includes the number of chromosomes, the number of homologues, and the chromosome-length vector.

Value

A named `list` with components `no.chrom` (number of chromosomes), `no.hom` (number of homologues), and `chrom.len` (chromosome-length vector).

---

genome.parameter.set *Set genome parameters used by the simulation code*

Description

Defines the chromosome structure used by the simulation backend.

Usage

```
genome.parameter.set(no.chrom, no.hom=2, chrom.len)
```

Arguments

`no.chrom`	Positive number of chromosomes.
`no.hom`	Number of homologues; the default is two.
`chrom.len`	Numeric vector containing one chromosome length for each chromosome.

Details

The genome definition is simulation-wide and determines the chromosome and homologue structure expected by population initialization, meiosis, crossing, single-seed descent, doubled-haploid production, and linkage-map operations.

`chrom.len` supplies one length per chromosome. Operations that model ordinary diploid marker data use two homologues. Changing genome parameters should therefore be coordinated with the loaded simulation linkage map and existing simulation populations.

Value

An invisible implementation-level `list` returned by the compiled state-setting or information routine. The function is primarily called for its side effect of resetting populations/map as needed and setting the genome parameters.

`genome.segments` *Extract or summarize genome segments for populations*

Description

Extract or summarize genome segments for populations.

Usage

```
genome.segments(pops, allele, chromosome=NULL, begin=NULL, end=NULL)
```

Arguments

`pops`	Population name or names.
`allele`	Allele specification.
`chromosome`	Chromosome identifier.
`begin`	Beginning position of a region.
`end`	Ending position of a region.

Value

A numeric `matrix` with one row per population and columns `Obs`, `Mean`, `SDev`, `Min`, `Q10`, `Med`, and `Max`, summarizing the number of genome segments.

`genotype.population` *genotype.population()*

Description

Calculates marker genotypes for one or more simulation populations from their chromosome-segment representation and the currently loaded linkage map.

Usage

```
genotype.population(PopNames)
```

Arguments

`PopNames`	One population name, a space-separated character string of population names, or a character vector of population names to genotype.

Details

Simulation populations store chromosome segments as the primary genetic representation. This function projects those segments onto the loci of the currently loaded map and stores the two alleles for every active individual and mapped locus.

All active individuals of a population are treated as one genotype category: after a successful call every active individual has genotype data. If a population contains a mixture of genotyped and non-genotyped active individuals, the complete population is recalculated. Allocation failure clears the population genotype cache rather than leaving a partial mixture.

Population names may be supplied as a vector; the R wrapper combines multiple names into the space-separated form accepted by the C routine. Genotyping does not evaluate genetic values. Existing evaluations that depend on a genotype may need to be recalculated after changes to the underlying population or map.

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of generating/storing genotype information for the requested populations.

get.genome.par *get.genome.par()*

Description

Returns the current simulation genome parameters.

Usage

```
get.genome.par()
```

Details

The returned values describe the genome used by simulation populations: chromosome count, number of homologues, and chromosome lengths. These parameters are separate from a SelectionTools marker data set, although the marker-to-simulation bridge can derive and install compatible simulation genome parameters from a marker map.

Value

Returns the same named `list` as `genome.parameter.get()`, with components `no.chrom`, `no.hom`, and `chrom.len`.

get.map *get.map()*

Description

Returns the currently loaded simulation linkage map.

Usage

```
get.map()
```

Details

The result describes the simulation map points in their active access order, including chromosome, map position, locus name, and class. Retrieving the map does not modify simulation populations or marker-data sets.

Value

Returns the same `data.frame` as `linkage.map.get()`, with columns `chrom`, `pos`, `name`, and `class`.

get.mdp *Return the current marker-assisted donor-parent information*

Description

Return the current marker-assisted donor-parent information.

Usage

```
get.mdp()
```

Value

An integer scalar containing the current marker-assisted donor-parent (MDP) value returned by the compiled routine.

get.population *get.population()*

Description

Returns the raw chromosome-segment representation of a simulation population as an R data frame.

Usage

```
get.population(PopName)
```

Arguments

PopName	Character string naming the simulation population.

Details

Each row identifies an individual, chromosome, homologue, segment start position, and allele. These rows are the primary simulation representation from which marker genotypes are derived when `genotype.population` is run.

The returned data frame can be used to inspect or serialize a population and is the representation accepted by `init.population`. Retrieving the population does not alter genotype or evaluation caches.

Value

A `data.frame` with columns `ind`, `chrom`, `hom`, `pos`, and `all`, representing the active population in the package population format.

get.population.gvalue *Get genetic values from a simulation population*

Description

Returns stored evaluated genetic values for the active individuals of a simulation population.

Usage

```
get.population.gvalue(name, EffName=NULL)
```

Arguments

name	Character string naming the population.
EffName	Optional character string identifying the effect-specific values to retrieve.

Details

Genetic values are derived population data created by population evaluation or explicitly supplied through `set.population.gvalue`.

If `EffName` is omitted, element 0 of the genetic value array is returned. After ordinary population evaluation this is the weighted genetic selection index. If `EffName` is supplied, the corresponding effect-specific genetic value is returned.

Value

A one-column `data.frame` named `gvalue`, containing one genetic value per active individual in the requested population.

See Also

`evaluate.population`, `set.population.gvalue`, `get.population.pvalue`

get.population.info *Return stored information for individuals in a population*

Description

Return stored information for individuals in a population.

Usage

```
get.population.info(PopName, ind)
```

Arguments

PopName	Name of the population.
ind	Individual index or indices.

Value

Returns the same character value or NULL as population.info.get().

---

get.population.pvalue *Get phenotypic values from a simulation population*

Description

Returns stored phenotypic values for the active individuals of a simulation population.

Usage

```
get.population.pvalue(name, EffName=NULL)
```

Arguments

name	Character string naming the population.
EffName	Optional character string naming an effect.

Details

Phenotypic values must previously have been generated with phenotype.population. If EffName is omitted, element 0 of the phenotypic value array is returned. This is the weighted phenotypic selection index. If EffName is supplied, the corresponding effect-specific phenotypic value is returned.

Value

A one-column data.frame named pvalue, containing one phenotypic value per active individual in the requested population.

See Also

phenotype.population, get.population.gvalue

get.population.size *get.population.size()*

Description

Returns the active size and allocated capacity of a simulation population.

Usage

```
get.population.size(PopName)
```

Arguments

PopName	Character string containing one population name or several population names separated by blanks.

Details

The active size is the number of individuals currently belonging to a population. The allocated capacity is internal storage available to the population and can differ from the active size after resizing or other population operations.

Several population names can be supplied in one character string, separated by blanks. In that case, NoInds and NoIndsAlloc are summed over all specified populations.

Value

Returns the same one-row data.frame as population.size.get(), with columns NoInds and NoIndsAlloc. For several population names these columns contain the corresponding sums. Returns NULL if a requested population is unavailable.

get.score *get.score()*

Description

The function *get.score()* returns a field with the calculated genotypic index for every individual of the population *pop*. If an effect is specified then the value of this effect is returned.

Usage

```
get.score(pop, effectfile=NULL)
```

Arguments

pop	Name of the evaluated population.
effectfile	Name of effect which should be evaluated.

Value

A one-column `data.frame` named `gvalue`, containing the genetic values obtained after genotyping and evaluating the requested population.

`gs.build.esvs` *Build estimation and validation subsets from a base data set*

Description

Build estimation and validation subsets from a base data set.

Usage

```
gs.build.esvs(es.size, base.set = "default", estimation.set = "none",
              validation.set = "none", es_m = NULL, es_p = NULL,
              vs_m = NULL, vs_p = NULL, auxfiles = FALSE)
```

Arguments

`es.size`	Size of the estimation set.
`base.set`	Name of the base data set.
`estimation.set`	Name of the estimation data set.
`validation.set`	Name of the validation data set.
`es_m`	Marker-data output file for the estimation set. Required when `auxfiles` = `TRUE`.
`es_p`	Phenotype-data output file for the estimation set. Required when `auxfiles` = `TRUE`.
`vs_m`	Marker-data output file for the validation set. Required when `auxfiles` = `TRUE`.
`vs_p`	Phenotype-data output file for the validation set. Required when `auxfiles` = `TRUE`.
`auxfiles`	Logical. If `TRUE`, the four persistent output files are written and `es_m`, `es_p`, `vs_m`, and `vs_p` must all be supplied.

Value

On a successful run, an invisible implementation-level `list` returned by the final timing/information helper; the meaningful result is the side effect of constructing the estimation and validation subsets. Validation failures may return `NULL`.

gs.build.tsps *Build training and prediction subsets from a base data set*

Description

Build training and prediction subsets from a base data set.

Usage

```
gs.build.tsps(ts.size, base.set = "default", training.set = "none",
              prediction.set = "none", es_m = NULL, es_p = NULL,
              vs_m = NULL, vs_p = NULL, auxfiles = FALSE)
```

Arguments

ts.size	Size of the training set.
base.set	Name of the base data set.
training.set	Name of the training data set.
prediction.set	Name of the prediction data set.
es_m	Marker-data output file for the training set. Required when auxfiles = TRUE.
es_p	Phenotype-data output file for the training set. Required when auxfiles = TRUE.
vs_m	Marker-data output file for the prediction set. Required when auxfiles = TRUE.
vs_p	Phenotype-data output file for the prediction set. Required when auxfiles = TRUE.
auxfiles	Logical. If TRUE, the four persistent output files are written and es_m, es_p, vs_m, and vs_p must all be supplied.

Value

On a successful run, an invisible implementation-level list returned by the final timing/information helper; the meaningful result is the side effect of constructing the training and prediction subsets. Validation failures may return NULL.

gs.build.V — *Build the genomic-selection variance or covariance matrix for a data set*

Description

Build the genomic-selection variance or covariance matrix for a data set.

Usage

```
gs.build.V(out.filename = "V.matrix", auxfiles = FALSE, data.set = "default")
```

Arguments

out.filename	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
auxfiles	Logical. If TRUE, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
data.set	Name of the SelectionTools data set.

Value

Invisibly returns a numeric matrix containing the generated variance/covariance matrix V when auxfiles = TRUE and the compiled calculation succeeds; otherwise NULL.

gs.build.Z — *gs.build.Z()*

Description

The function *gs.build.Z* builds the design (Z) matrix out of the marker data for genomic selection.

Usage

```
gs.build.Z(out.filename = "Z.matrix", auxfiles = FALSE, data.set = "default")
```

Arguments

out.filename	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
auxfiles	Logical. If TRUE, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
data.set	character. Name of the data set that is used by the function

Details

By default, genotypes are coded as `0`, `1`, or `2`, representing the number of copies of the effect allele. When effect alleles have not already been defined, the larger observed allele code at each marker is used as the effect allele. For biallelic STvcf data with both REF and ALT observed, this corresponds to `0` for REF/REF, `1` for REF/ALT, and `2` for ALT/ALT. Missing genotypes receive the same value as heterozygotes (`1` by default), so these cannot be distinguished in the resulting matrix.

Value

Invisibly returns a `matrix` containing the generated marker/design matrix `Z` when `auxfiles = TRUE` and generation succeeds; otherwise `NULL`.

---

gs.check.pvals *Check whether p-values are available for a genomic-selection data set*

Description

Check whether p-values are available for a genomic-selection data set.

Usage

```
gs.check.pvals(data.set ="default")
```

Arguments

`data.set`	Name of the SelectionTools data set.

Value

A logical scalar indicating whether p-values are available for the requested genomic-selection data set.

---

gs.compare.effects *gs.compare.effects()*

Description

Creates a scatter plot of effects of two different models to visualize differences in estimated effects.

Usage

```
gs.compare.effects(data.set.a, data.set.b, label.a = data.set.a,
                   label.b = data.set.b)
```

Arguments

data.set.a	character. Name of the first data set containing marker effects. These effects are plotted on the x axis.
data.set.b	character. Name of the second data set containing marker effects. These effects are plotted on the y axis.
label.a	character. Label of the x axis.
label.b	character. Label of the y axis.

Value

No useful return value; the function is called for its side effect of comparing and plotting marker-effect results. Validation failures return NULL invisibly.

---

gs.cross.eval.es *Evaluate Crosses by Expected Superior Progeny Value*

Description

Calculates the expected superior progeny value for every evaluated cross by combining its expected mean and segregation variance.

Usage

```
gs.cross.eval.es(data.set = "default", alpha = 0.1, N = 0, G = 0)
```

Arguments

data.set	Character string naming the SelectionTools data set. Cross means and segregation variances must already have been evaluated.
alpha	Selected fraction used to calculate the standardized selection differential when N = 0 or G = 0.
N	Number of selected progeny for the finite-population correction. When both N and G are nonzero, alpha is replaced by N/G.
G	Number of evaluated progeny for the finite-population correction. When both N and G are nonzero, alpha is replaced by N/G.

Details

The expected superior progeny criterion combines the expected cross mean with the amount of segregation expected among the progeny. For a selected fraction α, the criterion described by Zhong and Jannink (2007) is

$$\mathrm{es} = \mu + i(\alpha)\,\sigma_g$$

where μ is the expected cross mean and σ_g is the square root of the segregation variance. The standardized selection differential is

$$i(\alpha) = \frac{g_{0,1}\left[G_{0,1}^{-1}(1-\alpha)\right]}{\alpha}$$

where $g_{0,1}$ is the probability density function of the standard normal distribution and $G_{0,1}^{-1}$ is the inverse of its cumulative distribution function.

For a small finite progeny population, when both `N` and `G` are nonzero, the function sets $\alpha = N/G$ and applies the correction of Burrows (1972):

$$\mathrm{es} = \mu + i(N, G)\,\sigma_g$$

with

$$i(N, G) = i(\alpha) - \frac{G - N}{2N(G+1)i(\alpha)}$$

The calculated value is stored as the cross criterion `es`. Results for all evaluated crosses can subsequently be obtained with `gs.cross.info`.

Value

On a successful run, an invisible implementation-level `list` returned by the final timing/information helper; the meaningful result is the side effect of evaluating all crosses by expected superior progeny value. Validation failures may return `NULL`.

References

Zhong S, Jannink JL (2007) Using quantitative trait loci results to discriminate among crosses on the basis of their progeny mean and variance. Genetics 177:567–576.

Burrows RM (1972) Expected selection differentials for directional selection. Biometrics 28:1091–1100.

See Also

`gs.cross.eval.mu`, `gs.cross.eval.va`, `gs.cross.info`

`gs.cross.eval.gd` *Evaluate Crosses by Genetic Distance*

Description

Calculates a genetic distance for every possible pair of individuals in a SelectionTools data set and stores it as a cross-evaluation criterion.

Usage

```
gs.cross.eval.gd(dist = "rd", data.set = "default")
```

Arguments

`dist`	Character string specifying the genetic-distance measure. SelectionTools uses `"rd"` for Rogers distance, `"mrd"` for Modified Rogers distance, and `"euc"` for Euclidean distance.
`data.set`	Character string naming the SelectionTools data set.

Details

Genetic distance can be evaluated independently of estimated marker effects. The result is stored as the cross criterion `gd` and can subsequently be obtained together with the other cross criteria using `gs.cross.info`.

The definitions and formulas for Euclidean distance, Rogers distance, and Modified Rogers distance are documented in `st.genetic.distances`. The properties of these and related similarity and dissimilarity coefficients in plant breeding are discussed by Reif et al. (2005).

Value

On a successful run, an invisible implementation-level `list` returned by the final timing/information helper; the meaningful result is the side effect of evaluating all crosses by genetic distance. Validation failures may return `NULL`.

References

Reif JC, Melchinger AE, Frisch M (2005) Genetical and mathematical properties of similarity and dissimilarity coefficients applied in plant breeding and seed bank management. Crop Science 45:1–7.

See Also

`st.genetic.distances`, `gs.cross.eval.mu`, `gs.cross.info`

gs.cross.eval.gd.fct *Evaluate crosses using genetic-distance function*

Description

Evaluate crosses using genetic-distance function.

Usage

```
gs.cross.eval.gd.fct(dist = "rd", split = 1, data.set = "default")
```

Arguments

dist	Genetic-distance measure.
split	Split or partition control.
data.set	Name of the SelectionTools data set.

Value

On a successful run, an invisible implementation-level list returned by the final timing/information helper; the meaningful result is the side effect of evaluating factorial crosses by genetic distance. Validation failures may return NULL.

gs.cross.eval.ma *Evaluate Crosses by Maximum Marker-Effect Value*

Description

Calculates the maximum marker-effect based performance that can be assembled from the parental alleles for every possible pair of parents in a SelectionTools data set.

Usage

```
gs.cross.eval.ma(data.set = "default")
```

Arguments

data.set	Character string naming the SelectionTools data set. The data set must contain estimated marker effects.

Details

Using the notation of `gs.cross.eval.mu`, the maximum value for a line derived from a cross is

$$\mathrm{ma} = \beta_0 + \sum_m \max(\beta_{mi}, \beta_{mj})$$

where β_0 is the intercept and β_{mi} and β_{mj} are the marker-effect contributions of the alleles carried by P1 and P2 at marker m.

The calculated value is stored as the cross criterion `ma`. It is a marker-effect based upper limit obtained by combining, marker by marker, the larger of the two parental contributions. Results for all evaluated crosses can subsequently be obtained with `gs.cross.info`.

Value

On a successful run, an invisible implementation-level `list` returned by the final timing/information helper; the meaningful result is the side effect of evaluating all crosses by the maximum marker-effect value. Validation failures may return `NULL`.

See Also

`gs.cross.eval.mu`, `gs.cross.eval.mi`, `gs.cross.info`

`gs.cross.eval.mi` *Evaluate Crosses by Minimum Marker-Effect Value*

Description

Calculates the minimum marker-effect based performance that can be assembled from the parental alleles for every possible pair of parents in a SelectionTools data set.

Usage

```
gs.cross.eval.mi(data.set = "default")
```

Arguments

`data.set`	Character string naming the SelectionTools data set. The data set must contain estimated marker effects.

Details

Using the notation of `gs.cross.eval.mu`, the minimum value for a line derived from a cross is

$$\mathrm{mi} = \beta_0 + \sum_m \min(\beta_{mi}, \beta_{mj})$$

where β_0 is the intercept and β_{mi} and β_{mj} are the marker-effect contributions of the alleles carried by P1 and P2 at marker m.

The calculated value is stored as the cross criterion `mi`. It is a marker-effect based lower limit obtained by combining, marker by marker, the smaller of the two parental contributions. Results for all evaluated crosses can subsequently be obtained with `gs.cross.info`.

Value

On a successful run, an invisible implementation-level `list` returned by the final timing/information helper; the meaningful result is the side effect of evaluating all crosses by the minimum marker-effect value. Validation failures may return `NULL`.

See Also

`gs.cross.eval.mu`, `gs.cross.eval.ma`, `gs.cross.info`

`gs.cross.eval.mu` *Evaluate Crosses by Expected Progeny Mean*

Description

Calculates the expected mean performance of the progeny for every possible pair of parents in a SelectionTools data set from previously estimated marker effects.

Usage

```
gs.cross.eval.mu(data.set = "default")
```

Arguments

`data.set`	Character string naming the SelectionTools data set. The data set must contain estimated marker effects.

Details

For a cross between parents P1 and P2, the expected progeny mean is calculated as

$$\mu = \beta_0 + \sum_m (\beta_{mi} + \beta_{mj})/2$$

where β_0 is the intercept, β_{mi} is the effect of allele i carried by P1 at marker m, and β_{mj} is the effect of allele j carried by P2 at marker m.

The calculated value is stored as the cross criterion `mu`. Results for all evaluated crosses can subsequently be obtained with `gs.cross.info`.

Value

On a successful run, an invisible implementation-level `list` returned by the final timing/information helper; the meaningful result is the side effect of evaluating all crosses by expected progeny mean. Validation failures may return `NULL`.

See Also

gs.cross.eval.mi, gs.cross.eval.ma, gs.cross.eval.va, gs.cross.eval.es, gs.cross.info

---

gs.cross.eval.va *Evaluate Crosses by Segregation Variance*

Description

Calculates the expected segregation variance for every possible pair of parents in a SelectionTools data set from previously estimated marker effects.

Usage

```
gs.cross.eval.va(data.set = "default", pop.type = "unlinked", t = 0,
                 map.function = "Haldane")
```

Arguments

data.set	Character string naming the SelectionTools data set. The data set must contain estimated marker effects.
pop.type	Character string specifying the progeny population type. The implementation accepts "unlinked", "DH", "SSD", and "CRS".
t	Integer generation or method-specific time parameter used for linked population types.
map.function	Character string specifying the map function used to obtain recombination frequencies when linkage is taken into account. The default is "Haldane".

Details

The function estimates the segregation variance σ_g^2 of each cross from the marker effects. The segregation variance describes how strongly progeny from a cross are expected to differ genetically.

For pop.type = "DH" or "SSD", linkage and recombination are taken into account using the genetic map; this is the cross-variance calculation described by Osthushenrich et al. (2017). With pop.type = "unlinked", the calculation treats the loci as unlinked. The current implementation also accepts "CRS" as a linked population type.

The calculated variance is stored as the cross criterion va. Its square root, σ_g, is used by gs.cross.eval.es when the expected superior progeny criterion is calculated. Results for all evaluated crosses can subsequently be obtained with gs.cross.info.

Value

On a successful run, an invisible implementation-level list returned by the final timing/information helper; the meaningful result is the side effect of evaluating all crosses by segregation variance. Validation failures may return NULL.

References

Osthushenrich T, Frisch M, Herzog E (2017) Genomic selection of crossing partners on basis of the expected mean and variance of their derived lines. PLoS ONE 12(12):e0188839.

See Also

`gs.cross.eval.mu`, `gs.cross.eval.es`, `gs.cross.info`

`gs.cross.info` *Return or write information for evaluated crosses*

Description

Return or write information for evaluated crosses.

Usage

```
gs.cross.info(bestn = 0, sortby = "index", out.filename = "cross.info",
              data.set = "default")
```

Arguments

`bestn`	Number of best entries to retain. A value of 0 returns all evaluated crosses.
`sortby`	Criterion used for sorting.
`out.filename`	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
`data.set`	Name of the SelectionTools data set.

Value

Invisibly returns a `data.frame` containing the evaluated-cross information when the compiled operation succeeds; otherwise `NULL`. Columns `P1Name` and `P2Name` are character strings, so numeric-looking individual identifiers are retained unchanged.

gs.cross.info.gd *Return or write genetic-distance information for crosses*

Description

Return or write genetic-distance information for crosses.

Usage

```
gs.cross.info.gd(out.filename ="cross.info.gd", data.set ="default")
```

Arguments

out.filename	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
data.set	Name of the SelectionTools data set.

Value

Invisibly returns a data.frame containing the genetic-distance cross information read from the generated result when the compiled operation succeeds; otherwise NULL.

gs.cross.validation *gs.cross.validation()*

Description

Splits a given data set into estimation and prediction set. Estimates marker effects with the specified model using the estimation set and predicts direct genomic values (DGV) for the built prediction set.

Usage

```
gs.cross.validation(estimation.method, n.ts, n.runs, hsq = 0.8, alpha = 1,
                    maxiter = 100, precision = 0.0001,
                    out.filename.r = "crosval", out.filename.u = "effects",
                    auxfiles = TRUE, data.set = "default")
```

Arguments

estimation.method	character. Specifies the model: "rmlc", "rrwe", "rmlv", "rmlr", "rrwr" "bayes"
n.ts	Number of individuals in the training set
n.runs	Number of cross validation runs
hsq	Heritability of the analyzed trait. Used for "rrwe", "rrwr"
alpha	Function argument.
maxiter	Maximum number of iterations used for convergence of the REML estimator. Used if scheme "rmlc" or "rmlv"
precision	Gives the precision as second stop value for iteration next to maxiter. If the difference of the values of lambda in two subsequent iterations is less than the given precision, convergence is reached and the iteration will stop. Used if scheme "rmlc" or "rmlv"
out.filename.r	Character string retained for compatibility and used as part of the name of the internal session-temporary file used for cross-validation correlations.
out.filename.u	Character string retained for compatibility and used as part of the name of the internal session-temporary file used for estimated marker effects.
auxfiles	Logical. If TRUE, both cross-validation result files are created in the R session temporary directory, read back by the R wrapper, and removed before the function returns.
data.set	character. Name of the data set that is used by the function.

Value

Invisibly returns a list when auxfiles = TRUE and cross-validation succeeds. Component cor is a numeric matrix of cross-validation correlations and component eff is a numeric matrix of estimated effects. Otherwise NULL.

---

gs.esteff.lsq *Estimate marker effects using the LSQ method*

Description

Estimate marker effects using the LSQ method.

Usage

```
gs.esteff.lsq(out.filename = "eff.lsq", auxfiles = TRUE, data.set ="default")
```

Arguments

out.filename	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
auxfiles	Logical. If TRUE, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
data.set	Name of the SelectionTools data set.

Value

Invisibly returns a numeric matrix containing the least-squares marker-effect estimates when auxfiles = TRUE and the compiled calculation succeeds; otherwise NULL.

---

gs.esteff.rmla *Estimate marker effects using the RMLA method*

Description

Estimate marker effects using the RMLA method.

Usage

```
gs.esteff.rmla(alpha = 1, maxiter = 1000, precision = 0.0001, hsq = 0.9,
               out.filename = "eff.rmla", auxfiles = TRUE,
               data.set ="default")
```

Arguments

alpha	Method-specific tuning or significance parameter.
maxiter	Maximum number of iterations.
precision	Convergence tolerance.
hsq	Heritability parameter.
out.filename	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
auxfiles	Logical. If TRUE, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
data.set	Name of the SelectionTools data set.

Value

Invisibly returns a numeric matrix containing the RMLA marker-effect estimates when auxfiles = TRUE and the compiled calculation succeeds; otherwise NULL.

`gs.esteff.rmlc` *gs.esteff.rmlc()*

Description

Estimates marker effects using BLUP with constant variances for each marker.

Usage

```
gs.esteff.rmlc(maxiter = 1000, precision = 0.0001, hsq = 0.9,
               out.filename = "eff.rmlc", auxfiles = FALSE,
               data.set ="default")
```

Arguments

`maxiter`	Maximum number of iterations used for convergence of the REML estimator
`precision`	Gives the precision as second stop value for iteration next to maxiter. If the difference of the values of lambda in two subsequent iterations is less than the given precision, convergence is reached and the iteration will stop
`hsq`	Function argument.
`out.filename`	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
`auxfiles`	Logical. If `TRUE`, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
`data.set`	character. Name of the data set that is used by the function

Details

The restricted maximum likelihood (REML) method is used to estimate variance components. In a next step, lambda is calculated using the formula:

$$\lambda = \frac{\sigma_e^2}{\frac{\sigma_g^2}{m}}$$

where σ_e^2 is the residual error variance, σ_g^2 is the genetic variance and m is the number of markers.

Value

Invisibly returns a numeric `matrix` containing the RMLC marker-effect estimates when `auxfiles = TRUE` and the compiled calculation succeeds; otherwise `NULL`.

gs.esteff.rmlr *Estimate marker effects using the RMLR method*

Description

Estimate marker effects using the RMLR method.

Usage

```
gs.esteff.rmlr(maxiter = 1000, precision = 0.001, hsq = 0.9,
               out.filename = "eff.rmlr", auxfiles = TRUE,
               data.set ="default")
```

Arguments

maxiter	Maximum number of iterations.
precision	Convergence tolerance.
hsq	Heritability parameter.
out.filename	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
auxfiles	Logical. If TRUE, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
data.set	Name of the SelectionTools data set.

Value

Invisibly returns a numeric matrix containing the RMLR marker-effect estimates when auxfiles = TRUE and the compiled calculation succeeds; otherwise NULL.

gs.esteff.rmlv *gs.esteff.rmlv()*

Description

Estimates marker effects using BLUP with variable variances for the markers.

Usage

```
gs.esteff.rmlv(maxiter = 1000, precision = 0.001, hsq = 0.9,
               out.filename = "eff.rmlv", auxfiles = FALSE,
               data.set ="default")
```

Arguments

maxiter	Maximum number of iterations used for convergence of the REML estimator.
precision	Gives the precision as second stop value for iteration next to maxiter. If the difference of the values of lambda in two subsequent iterations is less than the given precision, convergence is reached and the iteration will stop.
hsq	Function argument.
out.filename	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
auxfiles	Logical. If TRUE, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
data.set	character. Name of the data set that is used by the function.

Details

The restricted maximum likelihood (REML) method is used to estimate variance components.

Value

Invisibly returns a numeric matrix containing the RMLV marker-effect estimates when auxfiles = TRUE and the compiled calculation succeeds; otherwise NULL.

---

gs.esteff.rr *Estimate Marker Effects by Ridge Regression*

Description

Estimates genome-wide marker effects by ridge-regression methods, including RR-BLUP.

Usage

```
gs.esteff.rr(method = "BLUP", maxiter = 1000, precision = 0.0001, hsq = 0.80,
             alpha = 1, out.filename = "eff.rr", auxfiles = FALSE,
             data.set = "default")
```

Arguments

method	Character string selecting the ridge-regression method. The default "BLUP" performs RR-BLUP.
maxiter	Maximum number of iterations used for variance-component estimation where applicable. For method="BLUP", setting maxiter=0 skips variance-component iteration and uses the variance components implied by hsq directly.
precision	Convergence tolerance for iterative variance-component estimation.

hsq	Heritability used to construct starting variance components. For method="BLUP" with maxiter>0, these values initialize the variance-component iteration. With maxiter=0, they are used without further variance-component estimation.
alpha	Method-specific tuning or significance parameter.
out.filename	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
auxfiles	Logical. If TRUE, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
data.set	Name of the SelectionTools data set containing marker and performance data.

Details

With method="BLUP", SelectionTools fits the standard ridge-regression BLUP model. The implementation derives starting genetic and residual variance components from the observed performance variance and hsq. If maxiter>0, these starting values are refined by the mixed-model engine until the requested precision is reached or maxiter iterations have been performed.

Setting maxiter=0 omits this variance-component iteration. In that case the hsq-derived variance components are retained directly. This provides the computationally faster fixed-heritability form used in the SelectionTools simulation vignette.

After effect estimation, the marker effects are stored in data.set and can be used for genomic prediction, cross evaluation, or transferred to the simulation backend. The model fit for the training population can be inspected with gs.plot.model.fit.

Value

Invisibly returns a numeric matrix containing the estimated marker effects from ridge regression when auxfiles = TRUE and the compiled calculation succeeds; otherwise NULL. The estimated effects are also stored in data.set.

References

Whittaker JC, Thompson R, Denham MC (2000) Marker-assisted selection using ridge regression. Genetics Research 75:249–252.

Meuwissen THE, Hayes BJ, Goddard ME (2001) Prediction of total genetic value using genome-wide dense marker maps. Genetics 157:1819–1829.

Habier D, Fernando RL, Dekkers JCM (2007) The impact of genetic relationship information on genome-assisted breeding values. Genetics 177:2389–2397.

Hofheinz N, Frisch M (2014) Heteroscedastic ridge regression approaches for genome-wide prediction with a focus on computational efficiency and accurate effect estimation. G3 4:539–546.

See Also

gs.plot.model.fit, gs.predict.genotypes, gs.return.effects

gs.estimate.gv *Estimate genomic values for an estimation or prediction data set*

Description

Estimate genomic values for an estimation or prediction data set.

Usage

```
gs.estimate.gv(estimation.set ="default", prediction.set ="default",
               out.filename = "breeding.values", auxfiles = TRUE)
```

Arguments

estimation.set	Name of the estimation data set.
prediction.set	Name of the prediction data set.
out.filename	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
auxfiles	Logical. If TRUE, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.

Value

Invisibly returns a data.frame with genomic-value results when auxfiles = TRUE and estimation succeeds. It contains individual identifiers and predicted genomic values, and also observed phenotypes when those are present in the generated result. Otherwise NULL.

gs.get.im *Return the genomic-selection information matrix for a data set*

Description

Return the genomic-selection information matrix for a data set.

Usage

```
gs.get.im(data.set = "default")
```

Arguments

data.set	Name of the SelectionTools data set.

Value

An invisible named integer vector with components i, the number of individuals, and m, the number of markers in the requested data set.

gs.get.info.level *Return the genomic-selection information level for a data set*

Description

Return the genomic-selection information level for a data set.

Usage

```
gs.get.info.level(data.set ="default")
```

Arguments

data.set	Name of the SelectionTools data set.

Value

An integer scalar giving the current genomic-selection information level for the requested data set.

gs.get.V *Return the genomic-selection variance or covariance matrix for a data set*

Description

Return the genomic-selection variance or covariance matrix for a data set.

Usage

```
gs.get.V(data.set = "default")
```

Arguments

data.set	Name of the SelectionTools data set.

Value

Invisibly returns the current numeric variance/covariance matrix V; returns NULL if it cannot be obtained from the current data set.

gs.get.y — *Return the genomic-selection response vector for a data set*

Description

Return the genomic-selection response vector for a data set.

Usage

```
gs.get.y(data.set = "default")
```

Arguments

data.set	Name of the SelectionTools data set.

Value

Invisibly returns the current numeric response vector y; returns NULL if it cannot be obtained from the current data set.

gs.get.Z — *Return the genomic-selection marker or design matrix for a data set*

Description

Return the genomic-selection marker or design matrix for a data set.

Usage

```
gs.get.Z(data.set = "default")
```

Arguments

data.set	Name of the SelectionTools data set.

Value

Invisibly returns the current numeric marker/design matrix Z; returns NULL if it cannot be obtained from the current data set.

gs.info — *Print or handle a genomic-selection information message*

Description

Print or handle a genomic-selection information message.

Usage

```
gs.info(lev, msg)
```

Arguments

lev	Information or verbosity level.
msg	Message text.

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of issuing the requested genomic-selection information message.

gs.lambda.aov — *Estimate or set genomic-selection lambda values using the aov method*

Description

Estimate or set genomic-selection lambda values using the aov method.

Usage

```
gs.lambda.aov(hsq, alpha = 1, out.filename = "lambda", auxfiles = TRUE,
              data.set = "default")
```

Arguments

hsq	Heritability parameter.
alpha	Method-specific tuning or significance parameter.
out.filename	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
auxfiles	Logical. If `TRUE`, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
data.set	Name of the SelectionTools data set.

Value

Invisibly returns a numeric `matrix` containing the lambda results from the ANOVA method when `auxfiles = TRUE` and the compiled calculation succeeds; otherwise `NULL`.

---

gs.lambda.const — *Estimate or set genomic-selection lambda values using the const method*

Description

Estimate or set genomic-selection lambda values using the const method.

Usage

```
gs.lambda.const(lambda, out.filename = "lambda", auxfiles = FALSE,
                data.set = "default")
```

Arguments

`lambda`	Regularization parameter.
`out.filename`	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
`auxfiles`	Logical. If `TRUE`, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
`data.set`	Name of the SelectionTools data set.

Value

Invisibly returns a numeric `matrix` containing the lambda results from the constant-lambda calculation when `auxfiles = TRUE` and the compiled calculation succeeds; otherwise `NULL`.

---

gs.lambda.emstep — *Estimate or set genomic-selection lambda values using the emstep method*

Description

Estimate or set genomic-selection lambda values using the emstep method.

Usage

```
gs.lambda.emstep(constvar = FALSE, out.filename = "lambda", auxfiles = TRUE,
                 data.set = "default")
```

Arguments

constvar	Logical flag controlling use of a constant variance.
out.filename	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
auxfiles	Logical. If TRUE, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
data.set	Name of the SelectionTools data set.

Value

Invisibly returns a numeric matrix containing the lambda results from the EM-step method when auxfiles = TRUE and the compiled calculation succeeds; otherwise NULL.

---

gs.lambda.hsq *Estimate or set genomic-selection lambda values using the hsq method*

Description

Estimate or set genomic-selection lambda values using the hsq method.

Usage

```
gs.lambda.hsq(hsq, out.filename = "lambda", auxfiles = TRUE,
              data.set = "default")
```

Arguments

hsq	Heritability parameter.
out.filename	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
auxfiles	Logical. If TRUE, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
data.set	Name of the SelectionTools data set.

Value

Invisibly returns a numeric matrix containing the lambda results calculated from heritability when auxfiles = TRUE and the compiled calculation succeeds; otherwise NULL.

gs.lambda.reg — *Estimate or set genomic-selection lambda values using the reg method*

Description

Estimate or set genomic-selection lambda values using the reg method.

Usage

```
gs.lambda.reg(hsq, out.filename = "lambda", auxfiles = TRUE,
              data.set = "default")
```

Arguments

hsq	Heritability parameter.
out.filename	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
auxfiles	Logical. If TRUE, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
data.set	Name of the SelectionTools data set.

Value

Invisibly returns a numeric matrix containing the lambda results from the regression method when auxfiles = TRUE and the compiled calculation succeeds; otherwise NULL.

gs.lambda.rmla — *Estimate or set genomic-selection lambda values using the rmla method*

Description

Estimate or set genomic-selection lambda values using the rmla method.

Usage

```
gs.lambda.rmla(alpha = 1, maxiter = 1000, precision = 0.0001, hsq = 0.9,
               out.filename = "lambda", auxfiles = TRUE, data.set = "default")
```

Arguments

alpha	Method-specific tuning or significance parameter.
maxiter	Maximum number of iterations.
precision	Convergence tolerance.
hsq	Heritability parameter.
out.filename	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
auxfiles	Logical. If TRUE, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
data.set	Name of the SelectionTools data set.

Value

Invisibly returns a numeric matrix containing the lambda results from the RMLA method when auxfiles = TRUE and the compiled calculation succeeds; otherwise NULL.

---

gs.lambda.rmla.02 *Estimate or set genomic-selection lambda values using the rmla.02 method*

Description

Estimate or set genomic-selection lambda values using the rmla.02 method.

Usage

```
gs.lambda.rmla.02(alpha = 1, maxiter = 1000, precision = 0.001, hsq = 0.9,
                  out.filename = "lambda", auxfiles = TRUE,
                  data.set = "default")
```

Arguments

alpha	Method-specific tuning or significance parameter.
maxiter	Maximum number of iterations.
precision	Convergence tolerance.
hsq	Heritability parameter.
out.filename	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
auxfiles	Logical. If TRUE, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
data.set	Name of the SelectionTools data set.

Value

Invisibly returns a numeric `matrix` containing the lambda results from the RMLA.02 method when `auxfiles = TRUE` and the compiled calculation succeeds; otherwise `NULL`.

`gs.lambda.rmlc` *Estimate or set genomic-selection lambda values using the rmlc method*

Description

Estimate or set genomic-selection lambda values using the rmlc method.

Usage

```
gs.lambda.rmlc(maxiter = 1000, precision = 0.0001, hsq = 0.9,
               out.filename = "lambda", auxfiles = TRUE, data.set = "default")
```

Arguments

`maxiter`	Maximum number of iterations.
`precision`	Convergence tolerance.
`hsq`	Heritability parameter.
`out.filename`	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
`auxfiles`	Logical. If `TRUE`, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
`data.set`	Name of the SelectionTools data set.

Value

Invisibly returns a numeric `matrix` containing the lambda results from the RMLC method when `auxfiles = TRUE` and the compiled calculation succeeds; otherwise `NULL`.

gs.lambda.rmlr — *Estimate or set genomic-selection lambda values using the rmlr method*

Description

Estimate or set genomic-selection lambda values using the rmlr method.

Usage

```
gs.lambda.rmlr(maxiter = 1000, precision = 0.001, hsq = 0.9,
               out.filename = "lambda", auxfiles = TRUE, data.set = "default")
```

Arguments

maxiter	Maximum number of iterations.
precision	Convergence tolerance.
hsq	Heritability parameter.
out.filename	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
auxfiles	Logical. If TRUE, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
data.set	Name of the SelectionTools data set.

Value

Invisibly returns a numeric matrix containing the lambda results from the RMLR method when auxfiles = TRUE and the compiled calculation succeeds; otherwise NULL.

gs.lambda.rmlv — *Estimate or set genomic-selection lambda values using the rmlv method*

Description

Estimate or set genomic-selection lambda values using the rmlv method.

Usage

```
gs.lambda.rmlv(maxiter = 1000, precision = 0.001, out.filename = "lambda",
               auxfiles = TRUE, data.set = "default")
```

Arguments

maxiter	Maximum number of iterations.
precision	Convergence tolerance.
out.filename	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
auxfiles	Logical. If TRUE, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
data.set	Name of the SelectionTools data set.

Value

Invisibly returns a numeric matrix containing the lambda results from the RMLV method when auxfiles = TRUE and the compiled calculation succeeds; otherwise NULL.

---

gs.lambda.rrblup *Estimate or set genomic-selection lambda values using the rrblup method*

Description

Estimate or set genomic-selection lambda values using the rrblup method.

Usage

```
gs.lambda.rrblup(maxiter = 1000, precision = 1e-4, hsq = 0.8,
                 out.filename = "lambda", auxfiles = TRUE,
                 data.set = "default")
```

Arguments

maxiter	Maximum number of iterations.
precision	Convergence tolerance.
hsq	Heritability parameter.
out.filename	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
auxfiles	Logical. If TRUE, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
data.set	Name of the SelectionTools data set.

Value

Invisibly returns a numeric matrix containing the lambda results from RR-BLUP when auxfiles = TRUE and the compiled calculation succeeds; otherwise NULL.

gs.mme.coeff *Gs mme coeff*

Description

Build or return the coefficient matrix for genomic-selection mixed-model equations.

Usage

```
gs.mme.coeff(out.filename = "mme.coeff", auxfiles = FALSE,
             data.set = "default")
```

Arguments

out.filename	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
auxfiles	Logical. If TRUE, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
data.set	Name of the SelectionTools data set.

Value

Invisibly returns a numeric matrix containing the mixed-model coefficient matrix when auxfiles = TRUE and the compiled calculation succeeds; otherwise NULL.

gs.mme.invcoeff *Gs mme invcoeff*

Description

Build or return the inverse coefficient matrix for genomic-selection mixed-model equations.

Usage

```
gs.mme.invcoeff(out.filename = "inv.mme", auxfiles = TRUE)
```

Arguments

out.filename	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
auxfiles	Logical. If TRUE, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.

Value

Invisibly returns a numeric `matrix` containing the inverse mixed-model coefficient matrix when `auxfiles = TRUE` and the compiled calculation succeeds; otherwise `NULL`.

---

gs.mme.restcoeff *Gs mme restcoeff*

Description

Build or return the restricted coefficient matrix for genomic-selection mixed-model equations.

Usage

```
gs.mme.restcoeff(out.filename = "mme.coeff", auxfiles = FALSE)
```

Arguments

`out.filename`	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
`auxfiles`	Logical. If `TRUE`, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.

Value

Invisibly returns a numeric `matrix` containing the restricted mixed-model coefficient matrix when `auxfiles = TRUE` and the compiled calculation succeeds; otherwise `NULL`.

---

gs.mme.restrhs *Gs mme restrhs*

Description

Build or return the restricted right-hand side for genomic-selection mixed-model equations.

Usage

```
gs.mme.restrhs(out.filename = "mme.rhs", auxfiles = FALSE)
```

Arguments

`out.filename`	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
`auxfiles`	Logical. If `TRUE`, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.

Value

Invisibly returns a numeric `matrix` containing the restricted mixed-model right-hand side when `auxfiles = TRUE` and the compiled calculation succeeds; otherwise `NULL`.

`gs.mme.rhs` *Build or return the right-hand side for genomic-selection mixed-model equations*

Description

Build or return the right-hand side for genomic-selection mixed-model equations.

Usage

```
gs.mme.rhs(out.filename = "mme.rhs", auxfiles = FALSE, data.set = "default")
```

Arguments

`out.filename`	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
`auxfiles`	Logical. If `TRUE`, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
`data.set`	Name of the SelectionTools data set.

Value

Invisibly returns a numeric `matrix` containing the mixed-model right-hand side when `auxfiles = TRUE` and the compiled calculation succeeds; otherwise `NULL`.

`gs.mme.solve` *Gs mme solve*

Description

Build or return the mixed-model equation solution for genomic-selection mixed-model equations.

Usage

```
gs.mme.solve(out.filename = "mme.solution", auxfiles = FALSE,
             data.set = "default")
```

Arguments

out.filename	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
auxfiles	Logical. If TRUE, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
data.set	Name of the SelectionTools data set.

Value

Invisibly returns a numeric matrix containing the mixed-model equation solution when auxfiles = TRUE and the compiled calculation succeeds; otherwise NULL.

---

gs.mmet.coeff *Build or solve transformed mixed-model equations (coeff)*

Description

Build or solve transformed mixed-model equations (coeff).

Usage

```
gs.mmet.coeff(out.filename = "mme.coeff", auxfiles = FALSE,
              data.set = "default")
```

Arguments

out.filename	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
auxfiles	Logical. If TRUE, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
data.set	Name of the SelectionTools data set.

Value

Invisibly returns a numeric matrix containing the transformed mixed-model coefficient matrix when auxfiles = TRUE and the compiled calculation succeeds; otherwise NULL.

gs.mmet.coeff.3 *Build or solve transformed mixed-model equations (coeff.3)*

Description

Build or solve transformed mixed-model equations (coeff.3).

Usage

```
gs.mmet.coeff.3(out.filename = "mme.coeff", auxfiles = FALSE,
                data.set = "default")
```

Arguments

out.filename	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
auxfiles	Logical. If TRUE, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
data.set	Name of the SelectionTools data set.

Value

Invisibly returns a numeric matrix containing the alternative transformed mixed-model coefficient matrix when auxfiles = TRUE and the compiled calculation succeeds; otherwise NULL.

gs.mmet.rhs *Build or solve transformed mixed-model equations (rhs)*

Description

Build or solve transformed mixed-model equations (rhs).

Usage

```
gs.mmet.rhs(out.filename = "mme.rhs", auxfiles = FALSE, data.set = "default")
```

Arguments

out.filename	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
auxfiles	Logical. If TRUE, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
data.set	Name of the SelectionTools data set.

Value

Invisibly returns a numeric `matrix` containing the transformed mixed-model right-hand side when `auxfiles = TRUE` and the compiled calculation succeeds; otherwise `NULL`.

---

gs.mmet.solve *Build or solve transformed mixed-model equations (solve)*

Description

Build or solve transformed mixed-model equations (solve).

Usage

```
gs.mmet.solve(out.filename = "mme.solution", auxfiles = FALSE,
              data.set = "default")
```

Arguments

`out.filename`	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
`auxfiles`	Logical. If `TRUE`, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
`data.set`	Name of the SelectionTools data set.

Value

Invisibly returns a numeric `matrix` containing the transformed mixed-model solution when `auxfiles = TRUE` and the compiled calculation succeeds; otherwise `NULL`.

---

gs.neg.effall *Set or process all marker effects using the negative-effect convention*

Description

Set or process all marker effects using the negative-effect convention.

Usage

```
gs.neg.effall(data.set ="default")
```

Arguments

`data.set`	Name of the SelectionTools data set.

Value

On a successful run, an invisible implementation-level `list` returned by the final timing/information helper; the meaningful result is the side effect of processing marker effects using the negative-effect convention. Validation failures may return `NULL`.

`gs.plot.effects` *gs.plot.effects()*

Description

Plotting function for the estimated marker effects.

Usage

```
gs.plot.effects(data.set = "default", absolute = TRUE, alpha = 0.05,
                p.adjust.method = "none", pvals = NULL, xlim = NULL,
                ylim = NULL, plt = TRUE, col.sp = "blue", pch.sp = 19,
                col.sn = "blue", pch.sn = 19, col.ns = "blue", pch.ns = 1,
                ...)
```

Arguments

`data.set`	character. Name of the data set that is used by the function.
`absolute`	Function argument.
`alpha`	Function argument.
`p.adjust.method`	Function argument.
`pvals`	Function argument.
`xlim`	Function argument.
`ylim`	Function argument.
`plt`	Function argument.
`col.sp`	Function argument.
`pch.sp`	Function argument.
`col.sn`	Function argument.
`pch.sn`	Function argument.
`col.ns`	Function argument.
`pch.ns`	Function argument.
`...`	Additional arguments.

Value

Invisibly returns the `data.frame` used for plotting, with columns `chrom`, `pos`, `name`, `cpos`, and `effect`, and a `pvalue` column when p-values are available. Returns `NULL` if the required effect or map data are unavailable.

gs.plot.model.fit *Plot the Fit of a Genomic Prediction Model*

Description

Plots observed performance values against genomic predictions for the individuals used to fit the marker model.

Usage

```
gs.plot.model.fit(training.set ="default", title = "")
```

Arguments

training.set	Character string naming the data set used for marker-effect estimation and model-fit assessment.
title	Character string added to the plot title.

Details

The function calls gs.predict.genotypes with training.set used as both training and prediction set. The plotted y values are the observed performance values stored in that data set, whereas yhat are the direct genomic predictions obtained from the estimated marker effects. In the simulation workflow, these yhat values represent estimated genotypic values of the training population.

The plot includes the identity line, horizontal and vertical lines through the respective means, and reports both the Pearson correlation r and the Spearman rank correlation rho between y and yhat. Marker effects must have been estimated before this function is called, and performance data must be available in training.set.

Value

No useful return value is intended. The function is called for its side effect of plotting model-fit information; invalid input returns NULL invisibly.

See Also

gs.esteff.rr, gs.predict.genotypes, gs.plot.validation

gs.plot.validation *gs.plot.validation()*

Description

Creates a graphical output wit two plots. The left plot shows the model fit as produced with the function *gs.plot.model.fit*. The right plot shows a scatter plot of the predicted DGV and the real phenotypes for a set of individuals that was not used for estimation of marker effects.

Usage

```
gs.plot.validation(training.set, prediction.set, title="")
```

Arguments

`training.set`	character. Name of the data set that was used for estimation of marker effects.
`prediction.set`	character. Name of the data set that is used for prediction of DGVs within this function.
`title`	character. Title of the plot.

Details

Within this function, the function *gs.estimate.gv* is called twice. For the model fit, the estimation set is both used as estimation and prediction set in *gs.estimate.gv*. To validate the model, the function *gs.estimate.gv* is called again with the validation set used as prediction set.
The validation set is a set of individuals that was not used for building the model, but having phenotypes to be correlated to the predicted DGVs.

Value

No useful return value is intended. The function is called for its side effect of plotting validation information; invalid input returns `NULL` invisibly.

gs.pos.effall *Set or process all marker effects using the positive-effect convention*

Description

Set or process all marker effects using the positive-effect convention.

Usage

```
gs.pos.effall(data.set ="default")
```

Arguments

data.set	Name of the SelectionTools data set.

Value

On a successful run, an invisible implementation-level list returned by the final timing/information helper; the meaningful result is the side effect of processing marker effects using the positive-effect convention. Validation failures may return NULL.

---

gs.predict.genotypes *gs.predict.genotypes()*

Description

Predicts direct genomic values (DGV) for a given set of individuals in the prediction set. Works only after the marker effects were estimated.

Usage

```
gs.predict.genotypes(training.set ="default", prediction.set ="default",
                     out.filename = "breeding.values", auxfiles = TRUE)
```

Arguments

training.set	character. Name of the data set that was used for estimating marker effects
prediction.set	character. Name of the data set that is used for predicting DGVs
out.filename	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
auxfiles	Logical. If TRUE, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.

Value

Invisibly returns a data.frame with prediction results when auxfiles = TRUE and prediction succeeds. It contains individual identifiers and predicted genomic values, and also observed phenotypes when those are present in the generated result. Otherwise NULL.

gs.reset *Reset genomic-selection data and settings*

Description

Reset genomic-selection data and settings.

Usage

```
gs.reset()
```

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of resetting the genomic-selection state.

gs.restrict.marker.data.01 *Restrict marker data using the package genomic-selection interface*

Description

Restrict marker data using the package genomic-selection interface.

Usage

```
gs.restrict.marker.data.01(outfile, traitfile = NULL, NoAll.MAX = 999,
                           MaMis.MAX = 1, ExHet.MIN = 0, InMis.MAX = 1,
                           data.set = "default")
```

Arguments

`outfile`	Function argument used by this operation.
`traitfile`	Function argument used by this operation.
`NoAll.MAX`	Function argument used by this operation.
`MaMis.MAX`	Function argument used by this operation.
`ExHet.MIN`	Function argument used by this operation.
`InMis.MAX`	Function argument used by this operation.
`data.set`	Name of the SelectionTools data set.

Value

On a successful run, an invisible implementation-level `list` returned by the final timing/information helper; the meaningful result is the side effect of restricting the genomic-selection marker data. Validation failures may return `NULL`.

gs.return.effects *gs.return.effects()*

Description

Returns the marker effects currently stored in a SelectionTools data set. The underlying routine writes the effects to an intermediate file; the R wrapper reads that file into R and deletes it before returning.

Usage

```
gs.return.effects(out.filename ="effects", data.set ="default")
```

Arguments

out.filename	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
data.set	character. Name of the data set that is used by the function.

Value

A data.frame containing the marker-effect table. If the underlying routine reports an error, NULL is returned invisibly. The intermediate file used to transfer the table is deleted after reading.

Note

This function provides access to effects stored by genomic-effect estimation functions such as gs.esteff.rr(), gs.esteff.rmlc(), and gs.esteff.rmlv(). Their auxfiles argument defaults to FALSE; gs.return.effects() can be used to retrieve the stored effects independently of that output option.

gs.return.pvals *Return marker p-values for a genomic-selection data set*

Description

Return marker p-values for a genomic-selection data set.

Usage

```
gs.return.pvals(out.filename ="effects", data.set ="default")
```

Arguments

out.filename	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
data.set	Name of the SelectionTools data set.

Value

A data.frame containing the marker p-value table. If the underlying routine reports an error, NULL is returned invisibly. The intermediate file used to transfer the table is deleted after reading.

---

gs.set.all.info.levels *Set all info levels used by the genomic-selection routines*

Description

Set all info levels used by the genomic-selection routines.

Usage

```
gs.set.all.info.levels(level)
```

Arguments

level	Function argument used by this operation.

Value

An invisible list returned by the underlying .C routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of setting all genomic-selection information levels.

---

gs.set.allele.codes *Set allele codes used by the genomic-selection routines*

Description

Set allele codes used by the genomic-selection routines.

Usage

```
gs.set.allele.codes(aa = 0, aA = 1, AA = 2, an = 0.5, nn = 1, nA = 1.5,
                    data.set ="default")
```

Arguments

aa	Function argument used by this operation.
aA	Function argument used by this operation.
AA	Function argument used by this operation.
an	Function argument used by this operation.
nn	Function argument used by this operation.
nA	Function argument used by this operation.
data.set	Name of the SelectionTools data set.

Value

NULL, returned invisibly. The function is called for its side effect of setting the allele codes used by the genomic-selection routines.

---

gs.set.effects *Set effects used by the genomic-selection routines*

Description

Set effects used by the genomic-selection routines.

Usage

```
gs.set.effects(eff, filename ="seteffects", data.set ="default")
```

Arguments

eff	Function argument used by this operation.
filename	Character string retained for compatibility and used as part of the name of the internal session-temporary file used to pass the effects to compiled code. The file is removed before the function returns.
data.set	Name of the SelectionTools data set.

Value

On a successful run, an invisible implementation-level list returned by the final timing/information helper; the meaningful result is the side effect of setting marker effects in the genomic-selection data set. Validation failures may return NULL.

gs.set.info.level *Set info level used by the genomic-selection routines*

Description

Set info level used by the genomic-selection routines.

Usage

```
gs.set.info.level(level, data.set ="default")
```

Arguments

level	Function argument used by this operation.
data.set	Name of the SelectionTools data set.

Value

An invisible list returned by the underlying .C routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of setting the genomic-selection information level for the data set.

gs.set.lambda *Set lambda used by the genomic-selection routines*

Description

Set lambda used by the genomic-selection routines.

Usage

```
gs.set.lambda(lambda, out.filename = "lambda", auxfiles = FALSE,
              data.set = "default")
```

Arguments

lambda	Regularization parameter.
out.filename	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
auxfiles	Logical. If TRUE, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
data.set	Name of the SelectionTools data set.

Value

Invisibly returns a numeric `matrix` containing the lambda values produced while setting lambda when `auxfiles = TRUE` and the compiled calculation succeeds; otherwise `NULL`.

`gs.set.num.threads` *Set num threads used by the genomic-selection routines*

Description

Set num threads used by the genomic-selection routines.

Usage

```
gs.set.num.threads(active)
```

Arguments

`active`	Function argument used by this operation.

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of setting the number of genomic-selection computational threads.

`gs.set.performance.data` *Set performance data used by the genomic-selection routines*

Description

Set performance data used by the genomic-selection routines.

Usage

```
gs.set.performance.data(y, data.set = "default")
```

Arguments

`y`	Response vector.
`data.set`	Name of the SelectionTools data set.

Value

An invisible integer scalar containing the status value returned by the compiled performance-data setter.

gs.single.marker.aov *Perform single-marker analysis for a genomic-selection data set*

Description

Perform single-marker analysis for a genomic-selection data set.

Usage

```
gs.single.marker.aov(alpha = 1, out.filename = "sm.reg", auxfiles = TRUE,
                     data.set = "default")
```

Arguments

alpha	Method-specific tuning or significance parameter.
out.filename	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
auxfiles	Logical. If TRUE, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
data.set	Name of the SelectionTools data set.

Value

Invisibly returns a numeric matrix containing the single-marker ANOVA results when auxfiles = TRUE and the compiled calculation succeeds; otherwise NULL.

gs.single.marker.reg *Perform single-marker analysis for a genomic-selection data set*

Description

Perform single-marker analysis for a genomic-selection data set.

Usage

```
gs.single.marker.reg(out.filename = "sm.reg", auxfiles = TRUE,
                     data.set = "default")
```

Arguments

out.filename	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
auxfiles	Logical. If TRUE, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
data.set	Name of the SelectionTools data set.

Value

Invisibly returns a numeric `matrix` containing the single-marker regression results when `auxfiles = TRUE` and the compiled calculation succeeds; otherwise `NULL`.

---

gs.start.timer *Start or stop the genomic-selection timing helper*

Description

Start or stop the genomic-selection timing helper.

Usage

```
gs.start.timer(depth=1)
```

Arguments

`depth`	Function argument used by this operation.

Value

An invisible implementation-level `list` returned by the SelectionTools timing helper. The function is called for the side effect of starting timing.

---

gs.stop.timer *Start or stop the genomic-selection timing helper*

Description

Start or stop the genomic-selection timing helper.

Usage

```
gs.stop.timer(depth=1, info.level=0)
```

Arguments

`depth`	Function argument used by this operation.
`info.level`	Function argument used by this operation.

Value

An invisible implementation-level `list` returned by the SelectionTools information helper after reporting elapsed time. The function is called for the timing/reporting side effect.

gs.test.mp *Test marker or model parameters used by genomic-selection routines*

Description

Test marker or model parameters used by genomic-selection routines.

Usage

```
gs.test.mp(n = 0)
```

Arguments

n	Number of items or repetitions.

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of running the compiled marker/model-parameter test.

gs.testeff.lsq *Test estimated marker effects using the indicated method*

Description

Test estimated marker effects using the indicated method.

Usage

```
gs.testeff.lsq(nperm = 0, out.filename = "test.lsq", auxfiles = TRUE,
               data.set ="default")
```

Arguments

nperm	Function argument used by this operation.
out.filename	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
auxfiles	Logical. If `TRUE`, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
data.set	Name of the SelectionTools data set.

Value

Invisibly returns a numeric `matrix` containing the LSQ effect-test results when `auxfiles = TRUE` and the compiled calculation succeeds; otherwise `NULL`.

gs.testeff.rmlc *Test estimated marker effects using the indicated method*

Description

Test estimated marker effects using the indicated method.

Usage

```
gs.testeff.rmlc(nperm = 1000, maxiter = 1000, precision = 0.001, hsq = 0.9,
                out.filename = "test.rmlc", auxfiles = TRUE,
                data.set ="default")
```

Arguments

nperm	Function argument used by this operation.
maxiter	Maximum number of iterations.
precision	Convergence tolerance.
hsq	Heritability parameter.
out.filename	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
auxfiles	Logical. If TRUE, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
data.set	Name of the SelectionTools data set.

Value

Invisibly returns a data.frame containing the RMLC effect-test results when auxfiles = TRUE and the test succeeds; otherwise NULL.

gs.testeff.rmlv *Test estimated marker effects using the indicated method*

Description

Test estimated marker effects using the indicated method.

Usage

```
gs.testeff.rmlv(nperm = 1000, maxiter = 1000, precision = 0.001, hsq = 0.9,
                out.filename = "test.rmlv", auxfiles = TRUE,
                data.set ="default")
```

Arguments

nperm	Function argument used by this operation.
maxiter	Maximum number of iterations.
precision	Convergence tolerance.
hsq	Heritability parameter.
out.filename	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
auxfiles	Logical. If TRUE, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
data.set	Name of the SelectionTools data set.

Value

Invisibly returns a numeric matrix containing the RMLV effect-test results when auxfiles = TRUE and the compiled calculation succeeds; otherwise NULL.

---

gs.vc.rrblup *Estimate variance components for ridge-regression BLUP*

Description

Estimate variance components for ridge-regression BLUP.

Usage

```
gs.vc.rrblup(maxiter = 1000, precision = 1e-4, hsq = 0.8,
             out.filename = "lambda", auxfiles = TRUE, data.set = "default")
```

Arguments

maxiter	Maximum number of iterations.
precision	Convergence tolerance.
hsq	Heritability parameter.
out.filename	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
auxfiles	Logical. If TRUE, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
data.set	Name of the SelectionTools data set.

Value

Invisibly returns a numeric matrix containing the variance-component results from RR-BLUP when auxfiles = TRUE and the compiled calculation succeeds; otherwise NULL.

gs.w.aov — *Calculate weights or statistics used by the genomic-selection ANOVA method*

Description

Calculate weights or statistics used by the genomic-selection ANOVA method.

Usage

```
gs.w.aov(alpha = 1, out.filename = "W", auxfiles = TRUE, data.set = "default")
```

Arguments

alpha	Method-specific tuning or significance parameter.
out.filename	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
auxfiles	Logical. If TRUE, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
data.set	Name of the SelectionTools data set.

Value

Invisibly returns a numeric matrix containing the ANOVA weights/statistics when auxfiles = TRUE and the compiled calculation succeeds; otherwise NULL.

gs.write.pseff — *Write Marker Effects for the Simulation Backend*

Description

Writes estimated marker effects in the effect-file format used by the simulation backend.

Usage

```
gs.write.pseff(e, beta0 = NA, file)
```

Arguments

e	A marker-effect table, typically returned by gs.return.effects(). The first row contains the general mean and the remaining rows contain marker names, effect alleles, complementary alleles, and estimated marker effects.
beta0	Optional general mean on the genomic-selection scale. If NA, the general mean in the first row of e is used.
file	Character string naming the effect file to be written. This argument is required because the function writes a persistent file.

Details

The genomic-selection and simulation parts of SelectionTools use different origins for additive marker coding.

With the default allele codes used by the genomic-selection routines, `gs.set.allele.codes()` assigns the additive dosage codes 0, 1, and 2 to the genotypes with zero, one, and two copies of the effect allele, respectively. For marker j with estimated effect u_j, the marker contribution is therefore 0, u_j, or $2u_j$.

The simulation effect-file representation stores an additive effect per allele. The effect allele is written with effect $+u_j/2$ and the complementary allele with effect $-u_j/2$. The corresponding diploid marker contributions are therefore $-u_j$, 0, and $+u_j$. This is equivalent to changing the marker dosage from z_{ij} to

$$z^*_{ij} = z_{ij} - 1.$$

On the genomic-selection scale,

$$\hat{g}_i = \hat{\mu}_{012} + \sum_{j=1}^{m} z_{ij}\hat{u}_j.$$

Using $z_{ij} = z^*_{ij} + 1$ gives

$$\hat{g}_i = \left(\hat{\mu}_{012} + \sum_{j=1}^{m} \hat{u}_j\right) + \sum_{j=1}^{m} z^*_{ij}\hat{u}_j.$$

Consequently, the first line written to the effect file is

$$\hat{\mu}_{\mathrm{sim}} = \hat{\mu}_{012} + \sum_{j=1}^{m} \hat{u}_j.$$

If `beta0` is supplied, it replaces $\hat{\mu}_{012}$ before this coding correction is applied.

This is the coding-shift correction described by Stranden and Christensen (2011), Equation (2). The shift is distinct from allele-frequency centering. For coding $z^c_{ij} = z_{ij} - 2p_j$, the corresponding mean shift is $2\sum_j p_j u_j$; the simulation effect-file coding instead uses $z^*_{ij} = z_{ij} - 1$.

The exact equivalence described above applies to the default allele codes. `gs.set.allele.codes()` permits power users to define other codes; an arbitrary custom coding is not in general represented by the fixed $-u_j/2$, $+u_j/2$ simulation allele effects.

Value

`NULL`, returned invisibly by the file-writing operation. The function is called for its side effect of writing marker effects for the simulation backend.

References

Stranden I, Christensen OF (2011) Allele coding in genomic evaluation. Genetics Selection Evolution 43:25. See Equation (2).

See Also

`gs.return.effects`, `gs.set.allele.codes`, `st.set.sim.ef`, `load.effmap`

homozygote *homozygote()*

Description

Produces a dataframe of a population whith *NoInd* individuals which carry at all loci the same allele (*allele*). This dataframe can be used to initialise a population.

Usage

```
homozygote(allele, NoInd = 1)
```

Arguments

allele	Allele which is carried at all loci
NoInd	Number of individuals to be generated

Value

A data.frame with columns ind, chrom, hom, pos, and all, representing the requested homozygous genome.

Note

See init.population for an example

il.eval.library *Construct or evaluate introgression-line libraries*

Description

Construct or evaluate introgression-line libraries.

Usage

```
il.eval.library(rp.allele = 1, dp.allele = 2, chrom = 0, lower = 0, upper = 0,
                data.set = "default")
```

Arguments

rp.allele	Allele-related parameter.
dp.allele	Allele-related parameter.
chrom	Chromosome identifier.
lower	Function argument used by this operation.
upper	Function argument used by this operation.
data.set	Name of the SelectionTools data set.

Value

A named numeric vector summarizing the evaluated introgression-line library. It contains `cov`, `dep`, `seglib`, `dpg`, `ndps`, and `ldps`; when target regions are supplied it additionally contains `tr.dpg`, `nt.rpg`, `nt.ndps`, and `nt.ldps`. Returns `NULL` on validation failure.

`il.eval.lines` *Construct or evaluate introgression-line libraries*

Description

Construct or evaluate introgression-line libraries.

Usage

```
il.eval.lines(allele = 1, chrom = 0, lower = 0, upper = 0, exclude = 0,
              data.set = "default")
```

Arguments

`allele`	Allele specification.
`chrom`	Chromosome identifier.
`lower`	Function argument used by this operation.
`upper`	Function argument used by this operation.
`exclude`	Function argument used by this operation.
`data.set`	Name of the SelectionTools data set.

Value

A `data.frame` with one row per evaluated line and columns `name`, `nseg`, `lseg`, `abs`, `tot`, and `rel`, summarizing introgressed segments and their lengths/proportions. Returns `NULL` on validation failure.

`il.ideal.library` *Construct or evaluate introgression-line libraries*

Description

Construct or evaluate introgression-line libraries.

Usage

```
il.ideal.library(n.c = 5, l.c = 100, s.l = 20, data.set = "default")
```

Arguments

n.c	Number or size used by the operation.
l.c	Function argument used by this operation.
s.l	Function argument used by this operation.
data.set	Name of the SelectionTools data set.

Value

An integer cleanup status returned by the final unlink() call after construction of the ideal introgression-line library. The main result is the population/library created by the function.

---

il.overlapping.library

Construct or evaluate introgression-line libraries

Description

Construct or evaluate introgression-line libraries.

Usage

```
il.overlapping.library(n.c = 5, l.c = 100, s.l = 20, data.set = "default")
```

Arguments

n.c	Number or size used by the operation.
l.c	Function argument used by this operation.
s.l	Function argument used by this operation.
data.set	Name of the SelectionTools data set.

Value

An integer cleanup status returned by the final unlink() call after construction of the overlapping introgression-line library. The main result is the population/library created by the function.

info *Print or handle an information message*

Description

Print or handle an information message.

Usage

```
info(lev, msg)
```

Arguments

lev	Information or verbosity level.
msg	Message text.

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of issuing the requested information message.

info.cat *Emit information-level-controlled status output*

Description

Emit a status message when the requested information level is enabled. The message is signaled with `message()` and can therefore also be suppressed with `suppressMessages()`.

Usage

```
info.cat(lev, msg)
```

Arguments

lev	Information or verbosity level.
msg	Message text.

Value

`NULL`. The function is called for its side effect of conditionally signaling a status message according to the current information level.

`init.population` *init.population()*

Description

Creates a simulation population from the internal chromosome-segment data frame representation.

Usage

```
init.population(name, data, delete=TRUE)
```

Arguments

`name`	Character string naming the simulation population to initialize.
`data`	Data frame with five columns representing individual, chromosome, homologue, segment position, and allele.
`delete`	Logical. If true, remove an existing population with the same name before initialization.

Details

The input data frame must have exactly five columns named or interpretable as individual number, chromosome number, homologue number, position, and allele. Each row describes the start of a chromosome segment carrying the specified allele. The data must be ordered consistently with the simulation representation.

The number of input rows must be positive. Chromosome and homologue identifiers must be compatible with the currently defined simulation genome. The routine constructs the raw chromosome representation; marker genotypes and evaluated genetic values are derived data and are not implied merely by initialization.

When `delete=TRUE`, an existing population of the same name is removed before initialization. If creation fails while chromosome segments are being inserted, the incomplete new population is removed rather than retained.

Uses the currently defined genome and linkage map. Both must be defined before calling this function.

Value

The function initialises the named simulation population from the supplied data in the backend. Returns list of name, total number of alleles and input data columns. The function is primarily called for its side effect of creating the simulation population.

Examples

```
# Define a population explicitly
set.genome.par(no.chrom = 1, no.hom = 2, chrom.len = 1)
define.map("form trait1 1/0.5")

# Individual 1: genotype 1/1
# Individual 2: genotype 1/2
pop <- data.frame(
   ind   = c(1, 1, 2, 2),
   chrom = c(1, 1, 1, 1),
   hom   = c(1, 2, 1, 2),
   pos   = c(0, 0, 0, 0),
   all   = c(1, 1, 1, 2)
)

init.population("P1", pop)
evaluate.genotype("P1", "form")

# Define a genome from a data frame and create homozygous individuals
reset.all()

map <- data.frame(
   chrom = 1,
   pos = 0.5,
   name = "form"
)
define.genome(map)

# Two individuals, both homozygous for allele 1
init.population("P2", homozygote(1, NoInd = 2))
evaluate.genotype("P2", "form")
```

---

lib00.map1 *Linkage Map for Marker-Assisted Backcrossing Examples*

Description

A linkage-map data set used in the marker-assisted backcrossing examples based on Frisch et al. (1999). This version represents the map with approximately 80 marker loci used for background selection.

Usage

```
data(lib00.map1)
```

Format

An R data object containing linkage-map records in the format used by SelectionTools linkage-map input routines.

Source

Data used for the examples based on Frisch et al. (1999), Crop Science 39:1295–1301.

---

lib00.map1a *Reduced Linkage Map for Marker-Assisted Backcrossing Examples*

Description

A reduced linkage-map data set used in the marker-assisted backcrossing examples based on Frisch et al. (1999). This version represents the reduced map with approximately 40 marker loci used to compare marker density for background selection.

Usage

```
data(lib00.map1a)
```

Format

An R data object containing linkage-map records in the format used by SelectionTools linkage-map input routines.

Source

Data used for the examples based on Frisch et al. (1999), Crop Science 39:1295–1301.

---

lib00.map2 *Linkage Map for Three-Stage Marker-Assisted Backcrossing*

Description

A linkage-map data set used in the three-stage marker-assisted backcrossing examples based on Frisch et al. (1999). The example uses target, flanking and background-marker information for selection.

Usage

```
data(lib00.map2)
```

Format

An R data object containing linkage-map records in the format used by SelectionTools linkage-map input routines.

Source

Data used for the examples based on Frisch et al. (1999), Crop Science 39:1295–1301.

linkage.drag *Calculate or summarize linkage drag*

Description

Calculate or summarize linkage drag.

Usage

```
linkage.drag(pops, allele, chrom, pos)
```

Arguments

pops	Population name or names.
allele	Allele specification.
chrom	Chromosome identifier.
pos	Position on the chromosome.

Value

A numeric matrix with one row per population and columns Obs, Mean, SDev, Min, Q10, Med, and Max, summarizing linkage-drag segment lengths.

linkage.map.get *Get a linkage map used by the simulation routines*

Description

Returns the currently active simulation linkage map.

Usage

```
linkage.map.get()
```

Details

The result contains the map points used by the simulation backend, including chromosome, position, locus name, and class, in the current simulation map order.

Value

A data.frame with columns chrom, pos, name, and class, describing the active simulation linkage map.

linkage.map.load *Load a linkage map used by the simulation routines*

Description

Loads and prepares the linkage map used by the simulation backend.

Usage

```
linkage.map.load(file, disperse=TRUE, file.disperse=NA, disperse.factor=100)
```

Arguments

file	Character string naming the simulation map file.
disperse	Logical flag controlling dispersion of coincident map positions.
file.disperse	Optional file-related dispersion specification used by the existing map loader.
disperse.factor	Numeric factor controlling the dispersion scale used by the existing loader.

Details

The simulation map supplies chromosome, position, locus-name, and class information used by population genotyping and recombination. The loader prepares map points in a valid access order and can disperse coincident positions when requested. The active simulation map is a simulation-wide object and is separate from a SelectionTools marker data set.

Value

An invisible `list` with a `retval` status component. Invalid input returns `list(retval = -2L)` invisibly; otherwise the list is returned by the compiled map loader. The main effect is loading the linkage map.

linkage.map.save *Save a linkage map used by the simulation routines*

Description

Writes a simulation linkage map to a text file.

Usage

```
linkage.map.save(file, dta.map=linkage.map.get())
```

Arguments

file	Character string naming the output map file.
dta.map	Map data to write; by default the currently active simulation linkage map.

Details

By default the currently active simulation linkage map is obtained and written. A supplied dta.map can instead be written. Saving does not alter the active map or any simulation population.

Value

NULL, returned invisibly by write.table(). The function is called for its side effect of writing the linkage map.

---

list.effects *list.effects()*

Description

Rerturns all defined effects and their weights.

Usage

```
list.effects()
```

Value

A data.frame with columns effect and weight, listing the currently defined effects and their weights.

Note

See define.effects for for an example

list.populations *list.populations*

Description

Returns the names and active sizes of the currently registered simulation populations.

Usage

```
list.populations()
```

Details

The result reflects the simulation population registry and is independent of SelectionTools marker-data-set names.

Value

Returns the same `data.frame` as `population.list()`, with columns `PopName` and `count`.

load.effmap *load.effmap()*

Description

Loads an effect file

Usage

```
load.effmap(name, file = NA)
```

Arguments

`name`	Name of the effect
`file`	File that contains the effect description

Details

Effect files are text files can be generated either by hand with an editor or with the command `generate.effect.file()`.

An effect file consists of a value, which is assigned to each individual of a population (the population mean) and a list of effects which are added to the population mean if the certain allelic combinations occur in an individual. The order of the effects is arbitrary, but the population mean must occur first in the map file.
Mean;

Structure of an effect file:
Locus Allele [Locus Allele . . .] Effect;
[,. . .]

Mean	The population mean
Classname	A class of Loci to which effects are assigned to
Allele	The alleles to which the effects are assigned
[Allele]	Optional. Is used to define dominance effects epistatic effects
[,. . .]	An arbitrary number of effects can be defined seperated by commas

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of loading the requested effect map.

Note

See `define.effects` for for an example

`load.internal.effmap` *Load an internal effect map*

Description

Load an internal effect map.

Usage

```
load.internal.effmap(name, spec, weight)
```

Arguments

`name`	Function argument used by this operation.
`spec`	Function argument used by this operation.
`weight`	Weight value.

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of loading the requested internal effect map.

load.linkage.map *load.linkage.map()*

Description

Loads the simulation linkage map from a text file.

Usage

```
load.linkage.map(file, disperse=TRUE, file.disperse=NA, disperse.factor=100)
```

Arguments

file	Character string naming the simulation linkage-map input file.
disperse	Logical. If true, coincident locus positions are deterministically separated during map preparation.
file.disperse	Optional character string naming a file to which the dispersed map is written. An omitted value does not create an additional file.
disperse.factor	Finite numeric factor controlling the spacing used when coincident positions are dispersed.

Details

The simulation linkage map is distinct from the marker-data map managed by st.read.map. It defines the loci available to simulation-population genotyping, recombination, and evaluation.

A map file contains one locus definition per record with four fields:

Chromosome Chromosome number.

Position Distance from the chromosome telomere in Morgan.

Locusname Name of the locus.

Classname Class to which the locus belongs.

The record grammar is therefore Chromosome Position Locusname Classname. Any number of locus records can be supplied.

The input file does not need to be pre-sorted. The current loader parses and validates all records and prepares the loci in chromosome/position order. Coincident map positions can be dispersed according to disperse, file.disperse, and disperse.factor. Only one simulation linkage map is active at a time.

The function linkage.map.load provides the same linkage-map loading operation.

Value

Returns the same invisible status list as linkage.map.load(), including its retval component.

`mab.compare` *Run and Compare a Series of Marker-Assisted Backcrossing Simulations*

Description

Runs up to nine marker-assisted backcrossing simulation scenarios by calling `mab.simulate()` for each active scenario. Scenario A provides the base settings; scenarios B through I can override individual settings.

Usage

```
mab.compare(
    a.simulation.name,
    a.simulation.run,
    a.repetitions = 1000,
    a.recurrent.parent = "H",
    a.donor.parent = "H",
    a.linkage.map,
    a.target.loci = NULL,
    a.flanking.loci = NULL,
    a.recipient.loci = NULL,
    a.gen.type = NULL,
    a.population.size = NULL,
    a.sel.strategy = NULL,
    a.no.selected = NULL,
    a.no.preselected = NULL,
    a.reg.chr = NULL,
    a.reg.begin = NULL,
    a.reg.end = NULL,
    a.missing.allele = "9",
    a.success.factor = 1,
    b.simulation.name = NULL,
    c.simulation.name = NULL,
    d.simulation.name = NULL,
    b.simulation.run = NULL,
    c.simulation.run = NULL,
    d.simulation.run = NULL,
    b.repetitions = NULL,
    c.repetitions = NULL,
    d.repetitions = NULL,
    b.recurrent.parent = NULL,
    c.recurrent.parent = NULL,
    d.recurrent.parent = NULL,
    b.donor.parent = NULL,
    c.donor.parent = NULL,
    d.donor.parent = NULL,
```

```
b.linkage.map = NULL,
c.linkage.map = NULL,
d.linkage.map = NULL,
b.target.loci = NULL,
c.target.loci = NULL,
d.target.loci = NULL,
b.flanking.loci = NULL,
c.flanking.loci = NULL,
d.flanking.loci = NULL,
b.recipient.loci = NULL,
c.recipient.loci = NULL,
d.recipient.loci = NULL,
b.gen.type = NULL,
c.gen.type = NULL,
d.gen.type = NULL,
b.population.size = NULL,
c.population.size = NULL,
d.population.size = NULL,
b.sel.strategy = NULL,
c.sel.strategy = NULL,
d.sel.strategy = NULL,
b.no.selected = NULL,
c.no.selected = NULL,
d.no.selected = NULL,
b.no.preselected = NULL,
c.no.preselected = NULL,
d.no.preselected = NULL,
b.reg.chr = NULL,
c.reg.chr = NULL,
d.reg.chr = NULL,
b.reg.begin = NULL,
c.reg.begin = NULL,
d.reg.begin = NULL,
b.reg.end = NULL,
c.reg.end = NULL,
d.reg.end = NULL,
b.missing.allele = NULL,
c.missing.allele = NULL,
d.missing.allele = NULL,
b.success.factor = NULL,
c.success.factor = NULL,
d.success.factor = NULL,
e.simulation.name = NULL,
f.simulation.name = NULL,
g.simulation.name = NULL,
e.simulation.run = NULL,
f.simulation.run = NULL,
g.simulation.run = NULL,
```

```
e.repetitions = NULL,
f.repetitions = NULL,
g.repetitions = NULL,
e.recurrent.parent = NULL,
f.recurrent.parent = NULL,
g.recurrent.parent = NULL,
e.donor.parent = NULL,
f.donor.parent = NULL,
g.donor.parent = NULL,
e.linkage.map = NULL,
f.linkage.map = NULL,
g.linkage.map = NULL,
e.target.loci = NULL,
f.target.loci = NULL,
g.target.loci = NULL,
e.flanking.loci = NULL,
f.flanking.loci = NULL,
g.flanking.loci = NULL,
e.recipient.loci = NULL,
f.recipient.loci = NULL,
g.recipient.loci = NULL,
e.gen.type = NULL,
f.gen.type = NULL,
g.gen.type = NULL,
e.population.size = NULL,
f.population.size = NULL,
g.population.size = NULL,
e.sel.strategy = NULL,
f.sel.strategy = NULL,
g.sel.strategy = NULL,
e.no.selected = NULL,
f.no.selected = NULL,
g.no.selected = NULL,
e.no.preselected = NULL,
f.no.preselected = NULL,
g.no.preselected = NULL,
e.reg.chr = NULL,
f.reg.chr = NULL,
g.reg.chr = NULL,
e.reg.begin = NULL,
f.reg.begin = NULL,
g.reg.begin = NULL,
e.reg.end = NULL,
f.reg.end = NULL,
g.reg.end = NULL,
e.missing.allele = NULL,
f.missing.allele = NULL,
g.missing.allele = NULL,
```

```
    e.success.factor = NULL,
    f.success.factor = NULL,
    g.success.factor = NULL,
    h.simulation.name = NULL,
    i.simulation.name = NULL,
    h.simulation.run = NULL,
    i.simulation.run = NULL,
    h.repetitions = NULL,
    i.repetitions = NULL,
    h.recurrent.parent = NULL,
    i.recurrent.parent = NULL,
    h.donor.parent = NULL,
    i.donor.parent = NULL,
    h.linkage.map = NULL,
    i.linkage.map = NULL,
    h.target.loci = NULL,
    i.target.loci = NULL,
    h.flanking.loci = NULL,
    i.flanking.loci = NULL,
    h.recipient.loci = NULL,
    i.recipient.loci = NULL,
    h.gen.type = NULL,
    i.gen.type = NULL,
    h.population.size = NULL,
    i.population.size = NULL,
    h.sel.strategy = NULL,
    i.sel.strategy = NULL,
    h.no.selected = NULL,
    i.no.selected = NULL,
    h.no.preselected = NULL,
    i.no.preselected = NULL,
    h.reg.chr = NULL,
    i.reg.chr = NULL,
    h.reg.begin = NULL,
    i.reg.begin = NULL,
    h.reg.end = NULL,
    i.reg.end = NULL,
    h.missing.allele = NULL,
    i.missing.allele = NULL,
    h.success.factor = NULL,
    i.success.factor = NULL,
    result.files
)
```

Arguments

`a.simulation.name`, `b.simulation.name`, `c.simulation.name`, `d.simulation.name`, `e.simulation.name`, `f.simulation.name`, `g.simulation.name`, `h.simulation.name`, `i.simulation.name`

: Simulation name for each scenario. Scenario A is required. For scenarios B through I, a non-`NULL` simulation name activates that scenario; a `NULL` value means that scenario is not run.

`a.simulation.run`, `b.simulation.run`, `c.simulation.run`, `d.simulation.run`, `e.simulation.run`, `f.simulation.run`, `g.simulation.run`, `h.simulation.run`, `i.simulation.run`

: Simulation-run identifier passed to `mab.simulate()`. For active scenarios B through I, `NULL` inherits `a.simulation.run`.

`a.repetitions`, `b.repetitions`, `c.repetitions`, `d.repetitions`, `e.repetitions`, `f.repetitions`, `g.repetitions`, `h.repetitions`, `i.repetitions`

: Number of simulation repetitions. Scenario A defaults to 1000. For active scenarios B through I, `NULL` inherits `a.repetitions`.

`a.recurrent.parent`, `b.recurrent.parent`, `c.recurrent.parent`, `d.recurrent.parent`, `e.recurrent.parent`, `f.recurrent.parent`, `g.recurrent.parent`, `h.recurrent.parent`, `i.recurrent.parent`

: Description of the recurrent parent passed to `mab.simulate()`; `"H"` denotes the homozygous-parent mode used by that function. For active scenarios B through I, `NULL` inherits the scenario-A setting.

`a.donor.parent`, `b.donor.parent`, `c.donor.parent`, `d.donor.parent`, `e.donor.parent`, `f.donor.parent`, `g.donor.parent`, `h.donor.parent`, `i.donor.parent`

: Description of the donor parent passed to `mab.simulate()`; it follows the same representation as `recurrent.parent`. For active scenarios B through I, `NULL` inherits the scenario-A setting.

`a.linkage.map`, `b.linkage.map`, `c.linkage.map`, `d.linkage.map`, `e.linkage.map`, `f.linkage.map`, `g.linkage.map`, `h.linkage.map`, `i.linkage.map`

: Name of the linkage-map input file passed to `mab.simulate()`. Scenario A has no default. For active scenarios B through I, `NULL` inherits `a.linkage.map`.

`a.target.loci`, `b.target.loci`, `c.target.loci`, `d.target.loci`, `e.target.loci`, `f.target.loci`, `g.target.loci`, `h.target.loci`, `i.target.loci`

: Character vector of target-locus names passed to `mab.simulate()`, or `NULL`. For active scenarios B through I, `NULL` inherits the scenario-A setting.

`a.flanking.loci`, `b.flanking.loci`, `c.flanking.loci`, `d.flanking.loci`, `e.flanking.loci`, `f.flanking.loci`, `g.flanking.loci`, `h.flanking.loci`, `i.flanking.loci`

: Character vector of flanking-locus names used for recurrent-parent preselection, or `NULL`. For active scenarios B through I, `NULL` inherits the scenario-A setting.

`a.recipient.loci`, `b.recipient.loci`, `c.recipient.loci`, `d.recipient.loci`, `e.recipient.loci`, `f.recipient.loci`, `g.recipient.loci`, `h.recipient.loci`, `i.recipient.loci`

: Character vector of loci used for preselection of recurrent-parent alleles, or

NULL. For active scenarios B through I, NULL inherits the scenario-A setting.

a.gen.type, b.gen.type, c.gen.type, d.gen.type, e.gen.type, f.gen.type, g.gen.type, h.gen.type, i.gen.type
: Character vector describing the generation types passed to mab.simulate() (for example "f1", "bc", "s", and "dh"), or NULL. For active scenarios B through I, NULL inherits the scenario-A setting.

a.population.size, b.population.size, c.population.size, d.population.size, e.population.size, f.population.size, g.population.size, h.population.size, i.population.size
: Numeric vector of population sizes for the generations of the backcrossing program, or NULL. For active scenarios B through I, NULL inherits the scenario-A setting.

a.sel.strategy, b.sel.strategy, c.sel.strategy, d.sel.strategy, e.sel.strategy, f.sel.strategy, g.sel.strategy, h.sel.strategy, i.sel.strategy
: Character vector of selection-strategy codes passed to mab.simulate(), or NULL. See mab.simulate() for the implemented strategies. For active scenarios B through I, NULL inherits the scenario-A setting.

a.no.selected, b.no.selected, c.no.selected, d.no.selected, e.no.selected, f.no.selected, g.no.selected, h.no.selected, i.no.selected
: Numeric vector giving the number of selected individuals by generation, or NULL. For active scenarios B through I, NULL inherits the scenario-A setting.

a.no.preselected, b.no.preselected, c.no.preselected, d.no.preselected, e.no.preselected, f.no.preselected, g.no.preselected, h.no.preselected, i.no.preselected
: Numeric vector giving the minimum number of individuals retained by the flanking-marker preselection step, or NULL. For active scenarios B through I, NULL inherits the scenario-A setting.

a.reg.chr, b.reg.chr, c.reg.chr, d.reg.chr, e.reg.chr, f.reg.chr, g.reg.chr, h.reg.chr, i.reg.chr
: Chromosome numbers for regions evaluated separately for recurrent-parent genome content, or NULL. For active scenarios B through I, NULL inherits the scenario-A setting.

a.reg.begin, b.reg.begin, c.reg.begin, d.reg.begin, e.reg.begin, f.reg.begin, g.reg.begin, h.reg.begin, i.reg.begin
: Beginning positions of the regions specified by reg.chr, in cM from the telomere, or NULL. For active scenarios B through I, NULL inherits the scenario-A setting.

a.reg.end, b.reg.end, c.reg.end, d.reg.end, e.reg.end, f.reg.end, g.reg.end, h.reg.end, i.reg.end
: End positions of the regions specified by reg.chr, in cM from the telomere; a value of zero is interpreted by mab.simulate() as the chromosome end. For active scenarios B through I, NULL inherits the scenario-A setting.

a.missing.allele, b.missing.allele, c.missing.allele, d.missing.allele, e.missing.allele, f.missing.allele, g.missing.allele, h.missing.allele, i.missing.allele
: Character string used by mab.simulate() to identify missing marker data. Sce-

nario A defaults to "9". For active scenarios B through I, NULL inherits the scenario-A setting.

a.success.factor, b.success.factor, c.success.factor, d.success.factor, e.success.factor, f.success.factor, g.success.factor, h.success.factor, i.success.factor

Success factor passed to mab.simulate(). Scenario A defaults to 1. When it is below 1, mab.simulate() draws the number of successful selected individuals from a binomial distribution and retains at least one individual. For active scenarios B through I, NULL inherits the scenario-A setting.

result.files Character vector containing one explicit result-file name for each active scenario, in scenario order A through I. The names must be non-empty and unique. Relative names are resolved against st.data.dir; absolute paths are used directly.

Details

The scenario prefixes a. through i. identify up to nine simulation definitions. Scenario A is always defined by a.simulation.name. A scenario B through I is run only when its simulation.name argument is non-NULL.

For every setting other than simulation.name, an active scenario B through I inherits the corresponding scenario-A value when its own argument is NULL; a non-NULL scenario-specific value overrides the scenario-A value. The resulting settings are passed directly to mab.simulate(). See mab.simulate() for the detailed meaning of the generation and selection settings and for the structure of the simulation results. The file name for every active scenario is taken from result.files; no result-file name is generated automatically.

Value

NULL, returned invisibly as the value of the final for loop. The function is called for its side effect of running the active simulation scenarios; each call to mab.simulate() saves its result in the corresponding explicitly supplied file from result.files.

See Also

mab.simulate, mab.load.data, mab.tabulate, mab.Input.files, mab.Examples

---

mab.df *Helper used by the marker-assisted backcross simulation routines*

Description

Resolves MAB file names against a configured directory while preserving explicit absolute paths.

Usage

```
mab.df(d, f)
```

Arguments

d	Directory used for a relative file name.
f	Non-empty file name or path. Absolute paths are returned unchanged.

Value

A character scalar. If f is absolute it is returned unchanged. If f is relative and d is non-empty, the two are combined with file.path(); otherwise f is returned unchanged.

---

mab.Examples *mab.Examples*

Description

Examples for the optimize-mab routines

Value

No return value. This is a documentation page describing examples for the marker-assisted backcrossing routines.

See Also

mab.simulate, mab.compare, mab.load.data, mab.tabulate, mab.Input.files

---

mab.Input.files *Structure of input files for the optimize-mab routines.*

Description

Structure of the files describing linkage maps and crossing parents. Files *must* be placed in the directory st.input.dir.

Details

A linkage-map file contains chromosome number, position in cM, and marker name. The old package documentation gave, for example:

```
1       0       umc94
1       0002    bnl805
1       0067    umc76
1       0070    umc137
2       0       bnl845
2       0012    umc53
```

For a homozygous crossing parent, the first line is the genotype name and each following line contains a marker name and one allele, for example:

```
dh.2010.12345
s1e2855 ade
s1e3177 cyt
s1e2083 ade
s1e4552 gua
```

For a heterozygous crossing parent, the first line is again the genotype name and each following line contains a marker name and two alleles, for example:

```
bc1i3
umc94   1  8
bnl805  1  8
umc76   8  8
umc137  8  8
```

Value

No return value. This is a documentation page describing the input-file formats used by the marker-assisted backcrossing routines.

See Also

`mab.simulate`, `mab.Examples`

`mab.load.data` *Loads the data of a simulation*

Description

Loads a simulation result from the explicitly supplied `result.file` for further analysis or printing. Relative file names are resolved against `st.data.dir`; absolute paths are used directly.

Usage

```
mab.load.data(simulation.name, simulation.run, result.file)
```

Arguments

`simulation.name`	Simulation-name identifier retained for compatibility with the MAB workflow. It does not determine the file name.
`simulation.run`	Simulation-run identifier retained for compatibility with the MAB workflow. It does not determine the file name.
`result.file`	Required name of the result file. Relative names are resolved against `st.data.dir`; absolute paths are used directly. No file name is generated from `simulation.name` or `simulation.run`.

Value

The object outp loaded from the requested simulation result file. For files created by mab.simulate(), this is a named list containing the simulation parameters and, depending on the requested analyses, the following summaries:

TargetAlleles Target-allele summary for selected plants over generations.

RPG.gw Recurrent-parent genome content over the whole genome.

RPG.reg1, RPG.reg2, ... Recurrent-parent genome content for separately defined chromosome regions.

DonorSegments.gw Donor chromosome segments in selected individuals, counted per homologue.

DonorSegments.reg1, DonorSegments.reg2, ... Donor chromosome segments in separately defined chromosome regions.

LinkageDrag1, LinkageDrag2, ... Donor chromosome segment attached to each target locus; target loci are evaluated separately and linkage drag is averaged over the two homologues.

SM Number of marker data points attributed to single-marker assays.

HT Number of marker data points attributed to high-throughput assays.

See mab.simulate for the selection-strategy rules used to count SM and HT marker data points.

See Also

mab.simulate, mab.tabulate, mab.Examples

---

mab.save.inds *Helper used by the marker-assisted backcross simulation routines*

Description

Writes one marker-data file for each individual in a population. Output file names use the explicitly supplied file.prefix.

Usage

```
mab.save.inds(pop, pname, file.prefix)
```

Arguments

pop	Population name or population specification.
pname	Prefix used for the individual names written inside the marker-data files.
file.prefix	Required prefix for the output file names. Each file receives a three-digit individual number and the extension .mda. Relative prefixes are resolved against st.output.dir; absolute paths are used directly.

Value

NULL. The function is called for its side effect of writing one marker-data file for each individual in the supplied population.

mab.simulate *Simulates a marker-assisted backcrossing program*

Description

Simulates one marker-assisted backcrossing program. The simulation result is stored in the explicitly supplied `result.file`. A relative file name is resolved against `st.data.dir`; an absolute path is used as supplied.

Usage

```
mab.simulate(simulation.name, simulation.run, repetitions = 1000,
             recurrent.parent = "H", donor.parent = "H", linkage.map,
             target.loci = NULL, flanking.loci = NULL, recipient.loci = NULL,
             gen.type = NULL, population.size = NULL, sel.strategy = NULL,
             no.selected = NULL, no.preselected = NULL, reg.chr = NULL,
             reg.begin = NULL, reg.end = NULL, missing.allele = "9",
             result.file, success.factor = 1, recode.infiles = FALSE,
             recode.file = NULL)
```

Arguments

`simulation.name`
: Name of the simulation stored in the returned result object. The argument is required and has no default value.

`simulation.run`
: Name of the simulation run. Usually a single character. This can be used to run a certain simulation for several times. For example, in a test run with 1000 repetitions, the character `x` is used. And in a final run with 50000 repetitions the character `f` is used. The argument is required and has no default value.

`repetitions`
: Number of repetitions of the simulation. The default is 1000. For final analyses, a larger number of repetitions may be used.

`recurrent.parent`
: Description of the recurrent parent. If `H` is given, then a homozygous parent is assumed. Polymorphism between donor and recurrent parent is assumed for all markers on the linkage map. If the name of a file is given, the genotype described by this file is used. The file must be located in the directory `st.input.dir`. Examples are given in the section 'Input Files'. The argument is optional. Default value is `H`. *Note:* `recurrent.parent` and `donor.parent` need to be specified consistently. Either for both a `H`, or for both a file name.

`donor.parent`
: Description of the donor parent. See `recurrent.parent`.

`linkage.map`
: Name of the file, in which the linkage map is described. See the section 'Input Files' for a description of the file structure.The file must be located in the directory `st.input.dir`. The argument is required and has no default value.

`target.loci`
: Character vector with names of target loci. Selection for the donor allele at `target.loci` is indicated by a `t` in the definition of a selection strategy. The default is `NULL`.

`flanking.loci` Character vector with names of loci, which are used to carry out pre selection for the recurrent parent allele before genome wide background selection is carried out. Selection for the recipient allele at `flanking.loci` is indicated by an `f` in the definition of a selection strategy. `flanking.loci` are used to reduce the length of the chromosome segment attached to the target genes. In general, they are flanking the target loci, but this in no requirement. For example, loci at a defined map distance from the target loci can be used. The default is `NULL`.

`recipient.loci` Character vector, which contains the names of loci wich are used to carry out preselection for the recipient alleles. Selection for the recipient allele at `recipient.loci` is indicated by `r` in the definition of a selection strategy. `recipient.loci` are used to define chromosome regions of the recipient, which must be present in the backcrossing product. The default is `NULL`.

`gen.type` A character vector consisting of a description of the generation types of the backcrossing program. The following generation types are implemented:

`f1`	F1 population
`bc`	Backcross population
`s`	Selfing population
`dh`	Doubled haploids

`population.size`
Numeric vector containing the population sizes of the generations in the backcrossing program. The formal default is `NULL`.

`sel.strategy` Character vector consisting of the selection strategies for the different generations of a backcrossing program. The following selection strategies are implemented. SI: selection index.

Strategy	Step	Description
`n`	1	Random selection of individuals
`t`	1	SI: Donor alleles at target loci
`tb`	1	SI: Donor alleles at target loci
	2	SI: Recurrent parent alleles genome wide
`tfb`	1	SI: Donor alleles at target genes
	2	SI: Recurrent parent alleles at flanking loci
	3	SI: Recurrent parent alleles genome wide
`tf`	1	SI: Donor alleles at target loci
	2	SI: Recurrent parent alleles flanking loci
`tfs`	1	SI: Donor alleles at target genes
	2	SI: Recurrent parent alleles at flanking loci
	3	SI: Low number of donor chromosome sgements
`tr`	1	SI: Donor alleles at target loci
	2	SI: Recurrent parent alleles at recipient loci
`trb`	1	SI: Donor alleles at target loci
	2	SI: Recurrent parent alleles at recipient loci
	3	SI: Recurrent parent alleles genome wide
`trfb`	1	SI: Donor alleles at target loci
	2	SI: Recurrent parent alleles at recipient loci
	3	SI: Recurrent parent alleles flanking loci

	4	SI: Recurrent parent alleles genome wide
tfrb	1	SI: Donor alleles at target loci
	2	SI: Recurrent parent alleles at recipient loci
	3	SI: Recurrent parent alleles genome wide
	4	SI: Recurrent parent alleles flanking loci

Selection strategies tb and tfb are the two-stage and three-stage selection strategies described by Frisch et al. (1999).

Counting marker data points:

t: No markers are counted for the foreground selection for the target genes(s). (This selection step requires the same number of marker analyses for all selection stragegies.)

f: Pre selection for markers flanking the target genes is assumed to be carries out with single marker assays. For each individual marker locus, a marker data point is counted. In advanced generations of a backcrossing program, only such markers are assayed, which were not already fixed for the recurrent parent allele in the non-recurrent parent (= the plant selected in the previous generation).

b: Background selection is assumed to be carried out with high throughput assays. For each plant one marker assay is counted for all loci on the linkage map (irrespective of their marker genotype).

s: Selection for a low number of donor segments is assumed to be carried out with high throughput assays. For each plant one marker assay is counted for all loci on the linkage map.

r: Pre selection for recipint alleles ia assumed to be carried out with high HT assays. For strategies tr, trb, trfb, one HT assay is counted for each plant carrying the target genes. For tfrb, the selection for the flanking markers is assumed to be carried out with single marker assays.

`no.selected` Numerical vector specifying the number of selected individuals in the backcross generations. *Note:* Specifying values larger than one makes only sense for some selection strategies. The argument is optional, default value is a vector consisting of ones.

`no.preselected` Numerical vector specifying the minimum number of individuls that are preselected when selection for flanking marker (step "f") is carried out. *Note:* Only used for selection strategies including preselection for flanking markers. The argument is optional. If not specified, all individuals are preselected that show recombination at the maximum number of flanking markers.

`reg.chr` Definition of chromosome regions, which are evaluated separately for the recurrent parent genome. `reg.chr` defines the chromosome number of the region to be evaluated. The default is `NULL`.

`reg.begin` Definition of chromosome regions, which are evaluated separately for the recurrent parent genome. `reg.begin` defines the begin of the region to be evaluated in cM distance from the telomere. The default is `NULL`. If `reg.chr` is specified, a corresponding value is required.

`reg.end` End of the region to be evaluated, in cM distance from the telomere. If 0 is given, the entire chromosome is evaluated. The default is `NULL`. If `reg.chr` is specified, a corresponding value is required.

`missing.allele` Character string that specifies missing marker data. The default is `"9"`.

`result.file` Required name of the file in which the simulation result is stored. Relative names are resolved against `st.data.dir`; absolute paths are used directly. No file name is generated automatically.

`success.factor` Numeric success probability used after selection. The default is 1. If it is below 1, the number of successful selected individuals is drawn from a binomial distribution and at least one individual is retained.

`recode.infiles` Logical. If `TRUE`, recoded parental marker data are written to the explicitly supplied `recode.file`.

`recode.file` File name for recoded parental marker data when `recode.infiles = TRUE`. It is required in that case. Relative names are resolved against `st.output.dir`; absolute paths are used directly.

Value

On successful completion, invisibly returns a named `list` containing both the simulation definition and the simulation summaries. The parameter part includes `simulation.name`, `simulation.run`, `repetitions`, `recurrent.parent`, `donor.parent`, `linkage.map`, `target.loci`, `flanking.loci`, `recipient.loci`, `gen.type`, `population.size`, `sel.strategy`, `no.selected`, `no.preselected`, `reg.chr`, `reg.begin`, `reg.end`, and `success.factor`.

Depending on the requested analyses, the result part can contain:

`TargetAlleles` Summary of the frequency of target alleles in the selected plants over the generations of the backcrossing program.

`RPG.gw` Summary of recurrent-parent genome content of the selected plants over the whole genome.

`RPG.reg1`, `RPG.reg2`, ... Recurrent-parent genome content in chromosome regions defined by `reg.chr`, `reg.begin`, and `reg.end`; the suffix identifies the region.

`DonorSegments.gw` Number of donor chromosome segments in the selected individuals over the whole genome. Segments are counted per homologue, so the two homologues of each chromosome in a diploid individual are counted separately.

`DonorSegments.reg1`, `DonorSegments.reg2`, ... Number of donor chromosome segments within separately evaluated chromosome regions.

`LinkageDrag1`, `LinkageDrag2`, ... Length of the donor chromosome segment attached to each target locus. Target loci are evaluated separately and linkage drag is averaged over the two homologues.

`SM` Summary of the number of marker data points attributed to single-marker assays. In the implemented accounting these arise from selection steps using flanking-marker assays.

`HT` Summary of the number of marker data points attributed to high-throughput assays, including genome-wide background selection and selection for a low number of donor segments as specified by the selection strategy.

Validation failures return `NULL` invisibly. The same list is saved in the explicitly supplied `result.file`.

See Also

`mab.compare`, `mab.load.data`, `mab.tabulate`, `mab.Input.files`, `mab.Examples`

mab.tabulate *Tabulate the results of a series of simulations*

Description

Tabulates results from a series of simulations. The files to load are supplied explicitly in `result.files`.

Usage

```
mab.tabulate(simulation.names.runs, par.summary = TRUE, ext.summary = FALSE,
             RPG.gw.mean = FALSE, TargetAlleles = FALSE, RPG.gw.Q10 = FALSE,
           RPG.reg.Q10 = FALSE, RPG.reg.mean = FALSE, DonorSegments.gw = FALSE,
         DonorSegments.reg = FALSE, LinkageDrag = FALSE, MarkerAnalyses = FALSE,
            digits = 1, percent = TRUE, result.files)
```

Arguments

`simulation.names.runs`	Character vector containing alternating simulation names and simulation-run identifiers. It is interpreted as a two-column matrix, one row per simulation result to be tabulated.
`par.summary`	Logical. Include a summary of the main simulation parameters.
`ext.summary`	Logical. Include an extended summary of the simulation parameters.
`RPG.gw.mean`	Logical. Include mean genome-wide recurrent-parent genome (RPG) values.
`TargetAlleles`	Logical. Include the mean target-allele summary.
`RPG.gw.Q10`	Logical. Include the Q10 value of the genome-wide RPG.
`RPG.reg.Q10`	Logical. Include Q10 values of RPG in the defined chromosome regions.
`RPG.reg.mean`	Logical. Include mean RPG values in the defined chromosome regions.
`DonorSegments.gw`	Logical. Include the mean number of donor segments genome-wide.
`DonorSegments.reg`	Logical. Include the mean number of donor segments in the defined chromosome regions.
`LinkageDrag`	Logical. Include mean linkage-drag summaries for the target loci.
`MarkerAnalyses`	Logical. Include mean single-marker (SM) and high-throughput (HT) marker-analysis counts.
`digits`	Number of decimal places used when rounding numerical summaries.
`percent`	Logical. If `TRUE`, RPG summaries are multiplied by 100; if `FALSE`, they remain on the proportion scale.
`result.files`	Character vector containing one explicit result-file name for each scenario in `simulation.names.runs`, in the same order. Relative names are resolved against `st.data.dir`; absolute paths are used directly.

Value

A named list of the summary matrices requested by the logical arguments. Possible components include par.summary, ext.summary, TargetAlleles.mean, RPG.gw.mean, RPG.gw.Q10, numbered regional RPG.reg*.mean and RPG.reg*.Q10 matrices, DonorSegments.gw.mean, numbered DonorSegments.reg*.mean matrices, numbered LinkageDrag*.mean matrices, SM.mean, and HT.mean. If incompatible scenario definitions are detected, NULL is returned invisibly.

See Also

mab.simulate, mab.load.data, mab.compare, mab.Examples

---

masbc.input *Input Files for Marker-Assisted Backcrossing Examples*

Description

A collection of linkage maps and parental marker-genotype files used by the marker-assisted backcrossing examples. The files are stored as an R data object so that they can be loaded with data() and written to a temporary directory when the file-based mab.* interface is used.

Usage

```
data(masbc.input)
```

Format

A named list of 16 character vectors. Each list element contains the exact text lines of one input file. The element names are the file names: ex-mab1.map, ex-mab2.map, ex-mab3.map, ex-mab4.map, ex-mabi1.mda through ex-mabi8.mda, ex-mabp1.mda, ex-mabp1_m.mda, ex-mabp2.mda, and ex-mabp2_m.mda.

Details

The *.map elements contain linkage-map input files. The *.mda elements contain marker genotypes for donor, recurrent-parent, or individual backcross plants. Keeping the files as character vectors preserves the original input-file representation while making the complete example data available through the standard R data() mechanism.

The marker-assisted backcrossing functions still use file names. Therefore, a vignette or analysis can load masbc.input, write the required list elements to tempdir(), and set st.input.dir to that temporary directory.

Source

Input files used by the marker-assisted backcrossing examples supplied with SelectionTools.

See Also

mab.simulate, mab.compare, mab.load.data, mab.tabulate, mab.Input.files

Examples

```
data("masbc.input")

mab.input.dir <- file.path(tempdir(), "masbc-input")
dir.create(mab.input.dir, showWarnings = FALSE)

invisible(Map(
    function(x, file) writeLines(x, file.path(mab.input.dir, file)),
    masbc.input,
    names(masbc.input)
))

list.files(mab.input.dir)
unlink(mab.input.dir, recursive = TRUE)
```

Md_technow *Technow Dent Marker Data*

Description

Marker data for the dent parental lines corresponding to the hybrid phenotype data in DT_technow. The data are used to construct genomic relationship matrices and for genomic-prediction examples.

Usage

```
data(Md_technow)
```

Format

A marker-data matrix or data frame with dent parental lines in rows and markers in columns. Row names identify the parental lines and the entries contain marker genotypes.

Source

Data supplied with SelectionTools for genomic prediction examples. Original data published in Technow et al. (2014): doi:10.1534/genetics.114.165860.

Mf_technow *Technow Flint Marker Data*

Description

Marker data for the flint parental lines corresponding to the hybrid phenotype data in DT_technow. The data are used to construct genomic relationship matrices and for genomic-prediction examples.

Usage

```
data(Mf_technow)
```

Format

A marker matrix with flint parental lines in rows and markers in columns. Row names identify the parental lines and the entries contain marker genotypes.

Source

Data supplied with SelectionTools for genomic prediction examples. Original data published in Technow et al. (2014): doi:10.1534/genetics.114.165860.

---

optimize.population *Optimize a population using the package selection routines*

Description

Optimize a population using the package selection routines.

Usage

```
optimize.population(PopNames)
```

Arguments

PopNames	Population name or population specification.

Value

Returns the same invisible implementation-level `list` as `population.optimize()`. The function is a compatibility wrapper and the meaningful result is the delegated population operation.

---

ph.linkage.map.create *Create a linkage map for phenotype or simulation use*

Description

Create a linkage map for phenotype or simulation use.

Usage

```
ph.linkage.map.create(map, disperse=FALSE, file.disperse=NA,
                      disperse.factor=100)
```

Arguments

`map`	Linkage map or map object.
`disperse`	Function argument used by this operation.
`file.disperse`	Function argument used by this operation.
`disperse.factor`	Function argument used by this operation.

Value

When `disperse = TRUE`, returns the modified map object after positions have been dispersed. With the default `disperse = FALSE`, returns `NULL`; in both cases the main side effect is installing the map through the compiled map-definition routine.

`phenotype.population` *Generate phenotypic values for a simulation population*

Description

Generates phenotypic values for all loaded effects and all active individuals of a simulation population.

Usage

```
phenotype.population(PopName, hsq)
```

Arguments

`PopName`	Character string naming one simulation population.
`hsq`	Numeric vector containing one heritability for each loaded effect, in the order returned by `list.effects()`. Each value must be greater than zero and not greater than one.

Details

Phenotypic values are generated for all loaded effects in one operation. Partial phenotypic evaluation is not supported. The length of `hsq` must therefore equal the number of loaded effects.

The function first genotypes and evaluates the population. For effect i, let $V_{G,i}$ be the sample variance of the effect-specific genetic values and let h_i^2 be the supplied heritability. The residual variance is

$$V_{E,i} = V_{G,i}/h_i^2 - V_{G,i}.$$

For every individual an independent normal residual with variance $V_{E,i}$ is added to the effect-specific genetic value.

The phenotypic value array is parallel to the genetic value array. Elements 1 through the number of effects contain the effect-specific phenotypic values. Element 0 contains the weighted phenotypic

selection index, calculated with the current effect weights in the same way as the genetic selection index.

Phenotypic values are population-wide derived data. Re-evaluating genetic values, explicitly setting genetic values, removing an evaluation, removing cached genotypes, changing the effect configuration or effect weights, or removing the linkage map invalidates stored phenotypic values.

Value

On successful phenotype generation, an invisible implementation-level `list` returned by the compiled routine; its `retval` component is the status code. Validation failures return `NULL` invisibly.

See Also

`evaluate.population`, `get.population.pvalue`, `population.sort`, `select.n.best`, `select.all.best`

---

plabsim *Perform the SelectionTools operation "plabsim"*

Description

Perform the SelectionTools operation "plabsim".

Format

A list containing PLABSIM run-time state and parameters.

Value

A `list` containing the current PLABSIM run-time state and parameters.

---

plabsim.init *Initialize the PLABSIM simulation state*

Description

Initialize the PLABSIM simulation state.

Usage

```
plabsim.init()
```

Value

Returns the same invisible implementation-level `list` as `rng.init()`; the main effect is initializing the simulation random-number generator state.

plabsim.R.date *Perform the SelectionTools operation "plabsim R date"*

Description

Perform the SelectionTools operation "plabsim R date".

Format

A character string containing the PLABSIM R-code date.

Value

A character scalar containing the PLABSIM R-code date.

plabsim.R.version *Perform the SelectionTools operation "plabsim R version"*

Description

Perform the SelectionTools operation "plabsim R version".

Format

A character string containing the PLABSIM R-code version.

Value

A character scalar containing the PLABSIM R-code version.

plabsim.version *Perform the SelectionTools operation "plabsim version"*

Description

Perform the SelectionTools operation "plabsim version".

Usage

```
plabsim.version()
```

Value

Returns the same invisible implementation-level `list` as `write.version.2()`; the main effect is reporting package version information.

plant *Create or process a plant object in the simulation framework*

Description

Create or process a plant object in the simulation framework.

Usage

```
plant(genotype, NoInd=1)
```

Arguments

genotype	Function argument used by this operation.
NoInd	Function argument used by this operation.

Value

A data.frame with columns ind, chrom, hom, pos, and all, representing the requested genome from the supplied homologue genotypes.

population.append *Append a copy of a simulation population*

Description

Appends a copy of a source population to the end of a destination population.

Usage

```
population.append(NameP1, NameP2)
```

Arguments

NameP1	Character string naming the recipient population.
NameP2	Character string naming the source population, which is not modified.

Details

Raw chromosome data are copied for every appended individual. If the destination is initially empty, complete genotype, genetic-value, and phenotypic-value categories are copied when present in the source. For a nonempty destination, a derived category is retained only when it is complete in both populations. Otherwise that category is cleared in the result.

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of appending a copy of one simulation population to another.

`population.concat` *Concatenate Simulation Populations*

Description

Moves all individuals of one population into another simulation population.

Usage

```
population.concat(NameP1, NameP2)
```

Arguments

`NameP1`	Character string naming the receiving population. If it does not already exist, it is created.
`NameP2`	Character string naming the population to concatenate and remove.

Details

Individuals already present in `NameP1` retain their order and remain at the beginning of the receiving population. The active individuals from `NameP2` are appended after them. Code that depends on population order can therefore, for example, concatenate a selected fraction first and the remaining individuals second and subsequently identify the two groups by position.

Individuals are transferred as complete individual structures and `NameP2` is removed. As in the existing implementation, cached genotype and genetic evaluation data are discarded before concatenation. Phenotypic values are also discarded because they depend on the genetic evaluation.

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of concatenating simulation populations.

See Also

`population.append`, `population.copy`, `population.divide`

population.copy *Copy a simulation population*

Description

Copies all or a contiguous subset of a source population into a destination population.

Usage

```
population.copy(NameP1, NameP2, start=1, n=-1)
```

Arguments

NameP1	Character string naming the destination population.
NameP2	Character string naming the source population.
start	One-based position of the first source individual to copy.
n	Number of individuals to copy. A negative value requests the remainder of the source population.

Details

The source population is not modified. Raw chromosome data are copied. Complete cached genotype, genetic-value, and phenotypic-value categories are deep-copied when present in the source. Derived categories are maintained as all-or-none population data; allocation failure cannot leave only a subset of active destination individuals with a category.

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of copying a simulation population.

population.divide *Divide a simulation population*

Description

Moves the first requested individuals of a source population into a new population.

Usage

```
population.divide(NameP1, NameP2, NoI=-1)
```

Arguments

NameP1	Character string naming the destination population.
NameP2	Character string naming the source population.
NoI	Number of individuals to move.

Details

The operation moves complete individual structures. Therefore chromosome data, information fields, genotype data, genetic values, and phenotypic values remain attached to the same individuals. No value category is recalculated merely because the population is divided.

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of dividing a simulation population.

---

population.exist *Check whether a simulation population exists*

Description

Checks whether the requested population name occurs in the current simulation population list.

Usage

```
population.exist(PopName)
```

Arguments

PopName	Name of the population.

Value

An invisible logical scalar indicating whether PopName occurs in the current population list.

population.individual.remove *Individual remove a population or population data*

Description

Individual remove a population or population data.

Usage

```
population.individual.remove(PopName, IndPos)
```

Arguments

PopName	Name of the population.
IndPos	Function argument used by this operation.

Value

For a valid individual position, the invisible implementation-level list returned by the final population operation used to remove the individual. If the position is outside the population, NULL. The main effect is modification of the population.

population.info.get *Return stored information for an individual*

Description

Returns the character information stored for an individual in a simulation population.

Usage

```
population.info.get(PopName, ind)
```

Arguments

PopName	Name of the population.
ind	Individual index or indices.

Value

A character value containing the stored information for the requested individual when the individual index is valid; otherwise NULL.

population.info.set *Set stored information for an individual*

Description

Sets the character information stored for an individual in a simulation population.

Usage

```
population.info.set(PopName, ind, info)
```

Arguments

PopName	Name of the population.
ind	Individual index or indices.
info	Information value.

Value

For a valid individual index, an invisible implementation-level `list` returned by the compiled setter. For an invalid index, `NULL`. The main effect is updating the stored individual information.

population.list *List a population or population data*

Description

Lists the currently registered simulation populations and their active sizes.

Usage

```
population.list()
```

Details

The returned population registry is independent of SelectionTools marker-data-set names. The count associated with each name is the active number of individuals, not necessarily the allocated storage capacity.

Value

A `data.frame` with columns `PopName` and `count`, giving each current population name and its number of individuals.

`population.matrix.load` *Matrix load a population or population data*

Description

Matrix load a population or population data.

Usage

```
population.matrix.load(file, backcross=FALSE, keep.IndID=TRUE)
```

Arguments

`file`	File name or file connection.
`backcross`	Function argument used by this operation.
`keep.IndID`	Function argument used by this operation.

Value

`NULL`. The function is called for its side effect of loading matrix-format population data and, when requested, restoring individual identifiers.

`population.matrix.save` *Matrix save a population or population data*

Description

Matrix save a population or population data.

Usage

```
population.matrix.save(file, PopNames, missing.rm=TRUE, keep.IndID=TRUE)
```

Arguments

`file`	File name or file connection.
`PopNames`	Population name or population specification.
`missing.rm`	Function argument used by this operation.
`keep.IndID`	Function argument used by this operation.

Value

`NULL`, returned invisibly by the file-writing operation. The function is called for its side effect of writing matrix-format population data.

population.name.swap *Swap the names of two simulation populations*

Description

Swaps the registered names of two simulation populations.

Usage

```
population.name.swap(NameP1, NameP2)
```

Arguments

NameP1	Name of the first parental population.
NameP2	Name of the second parental population.

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of swapping simulation-population names.

population.optimize *Optimize storage for simulation populations*

Description

Calls the compiled population-optimization routine for the requested simulation populations.

Usage

```
population.optimize(PopNames)
```

Arguments

PopNames	Population name or population specification.

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of optimizing the requested simulation populations.

population.plabsim.load

Load a population from PLABSIM-style data

Description

Loads a simulation population from the PLABSIM-style file representation and restores stored individual information.

Usage

```
population.plabsim.load(file, PopName=NA)
```

Arguments

file	File name or file connection.
PopName	Name of the population.

Value

NULL. The function is called for its side effect of loading a population from the PLABSIM-style representation and restoring stored individual information.

---

population.plabsim.save

Save a population in PLABSIM-style format

Description

Writes a simulation population and its stored individual information in the PLABSIM-style file representation.

Usage

```
population.plabsim.save(file, PopName)
```

Arguments

file	File name or file connection.
PopName	Name of the population.

Value

NULL. The function is called for its side effect of writing the population and individual information in the PLABSIM-style representation.

population.remove *Remove a population or population data*

Description

Removes one or more simulation populations.

Usage

```
population.remove(PopNames)
```

Arguments

PopNames	Population name or population specification.

Details

The named populations are deleted from the simulation population registry and their owned chromosome, information, genotype, and evaluation storage is released. Other simulation-wide definitions such as the genome, linkage map, and effects are separate and are not removed by this operation.

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of removing simulation populations.

population.remove.all *Remove all a population or population data*

Description

Removes all currently registered simulation populations.

Usage

```
population.remove.all()
```

Details

The function obtains the current population list and removes every listed population. This clears population-owned data but does not by itself redefine genome parameters, linkage maps, or effect maps.

Value

An invisible implementation-level `list` returned by the final information-message call after all populations have been removed. The meaningful result is the side effect of removing all populations.

population.rename *Rename a simulation population*

Description

Renames a registered simulation population.

Usage

```
population.rename(OldName, NewName)
```

Arguments

OldName	Function argument used by this operation.
NewName	New population name. The name must contain at least two and fewer than 256 characters.

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of renaming a simulation population.

population.resize *Resize a population or population data*

Description

Changes the allocated size of a simulation population.

Usage

```
population.resize(PopName, newSize)
```

Arguments

PopName	Character string naming the population to resize.
newSize	Requested new population allocation.

Details

A successful change of population allocation invalidates cached marker genotypes and evaluated genetic values because the active/storage relationship has changed. If no actual resize occurs, these derived categories are not cleared. If an attempted growth fails and the original allocation remains unchanged, the existing population and its derived data are preserved.

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of resizing a simulation population.

`population.sample` *Sample a simulation population*

Description

Creates a destination population by random sampling from a source population.

Usage

```
population.sample(NameP1, NameP2, size=-1, replace=FALSE)
```

Arguments

`NameP1`	Character string naming the sampled destination population.
`NameP2`	Character string naming the source population.
`size`	Requested sample size. A negative value requests the source population size.
`replace`	Logical. If true, sampling is with replacement.

Details

The source population is not changed. Raw chromosome data are copied for the sampled individuals. Complete cached genotype, genetic-value, and phenotypic-value categories are deep-copied when present in the source. Derived categories remain all-or-none for the destination population.

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of sampling a simulation population.

population.size.get *Return the size and allocated capacity of a simulation population*

Description

Returns the active number of individuals and the number of individual slots allocated for a simulation population.

Usage

```
population.size.get(PopName)
```

Arguments

PopName	Name of the population.

Value

A one-row data.frame with integer columns NoInds (active number of individuals) and NoIndsAlloc (number of individuals for which memory is allocated). Returns NULL invisibly if the population is not available.

population.sort *Sort a simulation population by genetic or phenotypic value*

Description

Sorts the active individuals of a simulation population by a stored genetic or phenotypic value.

Usage

```
population.sort(PopName, decreasing=TRUE,
                selection.criterion="G", effect=NULL)
```

Arguments

PopName	Character string naming the population to sort.
decreasing	Logical. If true, larger values are placed first.
selection.criterion	Character string. "G" selects genetic values and is the default. "P" selects phenotypic values.
effect	Optional character string naming an effect. If omitted, element 0 of the selected value array is used.

Details

With `selection.criterion="G"`, the population is sorted by `GValue[0]`. With `selection.criterion="P"`, the corresponding phenotypic value is used. If `effect` is omitted, element 0 is the current weighted selection index. If an effect is supplied, its effect-specific value is used.

All active individuals must have the requested value category. Sorting reorders complete individual structures, so chromosome data, genotype data, genetic values, phenotypic values, and information fields remain attached to the same individual.

Value

An invisible implementation-level `list` returned by the compiled sorter. Its `retval` component is the sort status; the main effect is reordering the requested population.

See Also

`phenotype.population`, `select.n.best`, `select.all.best`

`population.swap.name` *Swap simulation population names*

Description

Compatibility wrapper for swapping the names of two simulation populations.

Usage

```
population.swap.name(NameP1, NameP2)
```

Arguments

`NameP1`	Name of the first parental population.
`NameP2`	Name of the second parental population.

Value

Returns the same invisible implementation-level `list` as `population.name.swap()`. The function is a compatibility wrapper and the meaningful result is the delegated population operation.

population.transfer *Transfer a population or population data*

Description

Transfers the individuals of one simulation population into another, either retaining or deleting the source population.

Usage

```
population.transfer(NameP1, NameP2, population.NameP2.delete=TRUE)
```

Arguments

NameP1	Character string naming the recipient population.
NameP2	Character string naming the source population.
population.NameP2.delete	Logical. If true, remove the source after transfer by concatenation; if false, retain the source and append a copy.

Details

If population.NameP2.delete=TRUE, the operation uses population concatenation: the individuals of NameP2 are added to NameP1 and NameP2 is removed. If false, population append is used: the same individuals are copied to NameP1 while NameP2 remains available. Derived genotype and genetic-value state follows the consistency rules of the underlying append or concatenate operation.

Value

The invisible implementation-level list returned by population.concat() or population.append(), depending on population.NameP2.delete. The main effect is transfer of population data.

read.vcf *Read a Beagle-Oriented VCF into SelectionTools*

Description

Reads diploid genotype data from a VCF file directly into a SelectionTools marker data set. The function is intentionally focused on interchange with Beagle rather than on preserving every VCF annotation.

Usage

```
read.vcf(vcffile, data.set = "default")
```

Arguments

vcffile	Character string giving the input VCF file. Files ending in .gz are decompressed with the documented base-R gzfile() connection to a temporary plain VCF before the native parser is called.
data.set	Character string naming the SelectionTools marker data set to create or replace.

Details

The native parser reads the VCF GT subfield and ignores other FORMAT, INFO, QUAL, and FILTER information. Diploid phased (|) and unphased (/) GT fields are accepted. The two allele indices are stored in their input order. A marker data set is recorded as phased only when genotype separators are consistently |; mixed | and / input is accepted but recorded as unphased. If either allele of a diploid genotype is missing, the complete diploid genotype is stored as missing.

VCF REF/ALT allele indices are normalized to the compact SelectionTools allele codes used internally. Marker and sample names are retained. The VCF chromosome labels are represented by consecutive SelectionTools chromosome numbers in first-appearance order.

Files written by write.vcf() contain the metadata line SelectionToolsPositionScale=1000000. For ordinary Beagle-style VCF input without this metadata, physical positions are interpreted with the Beagle no-map convention of one centiMorgan per megabase. Consequently this reader is intended for Beagle interchange, not for reconstructing an arbitrary external genetic map from physical coordinates.

No external compression library is linked by SelectionTools. Gzip transport is handled only by base R; plain VCF parsing is implemented in C.

Value

No return value. The requested SelectionTools marker data set is created or replaced for its side effect.

See Also

write.vcf, st.phase, st.switch.error

---

remove.all.populations

remove.all.populations()

Description

Removes all currently registered simulation populations.

Usage

```
remove.all.populations()
```

Details

The current population list is collected and all listed populations are removed. Genome parameters, linkage-map definitions, effect definitions, and marker-data sets are separate objects and are not removed with the simulation populations.

Value

Returns invisible NULL. The main effect is removal of all populations.

See Also

For vector based input of loci, see `evaluate.allele.freq2`.

`remove.effmaps` *remove.effmaps()*

Description

Deletes all loaded effects

Usage

```
remove.effmaps()
```

Value

Returns the same invisible implementation-level `list` as `effmap.remove.all()`; the main effect is removal of all effect maps.

`remove.evaluate.population` *Remove genetic and phenotypic evaluation values*

Description

Clears stored genetic and phenotypic values for one or more simulation populations.

Usage

```
remove.evaluate.population(PopNames)
```

Arguments

`PopNames`	Population name or population specification.

Details

Population membership, chromosome segments, and cached marker genotypes are retained. Both genetic values and phenotypic values are discarded. Genetic values must be evaluated or set again before genetic selection, and phenotypes must be generated again before phenotypic selection.

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of removing stored evaluation values.

`remove.genotype.population` *Remove cached genotype information from a population*

Description

Clears cached marker-genotype data for one or more simulation populations.

Usage

```
remove.genotype.population(PopNames)
```

Arguments

`PopNames`	Population name or population specification.

Details

The raw chromosome-segment representation is retained. Existing genetic values are retained. Stored phenotypic values are discarded, because phenotypic values are tied to the current genetic/genotype state. Genotype-dependent operations must regenerate marker genotypes from the current linkage map and chromosome data.

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of removing cached genotype information.

`remove.map` *Remove the linkage map from the simulation*

Description

Removes the current linkage map and map-dependent cached simulation data.

Usage

```
remove.map()
```

Details

Removing the linkage map clears cached marker genotypes, genetic values, phenotypic values, and loaded effect definitions because these data depend on the current map/effect configuration. Population chromosome-segment data are retained according to the simulation backend.

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of removing the current linkage map.

`remove.population` *remove.population()*

Description

Removes one or more simulation populations and all data owned by them.

Usage

```
remove.population(PopNames)
```

Arguments

`PopNames`	Names of the populations to be deleted

Details

Removing a population frees its chromosome representation and associated per-population storage, including cached genotype and evaluation data. Multiple population names can be supplied.

Value

Returns the same invisible implementation-level `list` as `population.remove()`. The function is a compatibility wrapper and the meaningful result is the delegated population operation.

rename.population *rename.population()*

Description

Changes the registered name of a simulation population without changing its individuals.

Usage

```
rename.population(OldName, NewName)
```

Arguments

OldName	Name of the population to be renamed
NewName	New name of the population

Details

The population object and its chromosome, genotype, evaluation, and information data remain the same; only the population identifier changes.

Value

Returns the same invisible implementation-level `list` as `population.rename()`. The function is a compatibility wrapper and the meaningful result is the delegated population operation.

reset.all *reset.all()*

Description

"Resets" the program into the initial state.

Usage

```
reset.all()
```

Value

Returns invisble `NULL`. The main effect is resetting all package simulation state handled by the function.

Note

All informations are lost, besides the genome parameter (`genome.parameter.set`).

reset.mdp *Reset marker-assisted donor-parent information*

Description

Reset marker-assisted donor-parent information.

Usage

```
reset.mdp()
```

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of resetting marker-assisted donor-parent information.

resize.population *Resize a population*

Description

Changes the allocated size of a simulation population.

Usage

```
resize.population(PopName, newSize)
```

Arguments

`PopName`	Character string naming the population to resize.
`newSize`	Requested new population allocation.

Details

A successful change of allocation invalidates cached marker genotypes and evaluated genetic values because these derived categories must remain complete and consistent for the active population. If no actual resize occurs, the cached derived data are left unchanged. If an attempted growth fails and the original allocation remains unchanged, the original population state is preserved.

Value

Returns the same invisible implementation-level `list` as `population.resize()`. The function is a compatibility wrapper and the meaningful result is the delegated population operation.

`resources` *Return or display resource information used by SelectionTools*

Description

Return or display resource information used by SelectionTools.

Usage

```
resources(expr)
```

Arguments

`expr`	Function argument used by this operation.

Value

Invisibly returns the value produced by evaluating `expr`; its class and structure are therefore exactly those returned by `expr`. Resource/timing information is reported as a side effect.

`return.population` *return.population()*

Description

Converts one or more simulation populations to the marker-incidence data-frame representation used by the genetic-distance routines.

Usage

```
return.population(PopNames, missing.rm=TRUE)
```

Arguments

`PopNames`	Character string containing one or more simulation population names separated by blanks.
`missing.rm`	Logical flag controlling removal or retention of missing marker information in the returned representation.

Details

The requested simulation populations are evaluated at the current marker loci and returned in an NTSys-style matrix representation suitable for the corresponding SelectionTools data utilities. Multiple population names can be supplied in one space-separated string.

The returned object is a representation of population marker data; it does not remove or modify the source populations. The `missing.rm` argument controls handling of missing marker information in the conversion.

Value

A `data.frame` in the marker-incidence representation used by the genetic-distance routines. Returns `NULL` invisibly if the compiled population conversion cannot produce a result.

`rng.choose` *Choose the simulation random-number generator state*

Description

Choose the simulation random-number generator state.

Usage

```
rng.choose(rngName="")
```

Arguments

`rngName`	Function argument used by this operation.

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of selecting the requested random-number generator.

`rng.info` *Info the simulation random-number generator state*

Description

Info the simulation random-number generator state.

Usage

```
rng.info()
```

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of requesting random-number-generator information.

`rng.init` *Initialize the random-number generators*

Description

Initialize the internal random-number generators, optionally using a user-defined seed for reproducible runs.

Usage

```
rng.init(seed = NULL)
```

Arguments

`seed`	`NULL` for ordinary initialization, or one finite integer between `-.Machine$integer.max` and `.Machine$integer.max`.

Details

Package loading uses ordinary initialization, whose initial internal state is clock-based. Without a supplied seed, the existing initialization sequence is retained. Calling `rng.init()` after seeded use returns the internal generators to clock-based initialization and restores the preceding OpenMP dynamic-team setting; it does not reset R's random-number state.

With a seed, this function calls `set.seed(seed)`, draws 257 initialization values from R's generator, and resets every internal generator, including its buffered state. R's random-number state advances by these initialization draws. One value seeds the original MathLib12 stream derivation, and the other values seed the 256 internal simulation generators. R's currently selected RNG kind is respected. Existing generators and simulation algorithms are unchanged.

Seeded mode disables OpenMP dynamic-team adjustment and uses static scheduling for crossing, cross-validation and permutation tests. Reproducibility requires the same inputs, sequence of calls, R RNG kind, build and numerical libraries, and actual thread count. Set the thread count before initialization and do not change OpenMP settings during the run. Normal top-level execution with up to 256 threads is supported; shared fallback execution is outside this guarantee. Reset simulation data separately when repeating a complete experiment; `reset.all()` does not reset random generators. Other functions with explicit seed arguments may reset R's random-number state themselves.

Value

An invisible list returned by the underlying `.C` routine. The function is primarily called for its initialization side effects.

Examples

```
st.set.num.threads(1)
rng.init(seed = 1001)
# Run a simulation after preparing its input data.
rng.init()
```

rng.list *List the simulation random-number generator state*

Description

List the available simulation random-number generators and indicate the current choice. Output follows the package information-level setting and can therefore be suppressed through that setting.

Usage

```
rng.list()
```

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of requesting the available random-number generators.

sample.population *sample.population()*

Description

Creates a new population by sampling individuals from an existing simulation population.

Usage

```
sample.population(NameP1, NameP2, NoI=-1, rep=1)
```

Arguments

`NameP1`	Name of the sample population
`NameP2`	Name of the initial population
`NoI`	non-negative integer giving the number of individuals to choose
`rep`	Should sampling be with replacement?

Details

Sampling does not modify the source population. The requested sample is copied into the destination. Sampling can be with or without replacement according to `rep`. Raw chromosome data are always the authoritative copied representation; cached genotype and genetic-value categories are transferred only when complete in the source and successfully constructed for the entire destination.

Value

Returns invisible list of input arguments. The function is a compatibility wrapper and the meaningful result is the delegated population operation.

See Also

This function is a wrapper for `population.sample`.

Examples

```
map <- data.frame(chrom = 1,
                  pos = 0.01,
                  name = "form")
define.genome(map)

init.population("P1", homozygote(1))
init.population("P2", homozygote(2))

# Example: Generate 100 F1 hybrids
cross("F1","P1", "P2", 100)

# take a sample of 20 out of the 100 F1 plants
sample.population ("SampleFromF1","F1",252)
```

---

save.linkage.map *Save a linkage map*

Description

Save a linkage map.

Usage

```
save.linkage.map(file, dta.map=get.map())
```

Arguments

`file`	File name or file connection.
`dta.map`	Map-related parameter.

Value

`NULL`, returned by the delegated `linkage.map.save()` file-writing operation. The function is a compatibility wrapper.

sdev *Calculate a standard-deviation quantity used by the simulation code*

Description

Calculate a standard-deviation quantity used by the simulation code.

Usage

```
sdev(x, na.rm = FALSE)
```

Arguments

x	Input object.
na.rm	Function argument used by this operation.

Value

A numeric standard-deviation result. For a matrix or data frame, a numeric vector is returned with one standard deviation per column; for a vector or other object coerced to a vector, a numeric scalar is returned.

sel.target.define *Define a selection target*

Description

Define a selection target.

Usage

```
sel.target.define(criteria.df)
```

Arguments

criteria.df	Function argument used by this operation.

Value

A character vector containing one or two selection-target codes selected from "H1", "MIX", "H2", and "All". Returns NULL when the criterion cannot be translated into a supported target.

`select.all.best` *Select all individuals in the best value classes*

Description

Selects individuals belonging to the best distinct genetic or phenotypic value classes of a simulation population.

Usage

```
select.all.best(newPop, oldPop, effect=NULL, x=1,
                decreasing=TRUE, selection.criterion="G")
```

Arguments

`newPop`	Character string naming the selected population.
`oldPop`	Character string naming the source population.
`effect`	Optional character string naming the effect used for selection.
`x`	Number of distinct value classes to retain.
`decreasing`	Logical. If true, larger values are preferred.
`selection.criterion`	Character string. `"G"`, the default, selects on genetic values; `"P"` selects on phenotypic values.

Details

With `selection.criterion="G"`, the source population is genotyped and evaluated before sorting. Without a named effect, the weighted genetic index is used; with a named effect, selection is based on that effect.

Phenotypic selection uses previously stored phenotypic values and does not re-evaluate the population. Without a named effect, the weighted phenotypic index is used; with a named effect, its effect-specific phenotype is used. The argument `x` counts distinct values, so ties are retained together. Continuous phenotypic values will ordinarily produce fewer ties than many genetic scoring schemes.

Value

On successful selection, invisibly returns a one-row `data.frame` with columns `n`, `minscore`, and `maxscore`, giving the number selected and the score limits of the selected class(es). Returns `NULL` for an empty/unusable selection, or the invisible status `list` from `population.sort()` if sorting fails.

See Also

`phenotype.population`, `population.sort`, `select.n.best`

`select.all.best.intern` *Determine the best value classes in a sorted population*

Description

Determines how many individuals belong to the requested number of best value classes in an already sorted simulation population.

Usage

```
select.all.best.intern(NamePop, n=1, selection.criterion="G", effect=NULL)
```

Arguments

`NamePop`	Character string naming the already sorted population.
`n`	Number of distinct value classes to retain.
`selection.criterion`	Character string. `"G"`, the default, uses genetic values; `"P"` uses phenotypic values.
`effect`	Optional character string naming the effect whose genetic or phenotypic value is used. If omitted, element 0 of the selected value array is used.

Details

The function does not sort the population. It is used internally after `population.sort`. The first `n` distinct values in the current population order are treated as the best classes, and ties within a class are retained together.

With `selection.criterion="G"`, genetic values are inspected. With `selection.criterion="P"`, previously generated phenotypic values are inspected. If `effect` is omitted, element 0 of the corresponding value array is used; otherwise the named effect-specific value is used.

Value

On success, a one-row `data.frame` with columns `n`, `minscore`, and `maxscore`, giving the number of individuals in the selected class(es) and the corresponding score limits. Returns `NULL` invisibly if the compiled selection step fails.

See Also

`select.all.best`, `population.sort`

`select.genotypes` *Select specific allele combinations from populations*

Description

This function is used to select individuals from a population that fulfill certain selection criteria.

Usage

```
select.genotypes(newPop, oldPop, criteria)
```

Arguments

`newPop`	a character string specifying the name of the new population which is created from the selected genotype(s).
`oldPop`	a character string specifying the name of the population that selection is applied to.
`criteria`	a data.frame containing the locinames and the selection criteria for selection. Every row is one criterion. Data.frame must contain three columns: locusname, allele 1, allele 2.

Details

Individuals are selected based on their genotypes at the loci listed in ‘criteria‘. Genotype matching is performed using internally created effects.

If multiple rows are provided for the same ‘locus‘, an individual matches that locus if it satisfies **any** of the rows for that locus (OR within locus). Individuals must satisfy the requirements for **all** loci present in ‘criteria‘ (AND across loci).

Selected individuals are removed from ‘oldPop‘ and added to ‘newPop‘.

Value

On successful completion, returns the invisible implementation-level `list` produced by the final temporary-population removal. If the source population is empty, `NULL` is returned invisibly. The meaningful result is the side effect of constructing `newPop` according to the requested allele criteria.

See Also

[select.all.best()], [select.n.best()], [define.effects()]

Examples

```
reset.all()

map <- data.frame(1,0.0,"form","trait1")
define.genome(map)
```

```
init.population("P1",homozygote(1,100))
init.population("P2",homozygote(2,100))
append.population("ADM","P1")
append.population("ADM","P2")
cross("SYN1","ADM","ADM",1000)
evaluate.genotype("SYN1","form")
evaluate.allele.freq("SYN1","form")

# Select genotypes (1|1) and (1|2)
sel.crit <- data.frame(
  locus    = c ("form" , "form"),
  allele1  = c (1      ,  1    ),
  allele2  = c( 1      ,  2    )
)
select.genotypes("selected","SYN1",
                 sel.crit)
evaluate.genotype("selected","form")
```

`select.n.best` *Select a fixed number of best individuals*

Description

Selects a fixed number of individuals from a simulation population according to genetic or phenotypic values.

Usage

```
select.n.best(newPop, oldPop, effect=NULL, n=1,
              decreasing=TRUE, selection.criterion="G")
```

Arguments

`newPop`	Character string naming the selected population.
`oldPop`	Character string naming the source population.
`effect`	Optional character string naming the effect used for selection.
`n`	Number of individuals requested.
`decreasing`	Logical. If true, larger values are preferred.
`selection.criterion`	Character string. `"G"`, the default, selects on genetic values; `"P"` selects on phenotypic values.

Details

With `selection.criterion="G"`, the source population is genotyped and evaluated before selection. With `effect=NULL`, selection is based on the weighted genetic index. With a named `effect`, selection is based on that effect.

With `selection.criterion="P"`, stored phenotypic values are used and the population is not re-evaluated, because genetic re-evaluation invalidates phenotypic values. If `effect=NULL`, the weighted phenotypic index is used; with a named effect, its effect-specific phenotype is used.

Value

On successful selection, returns the invisible implementation-level `list` produced by `population.divide()`; if sorting fails the sort-status list is returned invisibly, and an empty population returns `NULL` invisibly. The main effect is creation of `newPop` containing the selected individuals.

See Also

`phenotype.population`, `population.sort`, `select.all.best`

`select.n.best.segments` *Select the best genome segments according to the requested criterion*

Description

Select the best genome segments according to the requested criterion.

Usage

```
select.n.best.segments(newPop, oldPop, n=1, allele = 1, decreasing = FALSE)
```

Arguments

`newPop`	Function argument used by this operation.
`oldPop`	Function argument used by this operation.
`n`	Number of items or repetitions.
`allele`	Allele specification.
`decreasing`	Function argument used by this operation.

Value

On successful selection, returns the invisible implementation-level `list` produced by `population.divide()`; if the requested number cannot be selected, `NULL` is returned invisibly. The main effect is creation of `newPop` from individuals ranked by the number of genome segments.

set.co.freq *Set crossover-frequency information*

Description

Set crossover-frequency information.

Usage

```
set.co.freq(cofreq=1)
```

Arguments

cofreq	Function argument used by this operation.

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of setting crossover-frequency information.

set.eff.weight *set.eff.weight()*

Description

Assigns a weight to a defined effect

Usage

```
set.eff.weight(fname, weight)
```

Arguments

fname	Name of the effect
weight	Weight of the effect

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of setting the requested effect weight.

Note

Effect weights are used when `evaluate.genotype` is carried out *without* specifing one effect. In this case all loaded effects are used to determin tne genotypic value, and weights are applied. If evaluate genotype is used *with* specifiying an effect, the weights are *not* used.
See `define.effects` for for an example

`set.genome.par` *set.genome.par()*

Description

Defines simulation chromosome number, ploidy, and chromosome lengths.

Usage

```
set.genome.par(no.chrom, no.hom, chrom.len)
```

Arguments

`no.chrom`	Positive number of simulation chromosomes.
`no.hom`	Number of homologues. Diploid population operations use two.
`chrom.len`	Numeric vector of chromosome lengths with one value per chromosome.

Details

The genome definition controls the chromosome structure used by simulation-population operations. `no.chrom` gives the number of chromosomes, `no.hom` the number of homologues, and `chrom.len` the chromosome lengths in the simulation map unit.

Population-generation operations including crossing, single-seed descent, doubled haploids, and the SelectionTools marker-to-simulation bridge require a diploid genome with two homologues. The chromosome-length vector must correspond to the declared chromosome count.

This function and `genome.parameter.set` modify the same simulation genome state.

Value

Returns the same implementation-level `list` as `genome.parameter.set()`; the main effect is setting the simulation genome parameters.

`set.info.level` *set.info.level()*

Description

The function *set.info.level()* sets the information level. The smaller the level number is the less program information is printed on the screen.

level	information printed
-2	error messages
-1	warning messages
0	default messages
1	verbouse mode

Usage

```
set.info.level(level)
```

Arguments

`level`	information level

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of setting the simulation information level.

Note

If you want to read only error messages then you have to set your information level to -2!

`set.mdp` *Set marker-assisted donor-parent information*

Description

Set marker-assisted donor-parent information.

Usage

```
set.mdp(MDP=0)
```

Arguments

`MDP`	Function argument used by this operation.

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of setting marker-assisted donor-parent information.

---

set.NoLociInit *Set the initial number of loci used by the simulation*

Description

Set the initial number of loci used by the simulation.

Usage

```
set.NoLociInit(NoLociInit)
```

Arguments

`NoLociInit`	Initial number of loci.

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of setting the initial number of loci.

---

set.population.gvalue *Set genetic selection values for a simulation population*

Description

Sets element 0 of the stored genetic value array for every active individual of a simulation population.

Usage

```
set.population.gvalue(name, gvalue)
```

Arguments

`name`	Character string naming the population.
`gvalue`	Finite numeric vector containing exactly one value for every active individual.

Details

The length of gvalue must exactly equal the active population size. The operation is population-wide: values are set for all active individuals or the operation is unsuccessful. Partial assignment and missing-value padding are not used.

When effects are loaded, the R interface first genotypes and evaluates the population so that the complete genetic value arrays exist for all effects. The supplied values then replace element 0 only; the effect-specific genetic values remain available. When no effects are loaded, the complete genetic value array consists only of element 0.

A successful assignment invalidates all stored phenotypic values for the population.

Value

On successful assignment, an invisible implementation-level list returned by the compiled setter, including its retval status component. Invalid population, length, or non-finite input returns NULL invisibly.

See Also

get.population.gvalue, evaluate.population, phenotype.population

---

set.population.info *Set population information used by the simulation routines*

Description

Set population information used by the simulation routines.

Usage

```
set.population.info(PopName, ind, info)
```

Arguments

PopName	Name of the population.
ind	Individual index or indices.
info	Information value.

Value

Returns the same invisible implementation-level list or NULL as population.info.set(), depending on whether the individual index is valid.

show.phase *Display a Compact Statistical Phasing Summary*

Description

Displays the main diagnostics from `st.phase()` together with switch-error statistics from `st.switch.error()`.

Usage

```
show.phase(label, result, score)
```

Arguments

`label`	Character string printed above the summary.
`result`	An object returned by `st.phase()`.
`score`	The one-column data frame returned by `st.switch.error()`.

Value

Returns `result` invisibly. The function is called for the side effect of printing a compact phasing summary.

See Also

`st.phase`, `st.switch.error`, `st.destroy.phase`

single.cross *Create or evaluate a single cross*

Description

Create or evaluate a single cross.

Usage

```
single.cross(PgName, PopName, classSize=1)
```

Arguments

`PgName`	Function argument used by this operation.
`PopName`	Name of the population.
`classSize`	Function argument used by this operation.

Value

An invisible implementation-level `list` returned by the final population-removal operation. The meaningful result is the side effect of creating the requested single-cross population.

`sm.start.timer` *Start or stop the simulation timing helper*

Description

Start or stop the simulation timing helper.

Usage

```
sm.start.timer(depth=1)
```

Arguments

`depth`	Function argument used by this operation.

Value

An invisible implementation-level `list` returned by the SelectionTools timing helper. The function is called for the side effect of starting timing.

`sm.stop.timer` *Start or stop the simulation timing helper*

Description

Start or stop the simulation timing helper.

Usage

```
sm.stop.timer(depth=1, info.level=0)
```

Arguments

`depth`	Function argument used by this operation.
`info.level`	Function argument used by this operation.

Value

An invisible implementation-level `list` returned by the SelectionTools information helper after reporting elapsed time. The function is called for the timing/reporting side effect.

splitdt *Split a data object according to the package-specific convention*

Description

Split a data object according to the package-specific convention.

Usage

```
splitdt(m.a)
```

Arguments

m.a	Marker or matrix input.

Value

A character matrix with two columns and one row for each input string. The first column contains the part before the first dot (or underscore if no dot is present), and the second column contains the remaining part; if no separator is present the second field is empty.

ssd.mating *ssd.mating()*

Description

Generates single-seed-descent progeny by repeated selfing of each individual in a diploid parental population.

Usage

```
ssd.mating(NamePg, NameP, NoPg=1, maxcycles=1)
```

Arguments

NamePg	Character string naming the progeny population to create.
NameP	Character string naming the parental population.
NoPg	Positive number of independently generated descendants per parental individual.
maxcycles	Positive maximum number of successive selfing generations for each descendant.

Details

For every active individual in NameP, NoPg descendant lines are generated. The total progeny size is therefore the parental population size multiplied by NoPg. Descendants belonging to one parent occupy consecutive positions in the new population.

Each descendant is repeatedly selfed for at most maxcycles generations. Selfing stops earlier when the simulated chromosomes have become completely homozygous. Large values of maxcycles can therefore be used to approach fully homozygous recombinant inbred lines without requiring unnecessary generations after homozygosity is reached.

The routine requires a diploid simulation genome. The destination is rebuilt from chromosome data and does not retain stale cached genotypes or evaluated values. Allocation or meiosis failure removes the incomplete destination population.

Value

An invisible list returned by the underlying .C routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of creating progeny using the SSD mating routine.

---

st.calc.ld *Calculate Linkage Disequilibrium from Marker Data*

Description

Calculates pairwise linkage disequilibrium within chromosomes. LD can either be calculated directly from the two stored homologues or, for biallelic markers, after estimation of unknown two-locus gametic phase.

Usage

```
st.calc.ld(ld.measure = "r2", auxfiles = TRUE, data.set = "default")
```

Arguments

ld.measure	Character string selecting the LD measure. Supported values are "r2", "Dp", "r2-estimate-phases", and "Dp-estimate-phases".
auxfiles	Logical. If TRUE, the pairwise results are written to an R-session temporary file, read back by the R wrapper, and returned as a data frame. If FALSE, LD is calculated and retained internally in the data set but the pairwise table is not transferred to R.
data.set	Character string naming the SelectionTools marker data set.

Details

LD is calculated for every marker pair on the same chromosome.

With `ld.measure="r2"` and `ld.measure="Dp"`, the marker representation is treated as phased. The two allele arrays represent the two stored homologues, and alleles at two loci from the same homologue are counted together as haplotypes. Thus, for heterozygous material the stored allele order must represent the homologues correctly. For completely homozygous lines the phase is immaterial. These two direct calculations support multiallelic loci.

For allele x at the first locus and allele y at the second locus, let f_x and f_y be the allele frequencies and f_{xy} the haplotype frequency. The gametic disequilibrium coefficient is

$$D_{xy} = f_{xy} - f_x f_y.$$

For `ld.measure="r2"`, the allele-pair value is

$$r^2_{xy} = \frac{D^2_{xy}}{f_x(1-f_x)f_y(1-f_y)}$$

and the multiallelic value returned by SelectionTools is

$$r^2 = \sum_x \sum_y f_x f_y r^2_{xy}.$$

For `ld.measure="Dp"`, the allele-pair value is the absolute normalized disequilibrium

$$D'_{xy} = \left| \frac{D_{xy}}{D_{max}} \right|,$$

where

$$D_{max} = \min(f_x f_y, (1-f_x)(1-f_y)) \quad \text{if } D_{xy} < 0$$

and

$$D_{max} = \min(f_x(1-f_y), (1-f_x)f_y) \quad \text{if } D_{xy} \geq 0.$$

The multiallelic value is

$$D' = \sum_x \sum_y f_x f_y D'_{xy}.$$

The options `"r2-estimate-phases"` and `"Dp-estimate-phases"` are intended for heterozygous material with unknown gametic phase. They require exactly two biological alleles at every marker in the selected data set. The missing-allele state is not a biological allele. Individuals with missing data at either locus of a marker pair are omitted from that pair's two-locus genotype table.

For the phase-estimating modes, the nine observable genotype classes at two biallelic loci are counted without using the order of the two stored homologues as phase information. All genotype classes except the double heterozygote have an unambiguous pair of haplotypes. Let h_{00}, h_{01}, h_{10}, and h_{11} denote the current haplotype-frequency estimates. In the expectation step, the probability that a double heterozygote has the coupling diplotype is

$$q = \frac{h_{00}h_{11}}{h_{00}h_{11} + h_{01}h_{10}}.$$

The ambiguous individuals are split between the coupling and repulsion diplotypes according to q; the four haplotype frequencies are then re-estimated from the resulting expected haplotype counts.

These expectation and maximization steps are repeated to convergence. SelectionTools uses several deterministic starting values and retains the converged solution with the greatest observed-data likelihood. The corresponding r^2 or D' is calculated from the estimated haplotype frequencies. This is a native maximum-likelihood implementation for the two-locus phase-ambiguity problem; Clayton and Leung (2007) discuss haplotype estimation for association data.

For the direct `"r2"` and `"Dp"` calculations, the missing allele is excluded from the allele-pair sums, while the frequency denominator is twice the number of individuals in the data set. Marker filtering for missingness and polymorphism can therefore be useful before LD calculation.

When `auxfiles=FALSE`, the internally stored LD values can be used by `st.def.hblocks` without constructing a potentially very large R data frame. This can reduce transfer and file-reading time for large data sets.

For a general review of LD measures and their dependence on allele frequencies, see Hedrick (1987), *Genetics* 117:331–341.

Value

If `auxfiles=TRUE` and calculation succeeds, invisibly returns a `data.frame` with columns `Chrom`, `Locus1`, `Locus2`, `Name1`, `Name2`, and `LD`. `Locus1` and `Locus2` are one-based marker positions within the chromosome. If `auxfiles=FALSE`, invisibly returns `NULL`; the calculated LD values remain stored in the selected data set for subsequent SelectionTools operations. If a phase-estimating measure is requested and the data set is not biallelic, the calculation is rejected.

References

Clayton D, Leung HT (2007) An R package for analysis of whole-genome association studies. Human Heredity 64:45–51.

Hedrick PW (1987) Gametic disequilibrium measures: proceed with caution. Genetics 117:331–341.

See Also

`st.LDplot.ld`, `st.LDplot.map`, `st.LDheatmap`, `st.def.hblocks`

---

st.calc.ld.2 *Calculate linkage disequilibrium from marker data*

Description

Calculate linkage disequilibrium from marker data.

Usage

```
st.calc.ld.2(ld.measure = "r2", auxfiles = TRUE, keep.ldfile = FALSE,
             data.set = "default", ld.filename = NULL)
```

Arguments

ld.measure	Function argument used by this operation.
auxfiles	Logical. Controls generation of the LD result file. With keep.ldfile = FALSE, this file is created in the R session temporary directory, read back, and removed before the function returns.
keep.ldfile	Logical. If FALSE, the LD result is transferred through a session-temporary file and returned to R. If TRUE and auxfiles = TRUE, a persistent LD file is written and retained; ld.filename must then be supplied. The function returns NULL invisibly when keep.ldfile = TRUE.
data.set	Name of the SelectionTools data set.
ld.filename	Character string naming the persistent LD output file. Required only when keep.ldfile = TRUE and auxfiles = TRUE.

Value

Invisibly returns a data.frame with columns Chrom, Locus1, Locus2, Name1, Name2, and LD when the generated LD file is read back. If keep.ldfile = TRUE, the R return value is NULL; when auxfiles = TRUE, the LD file named by ld.filename is retained.

---

st.calc.q *Calculate the package-specific Q statistic or quantity*

Description

Calculate the package-specific Q statistic or quantity.

Usage

```
st.calc.q(pop.type = "DH", t = 0, auxfiles = TRUE, data.set ="default")
```

Arguments

pop.type	Population type.
t	Generation or method-specific time parameter.
auxfiles	Logical. If TRUE, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
data.set	Name of the SelectionTools data set.

Value

Invisibly returns a data.frame with columns Chrom, Locus1, Locus2, and q, containing the calculated pairwise q values; otherwise NULL.

st.calc.rf *Calculate recombination fractions*

Description

Calculate recombination fractions.

Usage

```
st.calc.rf(map.function = "Haldane", auxfiles = TRUE, data.set = "default")
```

Arguments

map.function	Map function to use.
auxfiles	Logical. If TRUE, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
data.set	Name of the SelectionTools data set.

Value

Invisibly returns a data.frame with columns Chrom, Locus1, Locus2, and RecFreq, containing pairwise recombination fractions; otherwise NULL.

st.calculate.enrichment *Calculate Haplotype Enrichment and Select Donor Genotypes*

Description

Counts additional block-specific haplotype alleles supplied by one phased marker data set to another, or selects donor genotypes by exact best-subset search or stepwise forward selection. Both stored homologues are considered.

Usage

```
st.calculate.enrichment(base.pool, enrichment.pool,
                        haplotype.options = NULL, blocks = NULL,
                        exclude.compatible = TRUE)

st.select.enrichment(base.pool, enrichment.pool, n,
                     haplotype.options = NULL, blocks = NULL,
                     exclude.compatible = TRUE, method = "best.subset")
```

Arguments

base.pool	Name of the already phased base marker data set.
enrichment.pool	Name of the already phased donor marker data set. For calculation, all its genotypes contribute. For selection, it is the candidate pool.
n	Positive integer number of donor genotypes to select.
haplotype.options	Named list of options passed to st.def.hblocks: hap, hap.unit, ld.threshold, ld.criterion, tolerance, and hap.symbol. An additional ld.measure option selects the measure for st.calc.ld when LD-based blocks are requested (default "r2"). Omitted entries use the defaults of those routines. An empty list requests the default five-marker blocks. Supply exactly one of haplotype.options and blocks.
blocks	Block-definition data frame, typically returned by st.def.hblocks. The Name, Chrom and Markers columns define unique block names, chromosomes, and semicolon-separated marker membership. Blocks must be nonoverlapping and each must contain consecutive markers on one chromosome. A subset of blocks is allowed. Additional columns in the supplied table are retained.
method	Selection method: "best.subset" (default) finds an exact optimal subset; "forward" adds the genotype with the largest remaining gain at each step. Forward selection is faster for large pools but need not find the globally best subset.
exclude.compatible	Logical. If TRUE, exclude additional variants compatible with an incomplete base haplotype from the selection objective and reported Score. Both categories are always reported by st.calculate.enrichment.

Details

The two data sets must already have consistent gametic phase along each homologue. Phasing is not performed by these functions. The data sets must contain the same marker identities, chromosome assignments and map positions. Genotypes are aligned by marker name. Their internal integer allele coding must have the same biological meaning at each marker in both data sets. SelectionTools does not retain the original textual REF/ALT labels in marker data, so matching biological coding cannot be verified automatically. Creating both pools by copying and restricting a common marker data set preserves the coding. Individual names are local to each pool and may overlap between pools.

When options are supplied, block definitions are computed exclusively from a private working copy of the base population, including its LD if needed. The same base data and options therefore produce the same blocks when the full donor pool is replaced by selected donors. Supplying fixed blocks instead uses their marker membership directly. The input marker data sets, including their block definitions, are not modified. The temporary native working data set is released on return or error; no new simulation or phasing code is involved.

Haplotype identity is the ordered sequence of marker allele states within a block, not an independently assigned haplotype code. Only complete sequences are counted as observed alleles. A variant absent from complete base calls is potentially hidden if it agrees with every observed marker of at least one incomplete base haplotype. An entirely missing base haplotype is compatible with

every variant at that block. This is conditional on the supplied phase and calls; lack of a compatible incomplete sequence does not prove biological novelty. Native allele arrays are read directly. Missing values on one homologue do not discard observed values on the other homologue.

Selection maximizes the union of qualifying additional alleles, not the sum of individual donor scores. Shared variants count once per block. With `method="best.subset"`, an exact branch-and-bound search prunes only branches that cannot improve the current best score. Equal optima are resolved by the first combination of genotype names sorted in bytewise lexicographic order. The result is deterministic and does not consume random numbers. Runtime can grow combinatorially for large pools; no heuristic result is silently substituted for an exact optimum.

With `method="forward"`, each step chooses the remaining genotype contributing the most qualifying alleles not already supplied by selected genotypes. Equal gains are resolved by bytewise lexicographic genotype name. Selection continues to exactly `n` genotypes even when remaining gains are zero. Block borders and compatibility with incomplete base haplotypes remain fixed relative to the original base population throughout the search. This makes both methods optimize the same enrichment objective.

Counting and both search methods are implemented in C. OpenMP builds use the existing `st.set.num.threads` setting for parallel search; selections, including ties, are independent of thread count. Serial builds use the same algorithms. No random numbers are consumed.

Value

`st.select.enrichment` returns a character vector of exactly `n` donor genotype names: sorted by name for best-subset search, and in selection order for forward search. It does not create a selected marker data set.

`st.calculate.enrichment` returns a list containing:

`summary` One-row data frame with `Base.lines`, `Enrichment.lines`, `Main`, `Additional`, `Hidden`, `No.match`, `Score`, `Enriched`, and `Gaining.blocks`. `Additional` = `Hidden` + `No.match`; `Enriched` = `Main` + `Additional`. Counts are sums of distinct block-specific alleles, not allele copies.

`by.block` The same allele counts and score for each block, including blocks with zero gain.

`alleles` Additional observed variants, with columns `Block`, `Sequence` and logical `Hidden`.

`blocks` The supplied or generated block-definition table.

`exclude.compatible` The scoring option used.

See Also

`st.phase`, `st.def.hblocks`, `st.calc.ld`, `st.copy.marker.data`, `st.restrict.marker.data`

`st.chrom.stats` *Return chromosome statistics for a data set*

Description

Return chromosome statistics for a data set.

Usage

```
st.chrom.stats(filename = "chrom.stats", auxfiles = TRUE, data.set ="default")
```

Arguments

filename	Character string retained for compatibility and used as part of the name of the internal session-temporary result file.
auxfiles	Logical. If TRUE, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
data.set	Name of the SelectionTools data set.

Value

Invisibly returns a data.frame with columns Chrom, NLoci, and Length, giving chromosome identifier, number of loci, and chromosome length; otherwise NULL if the calculation fails.

---

st.compact.data.set *Compact a SelectionTools Marker Data Set*

Description

Releases calculated results and reusable workspace allocations associated with one SelectionTools marker data set while retaining its marker data and linkage map.

Usage

```
st.compact.data.set(data.set = "default")
```

Arguments

data.set	Character string naming the marker data set to compact.

Details

The retained core consists only of individual and marker names, the two allele arrays containing the marker genotypes, the data-set phase state, and the linkage map including its marker-to-data indices.

All other dynamically allocated data-set state is discarded. This includes marker statistics, phenotypic and genotypic values, effects and model results, design and relationship matrices, haplotype-block and cross structures, chromosome-specific recombination/LD workspaces, and reusable temporary buffers. Consequently columns such as effects, phenotypic.value, and genotypic.value in st.data.set.list() can change from TRUE to FALSE after compaction.

Values that are required again must be recalculated or reloaded by the corresponding SelectionTools function.

Value

Invisibly returns the estimated amount of work memory released, in MiB. The estimate excludes memory allocator overhead.

`st.copy.marker.data` *st.copy.marker.data()*

Description

Creates an independent copy of the primary marker data, marker statistics, linkage-map representation, and phenotype vector of another SelectionTools data set.

Usage

```
st.copy.marker.data(target.data.set, source.data.set)
```

Arguments

`target.data.set`
: Character string naming the data set that will receive the copy. Existing contents of this target are replaced.

`source.data.set`
: Character string naming the marker data set to copy.

Details

The target receives its own marker-name and individual-name arrays, allele matrices, recomputed marker statistics, map storage, map access pointers, marker-to-map indices, and phenotype vector when present. Map storage order and map access order are reconstructed so that the copy does not depend on pointers into the source data set.

The copy is intended as a copy of the marker-data state, not of every fitted model or temporary calculation attached to the source. Model matrices, fitted effects, p-values, cross-validation state, and other derived analysis objects are not implied by this operation.

The source data set is not modified. Existing contents of the target data set are replaced.

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of copying marker data from the source data set to the target data set.

st.data.set.list *List SelectionTools Marker Data Sets*

Description

Returns a compact summary of the marker data sets currently registered in SelectionTools.

Usage

```
st.data.set.list(digits=getOption("digits"))

## S3 method for class 'st.data.set.list'
print(x, digits=attr(x,"digits"), complete=FALSE, ...)
```

Arguments

digits	Integer between 1 and 22 giving the number of significant digits used when the returned summary is printed. The stored numeric values are not rounded.
x	An object returned by st.data.set.list().
complete	Logical. If FALSE (the default), print only the compact overview columns data.set, no.markers, no.individuals, map, ram.core, and ram.work. If TRUE, print all columns.
...	Additional arguments passed to print.data.frame().

Details

Only non-empty data sets are returned. A data set is omitted when both its number of markers and number of individuals are zero. Thus the built-in "default" data set is not shown while it is empty. Non-empty user-defined data sets are returned in creation order.

heterozygosity is the fraction of complete diploid genotype calls that are heterozygous. missing.markers is the fraction of marker-by-individual genotype calls with at least one missing allele. Both are between zero and one and are NA when they cannot be calculated.

phased is TRUE when the stored All1/All2 order is known to be phased. VCF input is marked phased only when genotype separators are consistently |; STvcf import/export retains the same data-set phase state. Simulation populations returned by st.get.simpop() or st.get.simpop2() are phased because the simulation backend returns the two chromosome homologues separately. A complete successful st.phase() commit is marked phased when no heterozygous phase remains unresolved, while st.destroy.phase() clears the state. Operations that only subset individuals or markers, replace a linkage map, add phenotypic values, copy marker data, or replace markers by derived haplotype blocks retain the state.

map indicates that a linkage map is present and no.chromosomes gives the number of chromosomes represented by that map. effects, phenotypic.value, and genotypic.value indicate the presence of estimated effects, phenotypic values, and genotypic values, respectively.

ram.core and ram.work are estimates in MiB (1024^2 bytes). ram.core is deliberately restricted to the marker data required to retain the data set: individual and marker names, the two allele arrays, and the linkage-map storage. ram.work contains all other dynamically owned state, including

marker statistics, phenotypic and genotypic values, effects and model results, haplotype/cross objects, chromosome-specific recombination and LD matrices, and reusable calculation buffers. Allocator overhead is not included. `st.compact.data.set()` releases the work portion while retaining the core.

The returned object always contains all columns. Its print method shows a compact overview by default: `data.set`, `no.markers`, `no.individuals`, `map`, `ram.core`, and `ram.work`. Use `print(x,complete=TRUE)` to display the complete summary.

The `digits` argument affects printing only. Thus, for example, `x <- st.data.set.list(digits=3)` stores full-precision values but prints the numeric columns with three significant digits. Printing can also be overridden later with `print(x,digits=5)`.

Value

An object inheriting from `data.frame`, with one row per non-empty registered data set and the following columns. If all registered data sets are empty, a zero-row data frame with these columns is returned:

- `data.set`: data-set name.
- `no.markers`: number of markers.
- `no.individuals`: number of individuals.
- `heterozygosity`: heterozygosity among complete genotype calls.
- `missing.markers`: fraction of missing marker-genotype calls.
- `phased`: whether allele order is known to be phased.
- `map`: whether a linkage map is present.
- `no.chromosomes`: number of chromosomes represented in the map.
- `effects`: whether estimated effects are present.
- `phenotypic.value`: whether phenotypic values are present.
- `genotypic.value`: whether genotypic values are present.
- `ram.core`: estimated marker-data and linkage-map RAM in MiB.
- `ram.work`: estimated calculated/result/workspace RAM in MiB.

The `digits` setting is stored as an attribute and changes printing only.

`st.data.set.remove` *Remove a SelectionTools Marker Data Set*

Description

Releases all native storage belonging to one named SelectionTools marker data set.

Usage

```
st.data.set.remove(data.set)
```

Arguments

`data.set`	One non-empty character string naming the data set to remove.

Details

For a user-defined data set, all marker, map, phenotype, model, haplotype, and other derived native storage is released and the data-set name is removed from the registry. For `"default"`, the contents are reset but the built-in name remains registered.

The function does not create a missing data set. If `data.set` is not registered, no state is changed.

Value

Invisibly returns `TRUE` when the named data set was found and cleared or removed, and `FALSE` when it was not registered.

`st.data.set.remove.all` *Remove All SelectionTools Marker Data Sets*

Description

Releases all user-defined SelectionTools marker data sets and resets the built-in default data set.

Usage

```
st.data.set.remove.all()
```

Details

All native storage belonging to registered user-defined marker data sets is released and their registry entries are removed. The contents of `"default"` are reset; the built-in default name itself remains registered.

Value

Invisibly returns the number of user-defined data sets removed. The built-in `"default"` data set remains registered internally but is empty. Because `st.data.set.list()` omits empty data sets, it returns a zero-row data frame if no other non-empty data sets remain.

st.datadir *Construct a path in the SelectionTools data directory*

Description

Construct a path in the SelectionTools data directory.

Usage

```
st.datadir(filename)
```

Arguments

filename	File name.

Value

A character scalar containing the constructed path in the SelectionTools data directory.

st.dataframe.to.STvcf *Convert a VCF-Like Data Frame to Compact STvcf Format*

Description

Converts a wide diploid VCF-like R data frame to the compact STvcf list representation used by `st.load.vcf.data()`.

Usage

```
st.dataframe.to.STvcf(data)
```

Arguments

data	A data frame containing VCF-style fixed fields and sample genotypes. The required layout and conversion conventions are described below.

Details

The converter is deliberately strict. It is intended to create only STvcf objects that can subsequently be accepted by the SelectionTools marker-data and simulation-transfer path. Invalid marker or individual names, malformed genotypes, inconsistent REF/ALT definitions, invalid map positions, and unsupported allele indices are rejected during conversion rather than being left for a later consumer to discover.

The first five data-frame columns are fixed by position and must be, in this order, `CHROM` or `#CHROM`, `POS`, `ID`, `REF`, and `ALT`. This positional convention is intentional. Sample names are allowed to be

ordinary alphanumeric strings such as POS, INFO, or FORMAT; therefore fixed fields cannot safely be identified only by searching all column names.

Data-frame column names must be non-missing and non-empty. Duplicate names are permitted only when they arise because a sample has the same name as a fixed or metadata field. Sample names themselves must be unique.

Marker IDs are mandatory and unique. Marker and individual identifiers may contain only ASCII letters A-Z and a-z and digits 0-9. Dots, underscores, hyphens, whitespace, punctuation, and non-ASCII characters are rejected. Names must contain fewer than 256 bytes. The same restrictions are checked again by st.load.vcf.data() and by the reverse marker-data conversion. Names are never silently modified.

Input chromosome labels may be numeric or character. They are converted to consecutive running integer codes 1, 2, ..., in order of first appearance. The original input labels are retained in CHROM.levels. One information message at level 0 reports the mapping. Chromosome labels are not marker or individual names and are not subject to the alphanumeric name restriction.

STvcf positions are genetic positions in centimorgan. If a valid metadata column named CM is supplied, it is used for the genetic position and the input POS values are not used for mapping. Otherwise POS is assumed to contain cM. This is important for conventional VCF data, where POS normally denotes physical base-pair position.

When a valid FORMAT column is present, columns between ALT and FORMAT may be the VCF metadata fields QUAL, FILTER, INFO, and the SelectionTools-specific CM field. These metadata fields must be unique. Every column after FORMAT is a sample column, irrespective of its name. This allows sample identifiers such as POS, INFO, CM, or FORMAT without ambiguity.

A FORMAT column is recognized only when every marker-specific definition contains exactly one non-empty GT field. Empty FORMAT fields, empty FORMAT subfields, trailing colons, and duplicated GT entries are invalid. The FORMAT definition may vary among markers. If more than one column can be interpreted as FORMAT, conversion is rejected. FORMAT must be followed by at least one sample column. The GT subfield is extracted from every sample cell. Other subfields, including DP, AD, GQ, and phase-set fields, are ignored and are not retained. A non-missing sample field that is too short to contain the GT subfield defined by FORMAT, or that contains an empty GT subfield, is rejected as malformed rather than being converted to missing. If FORMAT is absent, columns after the first five fixed fields are direct GT sample columns, except that an alphanumeric column named CM whose complete contents are valid finite nonnegative numeric values is interpreted as the genetic CM field. If more than one column can be interpreted as CM, conversion is rejected. This rule is the convention used to distinguish an optional CM field from a sample that itself is named CM.

Genetic positions are stored as R double-precision numeric values without intentional rounding. Factor CM or POS columns are converted through their printed character values before numeric conversion, so factor level numbers are never used as map positions. Positions must be finite and nonnegative.

REF must contain exactly one non-missing allele label. It may not be "." and may not contain a comma. ALT may be ".", denoting no defined alternative allele, or a comma-separated list. Every ALT entry must be non-empty, may not be ".", must differ from REF, and must be unique within the marker. Thus definitions such as REF=A with ALT=A or ALT=C,C are rejected.

Multiallelic markers are accepted. VCF allele index 0 denotes REF, index 1 the first ALT allele, index 2 the second ALT allele, and so on. Every observed non-missing allele index must be present in

the corresponding REF/ALT definition. At most 253 ALT alleles may be defined for a marker. Together with REF this corresponds to at most 254 biological allele states in a SelectionTools marker data set; missing data use a separate internal state.

Only diploid genotypes are supported. Both "/" and "|" are accepted. The supplied allele order is retained in allele1 and allele2. The returned STvcf object carries a logical phased attribute. It is TRUE only when genotype separators are consistently |; mixed | and / input is retained but marked unphased. Retaining allele order is not a statistical phasing procedure and must not be interpreted as phase inference. Phase-set boundaries and other phase metadata are not represented in STvcf.

Accepted complete-missing representations include R NA, an empty field, ".", "./.", ".|.", "-1", "-1/-1", "-1|-1", and the character value "NA". The -1 spelling is accepted as a SelectionTools backward-compatibility extension; standard VCF output continues to use "." for missing alleles. Partially missing diploid genotypes such as 0/., ./1, 0|., .|1, 0/-1, and -1/1 are normalized to complete missingness. The non-missing member of a partial pair must nevertheless be a syntactically valid nonnegative integer allele index within the REF/ALT definition. Malformed fields such as an empty allele component or a nonnumeric non-missing component are rejected rather than being hidden by the partial-missing rule.

The converter writes allele1 and allele2 as raw matrices. Valid non-missing VCF allele indices are 0 through 253, and raw value 255 denotes missing data. Integer allele matrices are also accepted by the STvcf loader and reverse converter for manually constructed STvcf objects, but the 253-ALT contract still applies.

The returned list has class "STvcf" and components CHROM, POS, ID, REF, ALT, individual, allele1, allele2, and CHROM.levels. Marker and individual order are retained. Marker ordering is changed only when the STvcf object is installed in the internal SelectionTools linkage-map structure.

The conversion is not a byte-for-byte VCF round trip. QUAL, FILTER, INFO, FORMAT subfields other than GT, phase-set information, and physical POS when CM is supplied are not stored. The data-set level distinction between consistently phased | genotypes and unphased/mixed genotypes is retained in the logical phased attribute.

Value

A list of class "STvcf" containing marker metadata, individual identifiers, two compact allele matrices, and the chromosome-label mapping. Positions are stored as double-precision values. The logical attribute phased records whether the input GT separators were consistently phased.

---

st.dd *Construct a path in the SelectionTools data directory*

Description

Construct a path in the SelectionTools data directory.

Usage

```
st.dd(filename)
```

Arguments

filename	File name.

Value

A character scalar containing the constructed path in the SelectionTools data directory.

---

st.def.hblocks *Define Haplotype Blocks*

Description

Defines haplotype-block boundaries by a fixed number of markers, a genetic window in cM, linkage disequilibrium, or disjunct chromosome segments.

Usage

```
st.def.hblocks(hap = 5, hap.unit = 1, ld.threshold = 0.8,
               ld.criterion = "none", tolerance = 0, hap.symbol = "b",
               out.filename = "blocks", auxfiles = TRUE, data.set = "default")
```

Arguments

hap	Block size for the fixed-block methods: number of markers when hap.unit=1, or genetic distance in cM when hap.unit=2.
hap.unit	Block-definition mode when ld.criterion="none": 1 builds blocks by marker count, 2 builds blocks over a specified genetic distance in cM, and 0 builds disjunct chromosome segments as used for introgression populations.
ld.threshold	Minimum LD value used when ld.criterion is "average" or "flanking". Previously calculated LD values must be available in data.set.
ld.criterion	Criterion used for LD-based block definition. "none" uses hap.unit; "average" evaluates the average pairwise LD within a candidate block, and "flanking" evaluates LD between the markers flanking a candidate block.
tolerance	Non-negative number of neighboring markers used as a tolerance region when extending an LD-defined block. Within that region the best value of the selected LD criterion is used for the extension decision.
hap.symbol	Character prefix used for generated block names. With the default "b", blocks are named b000000, b000001, and so on.
out.filename	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
auxfiles	Logical. If TRUE, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
data.set	Character string naming the marker data set in which haplotype blocks are defined.

Details

Haplotype analysis is performed in two steps. This function defines the block boundaries along each chromosome; `st.recode.hil` subsequently searches the marker sequences between those boundaries and recodes the observed haplotype variants as alleles of the block loci.

For `hap.unit=1`, consecutive observed markers are grouped into blocks of up to `hap` markers without crossing chromosome boundaries. For `hap.unit=2`, consecutive markers are grouped according to their genetic map positions in cM. For `hap.unit=0`, block boundaries are determined from disjunct marker segments in the current genotypes.

If `ld.criterion` is `"average"` or `"flanking"`, the LD-based method takes precedence over `hap.unit`. It requires LD values to have been calculated beforehand. The algorithm starts blocks from adjacent marker pairs exceeding `ld.threshold` and extends them while the selected LD criterion remains at least the threshold. Markers that cannot be joined to another marker under the criterion form single-marker blocks.

This function only defines the blocks; it does not yet replace the marker data by haplotype loci. That replacement occurs when `st.recode.hil` is called.

Value

When `auxfiles = TRUE` and the operation succeeds, invisibly returns a `data.frame` describing the defined haplotype blocks:

`Chrom` Chromosome number of the block.

`Pos` Block position, calculated by the current implementation as the midpoint between the positions of the first and last marker in the block.

`Name` Generated haplotype-block name.

`Class` Class assigned to the haplotype block by the compiled implementation.

`Markers` Names of the markers used to construct the block, stored as a semicolon-separated marker list in the intermediate representation.

Returns `NULL` when no auxiliary result is requested or when the operation does not produce a readable block description.

See Also

`st.calc.ld`, `st.recode.hil`

---

st.destroy.phase *Randomize the Stored Phase of a Marker Data Set*

Description

Destroys known phase in a SelectionTools marker data set without changing any unordered diploid genotype. The function is mainly intended for validation of statistical phasing on simulated data with known true phase.

Usage

```
st.destroy.phase(data.set = "default")
```

Arguments

data.set	Character string naming an existing SelectionTools marker data set.

Details

For every observed heterozygous genotype, the two stored alleles are exchanged with probability 0.5. Homozygous and missing genotypes are unchanged. Thus the allele pair at every marker remains biologically identical, while the correspondence of the two stored homologues across heterozygous markers is randomized.

The function uses R's random-number generator. Call `set.seed()` before `st.destroy.phase()` when a reproducible randomized phase is required. The marker data set is modified in place and its native phase state is set to `FALSE`, including the degenerate case in which no heterozygous genotype was present.

Value

Invisibly returns a list with components `heterozygotes`, the number of observed heterozygous genotypes examined, and `swapped`, the number whose two stored alleles were exchanged.

See Also

`st.switch.error`, `st.phase`, `write.vcf`

---

st.genetic.distances *st.genetic.distances()*

Description

The function *st.genetic.distances* calculates genetic distances for each possible pair of individuals.

Usage

```
st.genetic.distances(measure = "mrd", format = "l",
                     filename = NULL, auxfiles = FALSE,
                     data.set = "default")
```

Arguments

measure	character. Defines the measure to be used for calculation. Incorporated measures are modified Rogers distance ("mrd"), Rogers distance ("rd") and Euclidean distance ("euc").
format	character. Specifies the format of the output file. Set to "l" for long format or to "m" for matrix format.

filename	Character string giving the base name of the genetic-distance file. Required when auxfiles = TRUE; the file is written with extension '.gdi' in the output directory.
auxfiles	Logical. If TRUE, a persistent output file with the genetic distances is generated and filename must be supplied.
data.set	character. Name of the data set that is used by the function.

Details

The Euclidean distance is calculated as:

$$D_E = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n_j}(p_{ij} - q_{ij})^2}$$

where p_{ij} and q_{ij} are allele frequencies of the jth allele at the ith locus in the two individuals under consideration, n_i is the number of alleles at the ith locus, and m refers to the number of loci. The D_E ranges from zero to 2m, the limits being assumed when the two individuals have identical allele frequencies or are fixed for different alleles.
The D_E is appropriate if allelic informative marker data are available and the relationships between individuals are investigated in combination with multivariate methods that require dissimilarities possessing the Euclidean property (Reif 2005).

The Rogers distance (Rogers 1972) is a modification of D_E and is calculated as follows:

$$D_R = \frac{1}{m}\sum_{i=1}^{m}\sqrt{\frac{1}{2}\sum_{j=1}^{n_j}(p_{ij} - q_{ij})^2}$$

The D_R is the average D_E across all loci standardized with the factor 1/2 to restrict the values to the interval [0,1]. It is one only if two individuals are fixed for different alleles, but if one or both individuals are not fixed and they have no alleles in common, D_R is not equal to one.
Melchinger et al. (1991) derived theoretical results that D_R estimates between two homozygous inbreds are linearly related to the coancestry coefficient. Consequently, D_R is suitable for studying the relationship between the genetic dissimilarity of inbreds based on allelic informative marker data and the coefficient of coancestry.

The modified Rogers distance (Wright 1978) is calculated with the formula:

$$D_W = \frac{1}{\sqrt{2m}}\sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n_j}(p_{ij} - q_{ij})^2}$$

As an Euclidean distance with the values in [0,1] it can be used for the same applications as recommended for D_E. Like D_R, D_W is not equal to one in the case of multiple alleles, even if the two individuals have no allele in common.
D_W is especially suitable in studies based on allelic informative marker data for examining the prediction of heterosis with genetic dissimilarities or the establishment of heterotic groups. Furthermore, D_W can be used for the same applications as suggested for D_E (e.g., multivariate methods),

owing to its Euclidean property.

The Rogers distance and the modified Rogers distance are standardized measures in the interval [0,1]. The standardization is achieved by the divisions by m and by 2 in the respective equations. This requests codominant markers! These measures can't be used with dominant markers. However the Euclidean distance is not standardized and can also be used with dominant marker data.
For homozygous genotypes Rogers distance equals the Nei-Li Distance (Nei and Li 1979), which is the same as 1 minus the Dice coefficient (Dice 1945), further the modified Rogers distance is the square root of the Rogers distance.

Value

Invisibly returns either an object of class `"dist"` when `format = "m"`, representing pairwise genetic distances, or a long-format `data.frame` with columns `OTU1`, `OTU2`, and `Measure` when `format = "l"`.

References

Dice, L.R. 1945. Measures of the amount of ecologic association between species. Ecology 26:197–302.

Nei, M. and W.H. Li. 1979. Mathematical Models for studying genetic variation in terms of restriction endonucleases. Proc. Natl. Acad. Sci. USA. 76:5268–5371.

Reif, J., Melchinger, A. and Frisch, M. 2005. Genetical and mathematical properties of similarity and dissimilarity coefficients applied in plant breeding and seed bank management. Crop Sci.45:1-7.

Rogers, J.S. 1972. Measures of genetic similarity and genetic distance. VII. Univ. Tex. Publ. 7213:145–153.

Wright, S. 1978. Evolution and the Genetics of Populations. Volume 4: Variability Within and Among Natural Populations. University of Chicago Press, Chicago, Illinois.

`st.genetic.distances.02`

Calculate genetic distances from SelectionTools marker data

Description

Calculate genetic distances from SelectionTools marker data.

Usage

```
st.genetic.distances.02(measure = "mrd", format = "l",
                        filename = NULL, auxfiles = FALSE,
                        data.set = "default")
```

Arguments

measure	Distance or similarity measure.
format	Input or output format.
filename	Character string giving the base name of the genetic-distance file. Required when auxfiles = TRUE; the file is written with extension '.gdi' in the output directory.
auxfiles	Logical. If TRUE, a persistent output file with the genetic distances is generated and filename must be supplied.
data.set	Name of the SelectionTools data set.

Value

Invisibly returns either an object of class "dist" when format = "m", representing pairwise genetic distances, or a long-format data.frame with columns OTU1, OTU2, and Measure when format = "l".

---

st.genetic.distances.fct

Calculate genetic distances from SelectionTools marker data

Description

Calculate genetic distances from SelectionTools marker data.

Usage

```
st.genetic.distances.fct(measure = "mrd", format = "l", split = 1,
                         filename = NULL, auxfiles = FALSE,
                         data.set = "default")
```

Arguments

measure	Distance or similarity measure.
format	Input or output format.
split	Split or partition control.
filename	Character string giving the base name of the genetic-distance file. Required when auxfiles = TRUE; the file is written with extension '.gdi' in the output directory.
auxfiles	Logical. If TRUE, a persistent output file with the genetic distances is generated and filename must be supplied.
data.set	Name of the SelectionTools data set.

Value

Invisibly returns either an object of class "dist" when format = "m", representing pairwise genetic distances, or a long-format data.frame with columns OTU1, OTU2, and Measure when format = "l".

---

st.get.info.level *Return the current SelectionTools information level*

Description

Return the current SelectionTools information level.

Usage

```
st.get.info.level()
```

Value

An integer scalar giving the current SelectionTools information level.

---

st.get.map *Return the linkage map for a SelectionTools data set*

Description

Return the linkage map for a SelectionTools data set.

Usage

```
st.get.map(filename = "map", data.set = "default")
```

Arguments

filename	Character string retained for compatibility and used as part of the name of the internal session-temporary transfer file. No output file is retained.
data.set	Name of the SelectionTools data set.

Value

Invisibly returns a data.frame with columns Chrom, Pos, Name, and Class describing the map of data.set; returns NULL if no map can be returned.

st.get.num.threads *Return the configured number of computational threads*

Description

Return the configured number of computational threads.

Usage

```
st.get.num.threads()
```

Value

An integer scalar giving the configured number of computational threads.

st.get.simpop *Return Simulation-Population Data in SelectionTools Format*

Description

Imports a diploid simulation population into a SelectionTools marker data set.

Usage

```
st.get.simpop(pop.name, data.set ="default")
```

Arguments

pop.name	Character string naming the simulation population to import. For this transfer the population name must contain 2 to 255 ASCII letters or digits.
data.set	Character string naming the SelectionTools marker data set to create or replace.

Details

The simulation population is converted to the two-allele marker matrix representation used by SelectionTools. The simulation genome must be diploid. Marker identities and loci are obtained from the active simulation map. Marker names transferred from that map must contain only ASCII letters and digits; invalid names are rejected rather than modified. Simulation missing alleles are converted to the SelectionTools internal missing representation, and marker statistics are calculated for the imported matrix. Non-missing simulation allele codes must lie between 1 and 254 to be transferred into a SelectionTools marker data set; wider simulation allele codes are rejected rather than narrowed or wrapped.

The transfer creates new marker-data individual identifiers rather than recovering names that may have existed before the population entered simulation. For a population named P1, the generated identifiers are P1I1, P1I2, and so on. The separator I is alphanumeric so the resulting names obey the

marker-data/STvcf convention that individual identifiers contain only letters and digits. Population names containing dots, punctuation, whitespace, or other non-alphanumeric characters are rejected by this transfer rather than being silently sanitized.

The two homologues returned by the simulation backend are chromosome-homologue ordered. Consequently the resulting SelectionTools marker data set is marked as phased.

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of copying a simulation population into the named SelectionTools marker data set.

---

st.get.simpop.perfdata

Return simulation-population data in SelectionTools format

Description

Imports a simulation population as marker data and uses an evaluated simulation effect as the phenotype vector of the SelectionTools data set.

Usage

```
st.get.simpop.perfdata(pop.name, data.set, effect)
```

Arguments

`pop.name`	Character string naming the simulation population.
`data.set`	Character string naming the target SelectionTools marker data set.
`effect`	Effect name or effect specification passed to population evaluation.

Details

The function first imports the simulation population into `data.set`. It then ensures that the simulation population is genotyped, evaluates the requested effect, retrieves the resulting population genetic values, and writes those values into the phenotype vector of the imported SelectionTools marker data set.

The operation therefore combines a simulation-to-marker-data transfer with generation of performance data from the simulation effect model. The diploidy and naming requirements of the simulation-population importer apply. In particular, the population name used by this transfer must contain only ASCII letters and digits, because `st.get.simpop()` constructs alphanumeric marker-data individual identifiers from that name.

Value

An invisible integer status value returned by `gs.set.performance.data()` after simulation-population performance data have been transferred to the SelectionTools data set.

st.get.simpop2 *Return Simulation-Population Marker Data without a Linkage Map*

Description

Imports diploid simulation-population marker genotypes into a SelectionTools marker data set using the alternative simulation transfer path.

Usage

```
st.get.simpop2(pop.name, data.set ="default")
```

Arguments

pop.name	Character string naming the simulation population. For this transfer the population name must contain 2 to 255 ASCII letters or digits.
data.set	Character string naming the SelectionTools marker data set.

Details

The transfer is restricted to diploid simulation data. Generated individual identifiers use the same convention as `st.get.simpop()`: a population P1 produces P1I1, P1I2, and so on. Population names and marker names transferred from the simulation data must contain only ASCII letters and digits. Invalid names are rejected rather than modified.

The two homologues returned by the simulation backend are chromosome-homologue ordered. Consequently the resulting SelectionTools marker data set is marked as phased.

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of copying simulation-population marker data without a linkage map into the named SelectionTools data set.

st.id *Construct a path in the SelectionTools input directory*

Description

Construct a path in the SelectionTools input directory.

Usage

```
st.id(filename)
```

Arguments

filename	File name.

Value

A character scalar containing the constructed path in the SelectionTools input directory.

---

st.indir *Construct a path in the SelectionTools input directory*

Description

Construct a path in the SelectionTools input directory.

Usage

```
st.indir(filename)
```

Arguments

filename	File name.

Value

A character scalar containing the constructed path in the SelectionTools input directory.

---

st.info *Print or handle a SelectionTools information message*

Description

Print or handle a SelectionTools information message.

Usage

```
st.info(lev, msg)
```

Arguments

lev	Information or verbosity level.
msg	Message text.

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of issuing the requested SelectionTools information message.

st.keep.intersection.markers *Keep the Marker Intersection of Two SelectionTools Data Sets*

Description

Restricts two SelectionTools marker data sets to markers present in both data sets.

Usage

```
st.keep.intersection.markers(data.set1, data.set2)
```

Arguments

data.set1	One non-empty character string naming the first marker data set.
data.set2	One non-empty character string naming the second marker data set.

Details

Marker matching is by exact marker name. Both data sets are restricted to the intersection, and the second data set is reordered so that the common markers occur in the same order as in data.set1. Duplicate marker names are rejected.

Individual names and genotype calls are retained. Phenotypic data are retained when present. If a data set has a linkage map, the corresponding map entries for the retained markers are preserved and reordered with the marker matrix. Marker statistics are recalculated. Marker-order-dependent derived structures such as fitted model objects, haplotype definitions, and cached chromosome statistics are invalidated when a data set must be rebuilt.

The operation is transactional across the two data sets: both restricted replacements are constructed before either existing data set is changed. If construction fails, both original data sets remain unchanged. If the two arguments name the same data set, no change is made.

Value

Invisibly returns the number of markers in the intersection.

st.LDheatmap *Plot a Linkage Disequilibrium Heat Map*

Description

Plots a triangular heat map of pairwise linkage disequilibrium (LD) values using only the standard R graphics system. Marker cells are equally spaced in the LD matrix. If marker positions are supplied, their genetic or physical spacing is displayed separately on a map line connected to the matrix.

Usage

```
st.LDheatmap(LD, map = NULL,
             distances = "genetical",
             LDmeasure = "r",
             title = "",
             col = grDevices::heat.colors(20),
             zlim = c(0, 1),
             add.map = TRUE,
             add.key = TRUE,
             depth = Inf,
             max.dist = Inf,
             write.ld = NULL,
             write.snp.id = FALSE,
             text = FALSE,
             digits = 2,
             geneMapLocation = 0.15,
             geneMapLabelX = NULL,
             geneMapLabelY = NULL,
             cex.title = 0.9,
             cex.map = 0.7,
             cex.key = 0.6,
             cex.ld = NULL,
             cex.snp = NULL)
```

Arguments

LD	A numeric square matrix containing pairwise LD values. The marker order is given by the row and column order of the matrix. Values from the upper triangle are plotted. If an upper-triangle entry is NA and the corresponding lower-triangle entry is available, the lower-triangle value is used. The diagonal is not plotted.
map	A numeric vector containing one map position for each marker, in the same order as the rows and columns of LD. Positions must be finite and in non-decreasing order; equal positions are allowed. map is required when add.map = TRUE or when max.dist is finite.
distances	Character string specifying the units represented by map. "genetic" and "genetical" specify genetic positions in centiMorgans (cM). "physical" specifies physical positions in base pairs. For physical positions, the total map length is reported in kilobases. The spelling "genetical" is accepted as an alias for "genetic".
LDmeasure	Character string used to label the color key. Values "r", "r2", and "r^2" produce an R^2 label. Values "Dp", "Dprime", "D'", and "D" produce a D' label. Other strings are used literally. This argument does not calculate or transform LD values.
title	A single character string giving the plot title. An empty string omits the title.
col	A vector of at least two valid R colors. The first supplied color is used for the highest LD values and the last supplied color for the lowest LD values. Thus grDevices::heat.colors(20) gives high LD in red and low LD in pale yellow.

`zlim` Numeric vector of length two giving the lower and upper limits of the LD color scale. The default is `c(0, 1)`. Values outside this interval generate an error, apart from tiny floating-point deviations at the limits, which are clipped to the limits.

`add.map` Logical. If `TRUE`, draw a line representing the supplied marker positions and connect each marker in the equally spaced LD matrix to its position on that line.

`add.key` Logical. If `TRUE`, draw the color key.

`depth` A non-negative integer or `Inf`. Only marker pairs separated by at most `depth` marker intervals are displayed. For example, `depth = 1` displays only adjacent marker pairs. The default `Inf` applies no restriction based on marker index.

`max.dist` A non-negative number or `Inf`. Only marker pairs separated by at most `max.dist` map units are displayed. The units are the units of `map`: cM for a genetic map and base pairs for a physical map. A finite value requires `map`. The default `Inf` applies no restriction based on map distance.

`write.ld` `NULL` or a function used to construct text labels for displayed LD values. The function is called separately for each displayed value and must return one printable value. Returning `""` or `NA` suppresses the label for that cell. For example, `function(x) if (x >= 0.8) sprintf("%.2f", x) else ""` prints only LD values of at least 0.8. If supplied, `write.ld` takes precedence over `text` and `digits`.

`write.snp.id` Logical. If `TRUE`, print marker identifiers. Row names of `LD` are used if available, otherwise column names, otherwise the marker numbers `1, ..., n`. For densely spaced markers, labels can overlap; `cex.snp` can be supplied to control their size.

`text` Logical convenience option. If `TRUE` and `write.ld` is `NULL`, all displayed LD values are printed using a fixed number of decimal places specified by `digits`.

`digits` Non-negative integer giving the number of decimal places used when `text = TRUE` and `write.ld = NULL`.

`geneMapLocation` Non-negative numeric value controlling the perpendicular distance of the map line from the matrix diagonal. Larger values move the map line farther from the matrix.

`geneMapLabelX` Optional finite numeric x-coordinate for the text reporting the total map length. If `NULL`, a value of 0.52 is used.

`geneMapLabelY` Optional finite numeric y-coordinate for the text reporting the total map length. If `NULL`, a value of 0.24 is used.

`cex.title` Positive finite numeric character expansion factor for the plot title.

`cex.map` Positive finite numeric character expansion factor for the map-length label.

`cex.key` Positive finite numeric character expansion factor for the color-key title and tick labels.

`cex.ld` `NULL` or a positive finite numeric character expansion factor for LD values printed in the cells. If `NULL`, a size is estimated from the cell size and the labels to be printed.

`cex.snp` `NULL` or a positive finite numeric character expansion factor for marker identifiers. If `NULL`, a size is estimated from the number of markers and the identifier lengths.

Details

st.LDheatmap is a plotting function; it does not calculate LD. A precomputed pairwise LD matrix is supplied in LD. In a typical SelectionTools workflow the matrix and map can be obtained with st.LDplot.ld and st.LDplot.map, respectively.

The heat-map cells represent marker order rather than map distance. The marker centers are therefore equally spaced from zero to one along both matrix axes. When add.map = TRUE, the actual values in map are rescaled to a separate line parallel to the matrix diagonal, and each matrix position is connected to its corresponding map position. This keeps the LD cells equally sized while still displaying uneven marker spacing. Markers with identical map positions are supported and are connected to the same position on the map line.

Only the upper triangle of LD is displayed, and the diagonal is omitted. An upper-triangle value is authoritative when it is present. If it is missing and the corresponding lower-triangle value is non-missing, that lower-triangle value is copied to the displayed upper triangle. Missing LD values remain blank. The lower triangle is otherwise ignored.

The restrictions specified by depth and max.dist are applied simultaneously. A cell is displayed only if it satisfies both restrictions. The restrictions affect only which LD cells are drawn; they do not change the marker order or the map line.

The color scale covers zlim. The order of col is from high LD to low LD. This convention makes heat.colors() produce the customary red for high LD and yellow for low LD. Missing or filtered cells are not painted.

If write.ld is supplied, it is evaluated once for each displayed LD value. This permits selective labels without changing the plotted matrix. If cex.ld = NULL, the function estimates a label size intended to fit into one heat-map cell. Automatic sizing is necessarily approximate on very dense plots; an explicit cex.ld can be supplied when exact control is needed. The same consideration applies to marker identifiers and cex.snp.

The function uses standard R graphics and starts one new plot in the current graphics layout. It does not modify global par settings. Consequently, standard layouts such as par(mfrow = c(2, 2)) can be used to place several LD heat maps on one graphics device.

Value

An invisible named list with components:

LDmatrix	The upper-triangular LD matrix actually displayed after the plotting restrictions are applied.
map	The supplied map vector, or NULL.
snp.position	Plotting coordinates of markers in the LD matrix.
map.x, map.y	Plotting coordinates for the map line, or NULL when no map is added.
distances	The map-distance type used internally.
depth	The marker-depth restriction.
max.dist	The map-distance restriction.
zlim	The color-scale limits.
col	The color vector used for the heat map.

See Also

st.calc.ld, st.LDplot.ld, st.LDplot.map

Examples

```
LD <- matrix(c(
    1.00, 0.85, 0.35, 0.10, 0.05,
    0.85, 1.00, 0.70, 0.25, 0.10,
    0.35, 0.70, 1.00, 0.80, 0.30,
    0.10, 0.25, 0.80, 1.00, 0.90,
    0.05, 0.10, 0.30, 0.90, 1.00
), nrow = 5, byrow = TRUE)
rownames(LD) <- colnames(LD) <- paste0("m", 1:5)
map <- c(0, 5, 17, 18, 35)

st.LDheatmap(LD, map,
             title = "Pairwise LD",
             col = grDevices::heat.colors(20))

st.LDheatmap(LD, map,
             depth = 2,
             write.ld = function(x)
                 if (x >= 0.7) sprintf("%.2f", x) else "",
             write.snp.id = TRUE)

st.LDheatmap(LD, map,
             max.dist = 10,
             add.key = FALSE)
```

---

st.LDplot.ld *Prepare linkage-disequilibrium information for plotting*

Description

Prepare linkage-disequilibrium information for plotting.

Usage

```
st.LDplot.ld(ld, chrom)
```

Arguments

ld	Function argument used by this operation.
chrom	Chromosome identifier.

Value

A numeric matrix containing the linkage-disequilibrium values arranged for plotting.

st.LDplot.map *Prepare linkage-disequilibrium information for plotting*

Description

Prepare linkage-disequilibrium information for plotting.

Usage

```
st.LDplot.map(chrom, data.set)
```

Arguments

chrom	Chromosome identifier.
data.set	Name of the SelectionTools data set.

Value

A numeric vector containing map positions for the requested chromosome.

st.load.performance.data *Load Performance Data from an R Data Set*

Description

Matches performance records supplied in an R object to the individuals of an existing SelectionTools marker data set without intermediate text files.

Usage

```
st.load.performance.data(performance.data, data.set="default")
```

Arguments

performance.data	A data frame or matrix containing individual identifiers and numeric performance values.
data.set	Character string naming the SelectionTools marker data set.

Details

Marker data must already be present in `data.set`. Three input-column conventions are accepted. Columns named `individual` and `performance` are used when both are present; otherwise columns `ii` and `yy` are used when both are present; otherwise an object with exactly two columns is interpreted as identifier followed by performance.

Performance individual IDs must contain only ASCII letters and digits and fewer than 256 bytes. The same name convention is used for STvcf individual identifiers. Invalid IDs are rejected rather than being silently ignored as unmatched records.

Records are matched by individual identifier. Performance records whose valid IDs do not occur in the current marker data are ignored. Marker-data individuals with no matched performance record are removed. If an identifier occurs more than once in the performance input, the last matched value is used, following the existing performance-input convention. At least one record must match.

Performance values must be finite numeric values after conversion. The resulting marker data set retains the current marker names and linkage map, subsets both allele arrays to individuals with performance, recalculates marker statistics for the retained individuals, and stores the matched performance vector. The operation is transactional.

Value

`NULL`, returned invisibly. The function is called for its side effect of loading performance data into the named SelectionTools data set.

See Also

`st.read.performance.data` for reading performance data from a local file.

Examples

```
# Load marker data first
data("v-tropmaize-vcf")
st.load.vcf.data(v.tropmaize.vcf, data.set = "maize")

# Performance values for three individuals in the marker data
performance <- data.frame(
   individual = v.tropmaize.vcf$individual[1:3],
   performance = c(8.2, 7.5, 9.1)
)

st.load.performance.data(performance, data.set = "maize")
```

---

st.load.vcf.data *Load Compact STvcf Marker and Linkage-Map Data*

Description

Loads a compact STvcf object directly into a SelectionTools marker data set and installs its linkage map without intermediate marker or map text files.

Usage

```
st.load.vcf.data(STvcf, data.set="default")
```

Arguments

STvcf	A list in STvcf format.
data.set	Character string naming the SelectionTools marker data set to be created or replaced.

Details

STvcf is a compact exchange representation for diploid marker data with VCF-style allele indexing and one genetic position per marker. Loading converts it into the existing SelectionTools marker and linkage-map data structures; no parallel internal analysis representation is introduced.

The list must contain exactly the following nine components, each exactly once. Extra, missing, duplicated, unnamed, or unknown list components are rejected by the C loader:

CHROM Integer running chromosome codes, consecutive from 1.

POS Double-precision genetic positions in cM. Values must be finite and nonnegative.

ID Unique marker identifiers containing only ASCII letters and digits and fewer than 256 bytes.

REF One reference-allele label per marker. REF may not be missing, empty, ".", or comma-separated.

ALT "." or a comma-separated alternative-allele list. ALT entries must be non-empty, unique, and different from REF.

individual Unique individual identifiers containing only ASCII letters and digits and fewer than 256 bytes.

allele1 Marker-by-individual raw or integer matrix containing the first VCF allele index.

allele2 Marker-by-individual raw or integer matrix containing the second VCF allele index.

CHROM.levels Character labels associated with the running chromosome codes.

Individual names must be unique, contain only letters and digits (no dots, hyphens, etc.), and have between 1 and 255 characters.

Alleles use VCF-style indexing: 0 denotes the reference allele (REF), and 1, 2, and subsequent indices refer to the alleles listed in ALT. When loaded into SelectionTools, these are represented by allele codes 1, 2, 3, and so on.

Prepare the input object using st.dataframe.to.STvcf. This function converts a data frame of marker information and genotypes into the STvcf format required here.

Each marker must have one reference allele. Alternative alleles must be unique and differ from the reference allele. Use "." for ALT when no alternative allele is defined; empty entries and dots within a list of alternative alleles are not allowed.

Markers may have up to 254 alleles, excluding missing values. Multiallelic markers can be loaded, but genomic prediction methods requiring biallelic markers must be used with a suitable subset. Monomorphic markers must also be excluded before using these methods.

Only diploid genotypes are supported. If either allele is missing, the entire genotype is treated as missing. The order of non-missing alleles is preserved. The logical phased attribute indicates

whether the supplied genotypes are phased. If this attribute is absent, the data are treated as unphased. The attribute must be logical and must not contain missing values. Loading does not infer phase.

All markers must have genetic map positions in centimorgans. Markers are ordered by chromosome and position. Markers sharing the same position retain their relative input order, but their positions are adjusted slightly to make them distinct. The supplied STvcf object is not modified. Use `st.get.map` to inspect the map positions used by SelectionTools.

Marker statistics are calculated from the supplied data. An existing data set named `data.set` is replaced only if loading succeeds. Otherwise, it remains unchanged. Performance data are not loaded by this function.

Value

`NULL`, returned invisibly. The function is called for its side effect of loading the supplied STvcf marker and map information into `data.set`.

See Also

`st.dataframe.to.STvcf`, `st.markerdata.to.STvcf`, `st.STvcf.to.dataframe`.

Examples

```
# Three markers and three individuals.
# POS contains genetic positions in cM.
# 0 denotes REF and 1 denotes ALT.
markers <- data.frame(
   CHROM = c(1, 1, 2),
   POS = c(5, 15, 10),
   ID = c("M1", "M2", "M3"),
   REF = c("A", "C", "G"),
   ALT = c("G", "T", "A"),
   Plant1 = c("0/0", "0/1", "1/1"),
   Plant2 = c("0/1", "1/1", "0/0"),
   Plant3 = c("1/1", "0/0", "0/1")
)

# Convert the data frame to the compact STvcf format
marker.data <- st.dataframe.to.STvcf(markers)

# Load marker genotypes and their map into SelectionTools
st.load.vcf.data(marker.data, data.set = "example")

# Inspect the loaded linkage map
mp <- st.get.map(data.set = "example")
mp
```

st.mark.alleles *Identify markers by effect and replace selected marker alleles in marker data*

Description

Identify markers by effect and replace selected marker alleles in marker data.

Usage

```
st.mark.alleles(markers = NULL, data.set="default", effect.set=NULL,
                target.set, hap.lst = NULL, alpha=0.05,
                p.adjust.method="none", replace = "3")
```

Arguments

markers	Function argument used by this operation.
data.set	Name of the SelectionTools data set.
effect.set	Data set supplying marker effects.
target.set	Target data set.
hap.lst	Function argument used by this operation.
alpha	Method-specific tuning or significance parameter.
p.adjust.method	Function argument used by this operation.
replace	Logical or coding value controlling replacement.

Value

On successful completion, invisibly returns the character value used to restore the previous SelectionTools input directory. Validation failures return NULL invisibly. The main effect is the requested allele recoding and marker-data update.

st.marker.data.statistics *Summarize a SelectionTools Marker Data Set*

Description

Recomputes and optionally returns summary statistics for the current marker matrix.

Usage

```
st.marker.data.statistics(data.set ="default", filename ="marker.stats",
                          ind = TRUE, mar = TRUE, gen = TRUE, auxfiles = TRUE)
```

Arguments

data.set	Character string naming the marker data set.
filename	Character string retained for compatibility and used as part of the names of internal statistics files in the R session temporary directory. These temporary files are removed before the function returns.
ind	Logical. Request the individual missingness table.
mar	Logical. Request the marker statistics table.
gen	Logical. Request the genotype matrix table.
auxfiles	Logical. If TRUE, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.

Details

Statistics are calculated directly from the two stored allele matrices. For each marker the function determines the distinct biological allele codes, allele counts, the proportion of missing allele copies, and expected heterozygosity calculated as one minus the sum of squared non-missing allele frequencies.

For each individual, the proportion of markers with a missing first allele is reported as the individual missingness statistic. These statistics are also the values used by marker-data restriction functions.

The returned individual.list has columns Name and InMis, where InMis is the individual missing-marker frequency.

The returned marker.list contains Name, NoAll, MaMis, and ExHet. NoAll is the number of observed non-missing biological alleles, MaMis is the marker missing-allele frequency, and ExHet is the expected heterozygosity. The remaining columns contain allele copy counts. If missing alleles occur anywhere in the data set, column AM gives the number of missing allele copies at each marker. Columns A1, A2, and so on give the counts of the corresponding biological allele codes observed in the data set. A marker for which a particular code is absent has count zero in that column.

The returned genotypes component is the complete marker-by-individual genotype table. Its first column identifies the marker and the remaining columns are named by individual. Diploid entries are printed as allele1/allele2. If the first stored allele is missing, the entry is printed as --. 1 represents the REF allele, all ALT alleles are numbered starting with 2.

Value

Invisibly returns a named list containing the requested marker-data summaries. Depending on ind, mar, and gen, components are individual.list, marker.list, and/or genotypes, each a data.frame; NULL is returned if no result can be produced.

See Also

st.restrict.marker.data, st.plot.gene.diversity

Examples

```
# Two markers and three individuals
markers <- data.frame(
   CHROM = c(1, 1),
   POS = c(5, 10),
   ID = c("M1", "M2"),
   REF = c("A", "C"),
   ALT = c("G", "T"),
   Plant1 = c("0/0", "0/1"),
   Plant2 = c("0/1", "1/1"),
   Plant3 = c("1/1", "./.")
)

st.load.vcf.data(
   st.dataframe.to.STvcf(markers),
   data.set = "example"
)

stats <- st.marker.data.statistics(data.set = "example")
stats$genotypes

# Expected genotype entries:
# Marker  Plant1  Plant2  Plant3
# M1      1/1     1/2     2/2
# M2      1/2     2/2     --
#
# 1 = REF, 2 = first ALT, -- = missing genotype
```

---

st.markerdata.to.STvcf

Convert a SelectionTools Marker Data Set to STvcf

Description

Converts marker genotypes and the linkage map of an existing SelectionTools marker data set to the compact diploid STvcf representation.

Usage

```
st.markerdata.to.STvcf(data.set="default")
```

Arguments

data.set	Character string naming the SelectionTools marker data set to be converted.

Details

This function supplies the marker-data to STvcf step of the reverse data chain. With the existing `st.get.simpop()` function, a simulation population can first be returned to the ordinary SelectionTools marker-data structure and can then be converted to STvcf without introducing a separate simulation-to-STvcf bridge.

The marker data set must contain genotype data and a usable linkage map. Every marker-matrix column must have exactly one active map location. Map-only locations are ignored. Missing or duplicate marker locations, inconsistent marker and map names, invalid chromosome numbers, non-finite positions, and negative positions cause conversion to fail.

The current marker order is retained. Map positions are returned as the current double-precision cM values without intentional rounding. Current numeric chromosome numbers are converted to consecutive STvcf running codes 1, 2, ..., in order of first appearance, while `CHROM.levels` records the corresponding numeric chromosome labels as character strings.

Current marker and individual names become STvcf identifiers. They must be unique, contain fewer than 256 bytes, and contain only ASCII letters and digits. Dots and all other punctuation are rejected. Names are never silently modified during this conversion.

When the marker data came from a simulation population through `st.get.simpop()`, that existing transfer deliberately creates new individual identifiers. In the revised convention a population named P1 produces names P1I1, P1I2, and so on. The population name used by `st.get.simpop()` must itself contain only letters and digits so that the generated individual identifiers satisfy the marker-data/STvcf name rules. Original individual names that existed before simulation are not reconstructed; this name change is an intended feature of the current simulation-to-marker-data transfer.

The SelectionTools marker structure stores integer allele states but not the original textual VCF REF/ALT labels. The converter therefore normalizes the observed non-missing allele states separately for each marker. States are ordered by their current integer values. The first state becomes `REF="1"`; further states become `ALT="2,3,..."`; genotypes are converted to VCF indices 0, 1, 2, ... in that order.

For marker data originally loaded from STvcf, internal alleles are already 1 for REF, 2 for ALT1, 3 for ALT2, and so on. Transfer to and from simulation preserves those integer states. For marker data created by other routes, distinct genotype states are retained but historical integer labels are normalized.

An all-missing marker is retained with `REF="1"`, `ALT="."`, and missing genotypes. A marker with one observed non-missing state is likewise written with `ALT="."` and uses VCF allele index 0. If either homologue of a marker-data genotype is missing, the exported STvcf genotype is completely missing, consistent with the STvcf partial-missing convention.

The two current marker-data allele arrays are retained in order as `allele1` and `allele2`. The returned STvcf object receives a logical `phased` attribute from the native marker data set. For data obtained through `st.get.simpop()`, the two arrays correspond to the two simulation homologues and the attribute is `TRUE`. No new phasing calculation is performed.

The current SelectionTools marker representation permits at most 254 biological allele states per marker. After normalization their VCF indices are therefore at most 253, so this converter writes raw STvcf allele matrices with raw value 255 for missing. Marker genotypes that violate the SelectionTools allele-range or missingness invariants are rejected.

The linkage-map class is not part of STvcf. Exact map-position ties may have already been dispersed by the SelectionTools linkage-map or simulation machinery; this function returns the current stored positions and cannot reconstruct an earlier duplicate position that is no longer represented.

Only marker genotype and genetic-map information are returned. Performance data, estimated effects, genomic relationship matrices, selection values, simulation GValue/PValue arrays, and other derived objects are outside STvcf.

Value

A `list` of class `"STvcf"` containing the current marker data, individual identifiers, marker metadata and chromosome-label mapping. If conversion fails, `NULL` is returned.

`st.mxd` *Fit the package multi-kernel mixed-model routine*

Description

Fit the package multi-kernel mixed-model routine.

Usage

```
st.mxd(y, X, Z = NULL, V = NULL, estimates = FALSE, maxcyc = 100,
       precision = 1e-4)
```

Arguments

`y`	Response vector.
`X`	Fixed-effect design matrix.
`Z`	Marker or design matrix.
`V`	Function argument used by this operation.
`estimates`	Function argument used by this operation.
`maxcyc`	Maximum number of fitting cycles.
`precision`	Convergence tolerance.

Value

A named `list` on successful fitting, with components:

`sigma`	Numeric vector of variance-component estimates.
`b`	Fixed-effect estimates when `estimates = TRUE`; otherwise the scalar initialization value used by the routine.
`u`	Random-effect estimates for `Z`-based models when `estimates = TRUE`; otherwise the scalar initialization value used by the routine.

Returns `NULL` when input validation or the compiled fit reports failure.

st.od — *Construct a path in the SelectionTools output directory*

Description

Construct a path in the SelectionTools output directory.

Usage

```
st.od(filename)
```

Arguments

filename	File name.

Value

A character scalar containing the constructed path in the SelectionTools output directory.

st.outdir — *Construct a path in the SelectionTools output directory*

Description

Construct a path in the SelectionTools output directory.

Usage

```
st.outdir(filename)
```

Arguments

filename	File name.

Value

A character scalar containing the constructed path in the SelectionTools output directory.

st.phase *Cohort and Reference-Assisted Phasing of Diploid Genotypes*

Description

Infers haplotypes at existing markers from the cohort or from fixed reference haplotypes. Missing genotypes at typed markers can optionally be filled. Observed unordered genotypes are preserved. Automatic window selection uses the genetic map and the spacing of common markers.

Usage

```
st.phase(data.set = "default", ref = NULL, phasing.options = list(),
         impute = FALSE, commit = TRUE)
```

Arguments

data.set	Name of an existing SelectionTools marker data set with a linkage map. Genetic positions must be in centiMorgans. All individuals and all chromosomes in this data set are phased.
ref	Reference panel, either the name of a loaded SelectionTools data set or an STvcf object. With NULL, the cohort supplies donor haplotypes. Reference haplotypes remain fixed. See the reference requirements below. Default: NULL.
phasing.options	Uniquely named list of controls described below. Omitted entries use their defaults. Hyphens in option names are accepted as alternatives to dots, for example "phase-states". Unknown or duplicate entries are errors.
impute	Logical scalar. If FALSE, preserve missing target genotypes. If TRUE, retain inferred hard calls at missing genotypes of existing target markers. Observed unordered genotypes are preserved under both policies. Default: FALSE.
commit	Logical scalar. If TRUE, write the completed result to data.set and mark it phased. If FALSE, return the result without changing the data set. Default: TRUE.

Details

Cohort and reference modes. With ref = NULL, the target cohort supplies the donor haplotypes. With a reference panel, both target and reference haplotypes can serve as donors, but only target individuals are updated. References remain fixed during initialization, common-marker iterations, rare-marker reconstruction and window processing. EM estimation uses targets; target relatedness exclusions do not remove reference donors. This is a general copying model: there is no F2-specific ancestry model or pedigree option.

The implementation selects common markers, initializes haplotypes, searches for donors with the positional Burrows–Wheeler transform, iterates a reduced haplotype-copying hidden Markov model, and reconstructs remaining rare-marker alleles. Chromosomes and overlapping windows are processed in genetic-map order. Sample work within several stages runs in parallel. More threads do not remove the sequential dependencies between stages.

Input requirements. There must be 2–5000 total individuals, counting both targets and references; a reference-assisted call may contain just one target. Each chromosome must have at least two usable markers and a positive genetic span. Each sample must have at least one observed call on each chromosome. Marker IDs must be unique. The existing STvcf converter requires alphanumeric marker and individual names. The map must be ordered by chromosome and nondecreasing position. Common-stage markers must be monomorphic or biallelic; multiallelic markers can be handled by the rare-marker stage, up to 100 declared alleles. A selected multiallelic common marker is rejected. Small or sparse windows can fail these conditions; use automatic windows or a larger explicit window.

Options and defaults.

`window = "auto"` Positive window length in cM, a vector with one entry per chromosome, `Inf` for whole-chromosome processing, or `"auto"` (the default) for the spacing rule below. Vector entries follow the chromosome order in the exported input; names are not used for matching.

`overlap` Nonnegative overlap in cM; scalar or one value per chromosome. Automatic windows select overlap from marker spacing when it is omitted. Explicit windows use 2 cM when it is omitted. Finite windows must be at least 1.1 times their overlap.

`window.markers = 4000000L` Integer cap on markers per finite window, at least 4. Explicit `window = Inf` ignores this cap. An automatic whole-chromosome choice exceeding the cap is rejected.

`burnin = 3L` Number of burn-in iterations, an integer at least 1.

`iterations = 12L` Number of subsequent phasing iterations, an integer at least 1.

`phase.states = 280L` Maximum number of copying states; integer from 2 to 1000000. Increasing this can increase time and memory.

`ne = 100000` Positive initial effective population size. EM may update the associated recombination parameter.

`err = NULL` Initial allele mismatch probability. `NULL` selects the data-dependent initializer; an explicit value must lie in $[0, 1]$. EM may update this value.

`em = TRUE` Logical scalar controlling estimation of mismatch and recombination parameters.

`initial.lr = 100000` Initial likelihood-ratio limit, at least 1.

`step.scale = 3` Positive scale for donor-search step spacing.

`rare = 0.002` Positive rare-allele threshold, at most 0.5. The algorithm uses sample carrier counts, with a minimum count boundary of three; this is not simply a filter on alternate-allele frequency.

`buffer = 1` Positive donor-search context buffer in cM.

`seed = NULL` Integer-valued seed in the range $[-2147483647, 2147483647]$. `NULL` draws one seed from R's RNG, so `set.seed()` controls reproducibility. An explicit seed does not consume R random draws. The actual seed is returned in `settings`.

Automatic windows. `window = "auto"` chooses context from the input map and the spacing of common markers, without using phase truth. It targets a block of 400 common markers. Fewer than 800 common markers on a chromosome leads to whole-chromosome processing. Otherwise the window is at least 40 cM and spans the longest local 400-marker interval. Automatic overlap is at least 2 cM and spans the longest local 40-marker interval. A window that spans the chromosome becomes `Inf`. Explicitly supplying `overlap` overrides the adaptive overlap. These are context targets, not a guarantee of a minimum marker count in every terminal window. Automatic windows

are the default. They can use more memory than a fixed 40-cM window when the marker map is sparse.

Reference requirements. Reference markers are matched by unique marker ID and reordered to target order. Chromosome labels must agree and genetic positions must agree within 10^{-6} cM. The panel must contain complete calls at every target marker. Homozygous panels have unambiguous phase. If any reference call is heterozygous, the STvcf must have `attr(ref, "phased") = TRUE`; this is a caller declaration, not independent verification of biological phase.

Declared allele labels are aligned, including reversed REF/ALT order. At least one allele label must be shared at a matched marker. A reference-only allele at an existing target marker can expand its allele definition. Reference-only markers are ignored and reported. Reference individuals may have names also used by targets in a separate data set. A loaded reference must be a different data set from the target when writing back.

Allele coding matters: SelectionTools' existing marker-data exporter normalizes each marker's observed alleles to numeric labels and does not retain original VCF nucleotide labels. Independently imported target and reference panels must already use compatible exported allele coding. In particular, independently normalizing a monomorphic panel can obscure which allele it shares with a polymorphic target. Supplying an STvcf reference does not recover discarded target labels. Check coding before phasing; this interface does not infer strand orientation.

Missing data and write-back. The engine infers complete working haplotypes internally. With `impute = FALSE`, originally missing calls are restored before return and write-back. With `impute = TRUE`, inferred calls at these typed markers are retained without a posterior-confidence cutoff. No untyped markers, dosages, genotype probabilities or allele probabilities are produced.

All computations precede write-back. The complete output is validated and staged before the existing genotype arrays are changed. Original internal allele codes, map and other dataset fields are retained. A failed phasing or validation call does not partially write a chromosome into the source. The returned STvcf uses exported allele coding. References are never written. As with other genotype-changing operations, previously computed analyses should be recomputed when their inputs have changed through imputation.

Threads and reproducibility. Use `st.set.num.threads` and `st.get.num.threads`. There is no `nthreads` entry in `phasing.options`. OpenMP may grant fewer workers than requested; the maximum actual team is reported. Builds without OpenMP execute serially and report one phasing thread. OpenMP must be enabled in the compiler and linker settings used to build the package. A fixed explicit seed gives deterministic output across tested thread counts on the same build and input. Floating-point and compiler differences can affect exact agreement across platforms.

Value

Invisibly returns a list with:

`data` Phased target-only STvcf, with `phased = TRUE`. Missing calls follow `impute`. This is also returned with `commit = FALSE`.

`settings` Mode (`"cohort"` or `"reference"`), imputation policy, the supplied `ref`, effective `phasing.options`, requested `nthreads`, `actual.seed`, per-chromosome `window.settings` and `automatic.window`. Effective `ne` and `err` entries describe initialization controls, not final fitted EM estimates.

`diagnostics` Total `windows`, `chromosome.windows`, `maximum.worker.team`, `filled.genotypes` and `committed`. Reference runs additionally report reference sample and haplotype counts, invariant-check status, and ignored reference markers.

Subsetting

Restrict a copy of the source with `st.restrict.marker.data` before calling `st.phase()` when only a subset of chromosomes or individuals is wanted.

`st.phase.v1` *Statistical Phasing and Missing-Genotype Imputation of Diploid Marker Data*

Description

Phases diploid biallelic SelectionTools marker data either from an external SelectionTools reference data set or by reference-free cohort phasing. Observed heterozygous genotypes receive a complete final phase; posterior confidence is retained for diagnostics and for imputed genotypes, but local ambiguity does not suppress otherwise valid chromosome phase. Completely missing target genotypes can be imputed from posterior genotype probabilities. All proposed changes are constructed in private storage, audited, and by default written back transactionally after a fresh final audit.

Usage

```
st.phase.v1(
    reference.data.set = NULL,
    reference.individuals = NULL,
    initial.pair.weights = NULL,
    population.type = NULL,
    expected.breaks.per.morgan = NULL,
    max.states = NULL,
    error.rate = 0.001,
    break.rate1.per.morgan = NULL,
    break.rate2.per.morgan = NULL,
    confidence.threshold = 0.50,
    impute = TRUE,
    imputation.threshold = 0.90,
    individuals = NULL,
    chromosomes = NULL,
    burnin.max.iterations = 4,
    burnin.min.iterations = 2,
    burnin.convergence.fraction = 0.001,
    progressive.max.iterations = 2,
    progressive.min.iterations = 1,
    progressive.convergence.fraction = 0.001,
    burnin.states = 12,
    local.states = 8,
    stage1.minor.allele.count = 4,
    provisional.threshold = 0.80,
    commit = TRUE,
    data.set = "default"
)
```

Arguments

`reference.data.set`

`NULL` selects reference-free cohort phasing. Otherwise, one non-empty character string naming an existing SelectionTools marker data set that supplies reference haplotypes.

`reference.individuals`

Individuals in `reference.data.set` to use as the reference panel. `NULL` selects all individuals. Otherwise, supply unique individual names or unique one-based indices. This argument is valid only in reference-data-set mode.

All selected reference genotypes must be observed at every target marker. If every selected reference individual is homozygous, one haplotype is taken from each individual. If any selected reference individual is heterozygous, the two stored allele tracks of every selected reference are interpreted as already-phased haplotypes and `reference.data.set` must be marked phased. Data returned from the simulation backend and completely phased by `st.phase.v1()` carry this state automatically; consistently `|`-phased VCF input does as well.

`initial.pair.weights`

Optional nonnegative $R \times R$ matrix of first-marker weights for ordered reference donor pairs. It is valid only in reference mode and the sum of all entries must be positive. When `NULL`, SelectionTools constructs a canonical upper-triangular labeling prior internally. This breaks only the arbitrary global labels of the two copied homologues; it does not exclude an unordered donor pair.

`population.type`

Optional character string selecting a plant-population expectation for ancestry junctions. Accepted values are `"DH"`, `"F2"`, `"F3"`, `"F4"`, `"S2"`, `"S3"`, `"S4"`, `"SSD"`, and `"Finfty"`, case-insensitively.

The corresponding expected breaks per Morgan are 1 for DH and F2, 1.5 for F3 and S2, 2 for F4 and SSD, 1.75 for S3, and 1.875 for S4. `"Finfty"` denotes a general/outbred population and requests a lightweight data-derived copying-break estimate rather than a fixed plant pedigree value.

If `population.type = NULL`, a reference panel consisting of exactly two selected homozygous individuals is recognized as `"F2"`; all other cases default to `"Finfty"`.

`expected.breaks.per.morgan`

Optional nonnegative numeric expert override for the expected ancestry or copying breaks per Morgan. When supplied, this value overrides the value implied by `population.type` and is used for both internal break rates. It cannot be combined with the legacy explicit pair `break.rate1.per.morgan`/`break.rate2.per.morgan`.

`max.states` Positive whole number controlling the bounded donor set. The default is `min(60,R)` in reference mode, where R is the number of reference haplotypes, and 60 in cohort mode. In cohort mode it bounds the target-specific posterior donor subset and final imputation donor set.

`error.rate` Allelic mismatch probability between zero and one. The default is `0.001`. For a biallelic marker, a copied donor allele emits the matching target allele with probability $1 - e$ and the alternative with probability e. The parameter can absorb genotyping error, mutation, and imperfect representation by the donor panel.

break.rate1.per.morgan, break.rate2.per.morgan
: Legacy advanced overrides for nonnegative copying-break intensities per Morgan. They must be supplied together. When either is supplied, population.type must not be supplied explicitly and expected.breaks.per.morgan must be NULL.

  For adjacent markers separated by d centiMorgans, intensity λ gives

$$q = 1 - \exp(-\lambda d/100)$$

  as the probability of a copying break over the interval. In reference mode the two values belong to the two ordered copying chains. In cohort mode rate 1 is used for the Stage-1 scaffold and rate 2 for Stage-2 low-frequency insertion. When the legacy pair is not supplied, both are set to the effective value selected from population.type, expected.breaks.per.morgan, or the "Finfty" estimator.

confidence.threshold
: Probability threshold between 0.5 and 1 used for posterior-supported phase diagnostics and for assigning ordered phase to accepted imputed heterozygotes. The default is 0.50. Numerically indistinguishable orientation probabilities are treated as a tie rather than allowing floating-point roundoff to create a hard phase call. Observed heterozygotes are nevertheless completed from the continuous Viterbi path when an absolute local orientation is tied or falls below this threshold.

impute
: Logical. The default TRUE also decodes completely missing target genotypes from smoothed donor-state posteriors. FALSE phases observed genotypes but leaves missing target genotypes missing.

imputation.threshold
: Posterior genotype-probability threshold between zero and one for accepting a completely missing target genotype. The default is 0.90.

individuals
: Target individuals to phase in reference mode. NULL selects all; otherwise supply unique names or one-based indices. Cohort mode is a whole-data-set procedure and requires NULL.

chromosomes
: Target chromosomes to phase in reference mode. NULL selects all; otherwise supply unique chromosome labels. Cohort mode requires NULL.

burnin.max.iterations
: Maximum number of synchronous Stage-1 Viterbi burn-in passes in cohort mode. Default 4. It must be at least burnin.min.iterations.

burnin.min.iterations
: Minimum number of Stage-1 Viterbi burn-in passes before convergence can stop burn-in. Default 2.

burnin.convergence.fraction
: Cohort burn-in convergence tolerance. After the minimum number of passes, burn-in stops when the fraction of changed consecutive-heterozygote phase relations is no greater than this value. Default 0.001.

progressive.max.iterations
: Maximum number of synchronous Stage-1 posterior-refinement passes in cohort mode. Default 2.

progressive.min.iterations
: Minimum number of Stage-1 posterior-refinement passes before convergence can stop refinement. Default 1.

progressive.convergence.fraction
: Posterior-refinement convergence tolerance. Default `0.001`.

burnin.states
: Maximum number of balanced leave-one-out donor haplotypes retained per marker in the cohort Stage-1 Viterbi burn-in. Default 12.

local.states
: Number of local phase-invariant donor candidates retained per Stage-1 marker while constructing the chromosome-wide target-specific donor ranking. Default 8.

stage1.minor.allele.count
: Minimum observed minor-allele count for a polymorphic biallelic marker to enter the iteratively phased Stage-1 scaffold. Lower-frequency markers are inserted in Stage 2. Default 4.

provisional.threshold
: Probability threshold between 0.5 and 1 for allowing an allele at an originally missing genotype to become visible as a provisional donor allele in later cohort-refinement passes. Default `0.80`. This is distinct from final genotype imputation.

commit
: Logical. The default `TRUE` publishes the phased/imputed marker data transactionally after a fresh final audit. Set `FALSE` for a staging-only run that performs the complete calculation and audit without modifying marker data.

data.set
: One non-empty character string naming the SelectionTools marker data set to phase. The default is `"default"`.

Value

An object of class `"st.phase.result"` and `"list"`. Full marker-wise posterior arrays are not returned. Common diagnostics include:

`success` Logical indicating successful completion.

`mode` `"reference"` or `"cohort"`.

`pipeline.status`, `pipeline.status.name` Pipeline status.

`audit.status`, `audit.status.name` Private-stage audit status.

`commit.requested`, `commit.attempted`, `commit.status`, `commit.status.name`, `committed` Commit diagnostics.

`n.ind`, `n.marker`, `n.chrom`, `n.genotype` Target dimensions.

`n.missing`, `n.homozygous`, `n.heterozygous` Source genotype counts.

`n.written`, `n.changed.genotypes` Staged phase/imputation write counts.

`n.imputation.missing.requested`, `n.imputed`, `n.imputation.unresolved` Missing-genotype imputation counts.

`population.type` Effective population type.

`expected.breaks.per.morgan` Effective single break expectation; `NA` when the legacy two-rate pair was supplied.

`breaks.estimated` Logical indicating whether `"Finfty"` estimation supplied the effective single break rate.

`break.rate1.per.morgan`, `break.rate2.per.morgan` Effective break rates passed to the C phaser.

`error.rate`, `max.states` Effective mismatch rate and donor-state bound.

Reference mode additionally reports reference-panel sizes and reference-mode run diagnostics. Cohort mode additionally reports Stage-1/Stage-2 marker counts, burn-in/refinement iteration diagnostics, provisional-state counts, relative same/switch-link diagnostics, and confidence diagnostics. Under the complete-output policy, low posterior confidence for an observed heterozygote is diagnostic rather than an instruction to leave that observed phase unwritten.

Default population models and break-rate selection

Normal end-user use does not require HMM break-rate tuning. SelectionTools first determines `population.type`. Fixed plant-population types map to expected ancestry-junction densities as follows:

Type	Breaks/Morgan	Interpretation
DH	1.000	doubled haploids directly from an F1
F2	1.000	F2 from an F1
F3	1.500	one additional random-mating generation
F4	2.000	two additional random-mating generations
S2	1.500	two generations of selfing from the F1
S3	1.750	three generations of selfing
S4	1.875	four generations of selfing
SSD	2.000	single-seed descent / near-inbred selfing limit
Finfty	estimated	general or outbred population

For `"Finfty"`, SelectionTools estimates a single copying-break intensity on the same per-Morgan scale as the Li–Stephens transition. The estimator is lightweight and deterministic: it uses at most 500 individuals, common biallelic markers, and a bounded sample of short- to intermediate-distance marker pairs, then fits a one-dimensional LD-decay curve. It does not rerun phasing and does not perform a multi-parameter grid search. The resulting value is returned as `expected.breaks.per.morgan` and copied to both internal break rates unless the legacy pair is explicitly supplied.

Precedence is: the explicit legacy two-rate pair, otherwise `expected.breaks.per.morgan`, otherwise the fixed value implied by `population.type`, with `"Finfty"` triggering estimation.

Phasing modes

A named `reference.data.set` invokes the ordered donor-pair reference HMM. Reference and target markers are matched by marker name. With `reference.data.set = NULL`, the target cohort supplies leave-one-out provisional donor haplotypes and is phased with the two-stage common-marker / low- frequency procedure.

The cohort-only iteration/state arguments are ignored by the C reference pipeline and must not be supplied explicitly in reference mode. In cohort mode `reference.individuals`, `initial.pair.weights`, `individuals`, and `chromosomes` are not accepted.

Scope and restrictions

The public phaser operates on diploid SelectionTools marker data with at most two biological alleles per marker. Completely missing genotypes must be missing on both stored alleles; partial diploid missingness is rejected. Map positions are in centiMorgans and chromosomes are processed separately in map order.

Reference genotypes must be complete at all target markers. Heterozygous reference individuals are accepted only as already-phased diploids. A reference allele absent from the corresponding target marker's biological allele dictionary is rejected rather than silently recoded.

Cohort initialization can use private provisional alleles at completely missing genotypes. Such provisional alleles are never interpreted as observed biological genotypes and are never directly committed as imputation calls.

Statistical theory and implemented algorithm

Notation and copying model: For chromosome markers $m = 1, \ldots, M$ at positions x_m centiMorgans, let $g_m = \{a_m, b_m\}$ be the unordered target genotype and H_{mr} the allele carried by donor haplotype r. Biological SelectionTools alleles are translated to compact per-marker states for phasing and translated back only during staging.

For allelic mismatch probability e, a copied donor allele emits its match with probability $1 - e$; at a biallelic mismatch the probability is e. A completely missing observation is neutral. For interval length $\Delta_m = (x_m - x_{m-1})/100$ Morgans and copying-break intensity λ,

$$q_m = 1 - \exp(-\lambda \Delta_m).$$

A retained donor persists with mass $1 - q_m$ and the break mass q_m is redistributed over the current bounded donor states.

Reference-panel donor selection and ordered diploid HMM: Reference donor selection is invariant to the arbitrary input order of a target heterozygote. Contiguous compatibility runs between each donor haplotype and either allele of the unordered target genotype provide an IBS ranking; at most `max.states` distinct donors are retained per marker.

The hidden diploid state is the ordered donor pair $Z_m = (r_m, s_m)$. The two donor chains transition independently. For a heterozygote,

$$E_m(r, s) = P(a_m \mid H_{mr})P(b_m \mid H_{ms}) + P(b_m \mid H_{mr})P(a_m \mid H_{ms}).$$

Forward/backward posterior probabilities and a Viterbi donor-pair path are computed with factorized $O(MK^2)$ transitions after candidate construction.

complete reference phase: Posterior donor-pair mass is first partitioned between the two possible ordered orientations of each observed heterozygote. Calls that reach `confidence.threshold` are retained only when the two posterior orientation probabilities are numerically distinguishable. This prevents roundoff around an exact label-symmetric 50:50 posterior from creating an artificial hard orientation.

Observed heterozygotes that remain locally unresolved are then completed rather than omitted. The deterministic Viterbi donor-pair traceback provides a continuous labeled chromosome path. At an unresolved marker the two ordered emission components for the Viterbi state are compared; if one is larger it sets the orientation, and an exact tie retains the running chromosome orientation. This continue/swap rule avoids mistaking arbitrary homologue-label symmetry for absence of relative phase. Thus ordinary observed heterozygotes receive complete phase output.

Phased heterozygous reference individuals: When all selected reference individuals are homozygous, one haplotype per individual is used. If at least one selected reference is heterozygous, the reference marker data set must have its phased state set to TRUE and both stored allele tracks of every selected reference individual are used as phased homologues. Hence N_R selected diploids contribute $2N_R$ donor haplotypes in that case.

PBWT initialization of a reference-free cohort: Reference-free mode starts from unordered genotypes and never treats the input allele-1/allele-2 order as known phase. A PBWT-based phase-invariant initialization scores candidate allele assignments from compatible prefix matches to other individuals. Completely missing genotypes receive private provisional major-allele proxies so initialization can continue; the original missingness mask is retained.

Two-stage cohort phasing: Polymorphic markers with observed minor-allele count at least `stage1.minor.allele.count` form Stage 1. Rarer polymorphic markers are withheld and inserted once in Stage 2. Every iterative cohort pass is leave-one-individual-out and synchronous: a pass reads a frozen snapshot and writes a separate next snapshot, so individual processing order cannot alter that pass.

Stage-1 burn-in and posterior refinement: Stage 1 begins with synchronous Viterbi burn-in using at most `burnin.states` balanced phase-invariant donor states. Burn-in stops after at least `burnin.min.iterations` when its changed-link fraction is no greater than `burnin.convergence.fraction`, or at `burnin.max.iterations`.

Posterior refinement then retains `local.states` phase-invariant local candidates per marker, aggregates support across the chromosome, and keeps at most `max.states` target-specific donors. Originally missing donor alleles can enter later frozen panels only when their haploid posterior reaches `provisional.threshold`. Refinement uses analogous minimum, maximum, and convergence controls.

Stage-2 low-frequency insertion and ties: Stage-2 markers are oriented once conditional on the final common-marker scaffold. Haploid donor-state probabilities are computed at neighboring Stage-1 markers and interpolated along genetic distance. For a heterozygous low-frequency genotype, the two ordered allele products are compared.

If one orientation has larger support it is chosen. If the two scores are exactly equal, including a singleton minor allele absent from every leave-one-out donor, a stable 50:50 pseudo-random tie decision based on the individual, marker, and chromosome coordinates chooses one orientation. This resolves Stage-2 ties while preserving bitwise reproducibility across OpenMP thread counts.

Complete cohort output: The Stage-1 relative-phase chain and Stage-2 insertion decisions are always written to the private stage for observed heterozygotes. Confidence flags are retained as diagnostics but do not veto an otherwise valid chromosome or turn a local singleton ambiguity into chromosome-wide missing phase. Consequently normal cohort phasing has complete observed-heterozygote phase coverage.

Posterior missing-genotype imputation: When `impute = TRUE`, final biological imputation is separate from the provisional missing-data bootstrap. In cohort mode the target is excluded and only donor individuals whose biological genotype was originally observed at every marker on that chromosome are eligible for the imputation HMM. If no such donor remains, missing genotypes stay unresolved rather than causing the pipeline to fail.

The smoothed ordered-state posterior is collapsed to probabilities for unordered diploid genotypes. A genotype is accepted only when its largest posterior reaches `imputation.threshold`.

For an accepted imputed heterozygote, ordered phase is attached only when the conditional orientation posterior reaches `confidence.threshold`; otherwise the genotype can be imputed without claiming confident ordered phase.

Transactional staging, audit, and commit: Phasing and imputation operate on private storage. The audit verifies data sizes, marker mapping, map positions, source-byte freshness, preservation of observed unordered biological genotypes, and consistency of staged writes. With `commit = TRUE`, a fresh audit is performed immediately before publication. With `commit = FALSE`, the complete calculation and audit run without changing the source marker data. A successful complete-data-set commit with no unresolved heterozygous phase marks the SelectionTools marker data set as phased. Partial phasing requests do not by themselves promote an otherwise unphased data set to the phased state.

Computational structure and OpenMP: Reference diploid HMM work is $O(MK^2)$ after bounded donor selection. Cohort mode repeats target-specific leave-one-out haploid calculations across a small fixed number of synchronous burn-in/refinement passes and inserts rare markers once. When OpenMP is available, independent target individuals are parallelized within frozen passes and Stage-2 insertion. Chromosomes are not parallelized.

Interpretation of defaults and expert controls

The default interface is intended for end users rather than per-data-set parameter tuning. Known plant population types use fixed biologically interpretable ancestry-junction expectations. General populations use the lightweight `"Finfty"` estimator. The default `error.rate = 0.001`, state bounds, and short convergence-controlled iteration limits are fixed.

`expected.breaks.per.morgan` is the preferred expert override when the expected ancestry-junction density is known. The two legacy break-rate arguments remain available only for advanced/backward-compatible analyses that need distinct chain or stage rates. Increasing state bounds generally uses more donor information at higher computational cost. Increasing `imputation.threshold` reduces the number of missing genotypes accepted.

`st.plot.corr` *Plot correlation information from SelectionTools results*

Description

Plot correlation information from SelectionTools results.

Usage

```
st.plot.corr(x, y, title = "", pch = 16, cex = 1, xlab = "x", ylab = "y")
```

Arguments

`x`	Input object.
`y`	Response vector.
`title`	Function argument used by this operation.

pch	Function argument used by this operation.
cex	Function argument used by this operation.
xlab	Function argument used by this operation.
ylab	Function argument used by this operation.

Value

No useful return value is intended. The function is called for its side effect of plotting correlation information.

`st.plot.corr.l` *Plot correlation information from SelectionTools results*

Description

Plot correlation information from SelectionTools results.

Usage

```
st.plot.corr.l(x, y, title = "", pch = 16, cex = 1, xlab = "x", ylab = "y",
               ylim, xlim)
```

Arguments

x	Input object.
y	Response vector.
title	Function argument used by this operation.
pch	Function argument used by this operation.
cex	Function argument used by this operation.
xlab	Function argument used by this operation.
ylab	Function argument used by this operation.
ylim	Function argument used by this operation.
xlim	Function argument used by this operation.

Value

No useful return value is intended. The function is called for its side effect of plotting correlation information.

st.plot.gene.diversity *st.plot.gene.diversity()*

Description

The function *st.plot.gene.diversity* plots gene diversity along the chromosomes and returns the marker-level values used for the plot.

Usage

```
st.plot.gene.diversity(data.set ="default", plt.pdf = FALSE,
                       plt.fname = NULL, plt.width = 10, plt.height = 6,
                       plt.ptsize = 14)
```

Arguments

data.set	character. Name of the data set that is used by the function.
plt.pdf	logical. If TRUE, a PDF file is generated in the output directory.
plt.fname	Character string giving the base name of the PDF file. Required when plt.pdf = TRUE.
plt.width	Width of the generated PDF file.
plt.height	Height of the generated PDF file.
plt.ptsize	Point size of the generated PDF file.

Details

The gene diversity (expected heterozygosity) at a marker is calculated as

$$1 - \sum_i p_i^2,$$

where p_i is the frequency of allele i at that marker. Gene diversity ranges from 0 to 1. It is 0 when a marker is fixed and increases as allele frequencies become more even; for a given number of alleles it is maximized when the alleles have equal frequencies, and its maximum approaches 1 as the number of equally frequent alleles increases. The values used for plotting are obtained from st.marker.data.statistics(). The average gene diversity for each chromosome is printed at the top of the plot.

Value

Invisibly returns a data.frame with columns chrom, pos, name, cpos, and gene.div, containing the marker-level positions and gene-diversity values used for plotting.

st.plot.ggt *Plot Graphical Genotypes*

Description

Plots graphical genotypes for all individuals or for a selected subset. The individuals can optionally be ordered by performance, and stored performance values can be appended automatically to the individual labels.

Usage

```
st.plot.ggt(data.set = "default", ifilename = "", i.list = "", f.ind = 0,
            l.ind = 0, f.lr = 2, f.tb = 2, d.h = 0.001, d.v = 0.001,
            p.t = FALSE, p.s = FALSE, d.t = 50, d.map = 100,
            c.nme = "", i.nme = "", cex.chrom = 1, cex.ind = 1,
            color = c("yellow", "blue", "red", "green", "wheat",
                      "skyblue", "tomato", "palegreen", "yellow4",
                      "darkblue", "darkblue", "darkgreen"),
            z.min = 0, z.max = 12,
            sort.performance = FALSE, show.performance = FALSE,
            performance.decreasing = TRUE, performance.format = "%4.2f")
```

Arguments

data.set	Character string giving the SelectionTools marker data set.
ifilename	Character string giving a file with one individual identifier per row. The file is read from the input directory. Ignored when i.list is supplied.
i.list	Vector of individual identifiers to plot. The order is retained unless sort.performance = TRUE.
f.ind	Position of the first individual to plot. Without performance sorting this has the established meaning in the current data-set/list order. With sort.performance = TRUE, it refers to the order after sorting.
l.ind	Position of the last individual to plot. Without performance sorting this has the established meaning in the current data-set/list order. With sort.performance = TRUE, it refers to the order after sorting.
f.lr	Relative width of the left label area.
f.tb	Relative height of the top and bottom label areas.
d.h	Horizontal distance between chromosome panels.
d.v	Vertical distance between individual panels.
p.t	Logical. If TRUE, marker tick marks are plotted.
p.s	Logical. If TRUE, a map-position scale is plotted.
d.t	Length of marker tick marks.
d.map	Distance between map-position labels.

`c.nme` Character vector with chromosome labels. The default uses consecutive chromosome labels.

`i.nme` Character vector with individual labels. When supplied, it takes precedence over automatic performance labels. If performance sorting is requested, these labels are reordered together with their individuals.

`cex.chrom` Character expansion factor for chromosome labels.

`cex.ind` Character expansion factor for individual labels.

`color` Character vector of colors used for genotype or allele codes.

`z.min` Minimum genotype or allele code represented by the color scale.

`z.max` Maximum genotype or allele code represented by the color scale.

`sort.performance`
Logical. If `TRUE`, order the eligible individuals by the performance values stored in `data.set` before applying `f.ind` and `l.ind`.

`show.performance`
Logical. If `TRUE` and `i.nme` is not supplied, append the stored performance to each individual identifier.

`performance.decreasing`
Logical. Direction of performance sorting. `TRUE` places the largest performance first.

`performance.format`
A single `sprintf`-style format string used for the numeric performance value in automatically generated labels.

Details

The plot is drawn on the currently active R graphics device. File output is therefore controlled by the caller, for example with `pdf()`, `png()`, or another graphics device followed by `dev.off()`.

If neither `sort.performance` nor `show.performance` is requested, the established individual-selection and ordering behavior is unchanged. When `sort.performance = TRUE`, sorting is restricted to the individuals selected by `i.list` or `ifilename`; if neither is supplied, all individuals in the data set are eligible. The range `f.ind:l.ind` is then applied after sorting. With `sort.performance = FALSE`, `f.ind` and `l.ind` retain their established meaning in the current data-set/list order.

Performance values are obtained from the performance data stored in the specified marker data set. If performance sorting or automatic performance labels are requested but performance data are unavailable, the function stops with an error. A supplied `i.nme` always takes precedence over automatic performance labels.

For SNP data the default colors correspond to successive allele codes. After haplotype recoding, colors represent the resulting haplotype-allele codes. Missing genotypes are left unpainted. The function cannot be entered while the current device is already in `split.screen()` mode, because it uses split-screen panels internally. Graphics parameters changed by the function are restored on exit.

Value

No useful value is intended. The function is called for its side effect of drawing the graphical genotype plot on the active graphics device.

st.plot.ggt.src *Plot Graphical Genotypes from Map and Genotype Objects*

Description

Low-level plotting routine used by `st.plot.ggt`. It draws a graphical genotype representation from an already prepared linkage map and genotype matrix on the currently active R graphics device.

Usage

```
st.plot.ggt.src(map, pop, f.lr, f.tb, d.h, d.v, p.t, p.s, d.t, d.map,
                c.nme, i.nme, color, z.min, z.max, cex.chrom, cex.ind)
```

Arguments

`map`	Data frame or matrix describing chromosome, position, marker name, and marker class in the format used by `st.plot.ggt`.
`pop`	Genotype matrix; rows are markers and columns are individuals.
`f.lr`	Relative width of the left individual-label area.
`f.tb`	Relative height of the top and bottom label areas.
`d.h`	Horizontal distance between chromosome panels.
`d.v`	Vertical distance between individual panels.
`p.t`	Logical. If `TRUE`, marker tick marks are plotted.
`p.s`	Logical. If `TRUE`, a map-position scale is plotted.
`d.t`	Length of marker tick marks.
`d.map`	Distance between map-position labels.
`c.nme`	Character vector of chromosome labels.
`i.nme`	Character vector of individual labels.
`color`	Character vector of colors used for genotype or allele codes.
`z.min`	Minimum genotype or allele code represented by the color scale.
`z.max`	Maximum genotype or allele code represented by the color scale.
`cex.chrom`	Character expansion factor for chromosome labels.
`cex.ind`	Character expansion factor for individual labels.

Details

This is a low-level plotting function. It does not open or close a graphics device. The caller controls file output with the standard R graphics-device functions. The function uses `split.screen()` internally, refuses to run inside an already active split-screen layout, and restores graphics parameters when it exits. Missing genotype codes are represented as missing values and are not painted.

Value

No useful value is intended. The function is called for its side effect of drawing on the active graphics device.

st.read.map *st.read.map()*

Description

Reads a linkage map for an existing marker data set, matches loci by marker name, and restricts the marker matrix to markers represented in both sources.

Usage

```
st.read.map(filename, skip = 0, format = "cpms", m.stretch = 1,
            auxfiles = TRUE, data.set = "default")
```

Arguments

filename	Character string naming the map input file. Absolute paths are used directly; relative names are resolved using st.input.dir.
skip	Non-negative integer giving the number of initial input lines to ignore.
format	Character string specifying "cpms" or "mcp".
m.stretch	Finite numeric multiplier applied to every map position after reading. Use 1 when positions already use the desired map unit.
auxfiles	Logical. If TRUE, the matched and sorted map is written to an internal file in the R session temporary directory, read back by the R wrapper, and removed before the function returns.
data.set	Character string naming the marker data set to which the map is attached.

Details

A marker data set must already be loaded. The supported map layouts are "cpms", containing chromosome, position, marker name, and class, and "mcp", containing marker name, chromosome, and position. In "mcp" input the class is assigned internally. The first skip lines can be ignored before map parsing. Map positions are multiplied by m.stretch.

Marker matching is by marker name. Loci present in the map but absent from the marker matrix are silently ignored. Marker rows present in the marker matrix but absent from the map are silently removed. This behavior is intentional: after a successful read the marker data set represents the intersection of marker matrix and linkage map. The remaining genotype rows are reordered into linkage-map order and marker statistics are recalculated. Phenotypic data associated with individuals are preserved.

Duplicate marker names in either the linkage map or marker matrix are rejected because they make the mapping ambiguous. Equal positions on the same chromosome are separated deterministically while preserving their relative order and without moving a tied locus beyond the next genuinely distinct map position.

The operation is transactional. Parsing, matching, construction of the restricted genotype matrix, linkage-map structures, and marker statistics are completed before the existing marker data are replaced. On failure, the previous marker matrix, map, phenotype vector, and derived state remain

available. On successful replacement, model- and marker-order-dependent derived structures belonging to the old marker set are invalidated.

Value

Invisibly returns a `data.frame` with columns `Chrom`, `Pos`, `Name`, and `Class` when `auxfiles = TRUE` and map import succeeds; otherwise `NULL`. The main effect is updating the map associated with `data.set`.

`st.read.map.df` *Load a marker map from a data frame or matrix*

Description

Attaches a linkage map supplied in R to an existing marker data set, without writing temporary files.

Usage

```
st.read.map.df(map, m.stretch = 1, data.set = "default")
```

Arguments

`map`	A non-empty data frame or matrix with exactly three columns, interpreted in order as marker name, chromosome, and position (MCP). Column names are ignored. Marker names must be unique, nonempty, and contain no whitespace. Chromosomes must be positive integers and positions finite, non-negative numbers in the same units used by `st.read.map()`.
`m.stretch`	Positive finite multiplier applied to map positions.
`data.set`	Name of an existing marker data set.

Details

Only markers shared by the map and marker data are retained. Genotypes are reordered by chromosome and position together with their marker names. At least one marker must match. Equal positions are separated by small internal offsets using the same rules as `st.read.map()`.

Individual order, performance data, and the phase state are preserved. Marker statistics are recalculated; previous marker-dependent model results are cleared. Invalid input leaves the existing data set unchanged. This function attaches a marker-data map; it does not define the simulation genome.

Value

Returns `NULL` invisibly. The named marker data set is updated.

st.read.marker.data *st.read.marker.data()*

Description

Reads marker data into a named SelectionTools marker data set and derives the marker statistics used by subsequent analyses.

Usage

```
st.read.marker.data(filename, format, auxfiles = FALSE, data.set = "default")
```

Arguments

filename	Character string naming the marker input file. Absolute paths are used directly; relative names are resolved using st.input.dir.
format	Character string selecting the input layout: "l" for list format, "m" for marker-major matrix format, "t" for transposed semicolon-separated matrix format, or "n" for NTSys/Plabsim allele-incidence format.
auxfiles	Logical. If TRUE, marker, individual, and genotype summary files are created in the R session temporary directory, read back by the R wrapper, and removed before the function returns.
data.set	Character string naming the SelectionTools marker data set that is replaced after a successful read.

Details

Four input layouts are supported. Format "l" is a list or long format in which a record identifies an individual, a marker, and one allele. A homozygous genotype can therefore be represented by one allele record and a heterozygous genotype by two records for the same individual and marker. In this format a multi-digit token such as 12 is a single allele number 12.

Format "m" is a marker-major matrix. Individual names are read from the header and each subsequent row starts with a marker name followed by one genotype for each individual. Format "t" is the transposed, individual-major matrix and uses semicolon-separated fields. Format "n" is the NTSys/Plabsim allele-incidence representation in which rows have names of the form marker.allele followed by zero-one incidence values.

In matrix genotype fields, a single allele is interpreted as homozygous. A slash separates explicitly coded alleles, such as 12/14. Compact two-character genotypes are interpreted as two one-character alleles, so 12 denotes genotype 1/2. Compact nucleotide genotypes are also recognized; the internal nucleotide codes are A=1, C=2, G=3, T=4, and D=5. Numeric biological allele codes must be integers from 1 through 254; they do not need to be consecutive within a marker. The tokens -1, --, -, ., and NA denote missing data. The -1 spelling is retained for compatibility with historical SelectionTools marker files. If either allele of a diploid genotype is missing, the complete genotype is stored as missing.

The read is transactional. The new marker data, names, genotype matrix, and marker statistics are built and validated before replacing an existing data set. If the file cannot be read or is structurally

invalid, the previous marker data set is left unchanged. Duplicate marker or individual identities that would make a matrix ambiguous are rejected. NTSys marker blocks must be contiguous and incidence values must be zero or one.

After a successful read the marker statistics are recomputed from the imported matrix. When `auxfiles=TRUE`, the R wrapper reads the generated individual, marker, and genotype summaries back into R and removes the temporary output files.

Value

Invisibly returns a `list` when the requested auxiliary tables are produced, with components `individual.list`, `marker.list`, and `genotypes`, each a `data.frame`. Otherwise the return value is `NULL`; the main effect is loading marker data into `data.set`.

`st.read.marker.data.df`

Load marker data from an R data frame or matrix

Description

Loads diploid marker genotypes into a named SelectionTools data set.

Usage

```
st.read.marker.data.df(markerdata, format = NULL, data.set = "default")
```

Arguments

`markerdata`	For `format="list"`, a data frame whose first three columns are character vectors containing individual names, marker names, and allele codes. For `format="matrix"`, a numeric or character matrix or data frame with individuals in rows and markers in columns. Explicit row names identify individuals and column names identify markers. Both sets of names must be unique, nonempty, and contain no whitespace.
`format`	Either `"list"` or `"matrix"`. The default, `NULL`, selects list format.
`data.set`	Character string naming the data set to create or replace.

Details

In list format, each row supplies an allele record for one individual and marker. Alleles may be supplied as separate records or as pairs separated by a slash, for example `"1/2"`. Biological allele codes range from 1 to 254. Missing values and the tokens `"-1"`, `"--"`, `"-"`, `"."`, and `"NA"` represent missing alleles. If either allele is missing, the complete genotype is missing. Individual and marker names are sorted in list format.

In matrix format, rows represent individuals and columns represent markers. Row names identify individuals and column names identify markers.

Genotypes may be supplied as dosages: `0` denotes a homozygous major-allele genotype, `1` a heterozygote, and `2` a homozygous minor-allele genotype. These correspond to allele pairs `1/1`, `1/2`, and `2/2`, respectively.

Alternatively, genotypes may be supplied as character allele pairs: `"1/1"`, `"1/2"`, `"2/1"`, or `"2/2"`. Use a pipe to indicate known phase, for example `"1|2"`. The supplied allele order is preserved for both separators.

The data set is marked as phased only when all non-missing genotypes use pipe-separated allele pairs. Mixing these with slash-separated pairs or dosages marks the data set as unphased. Missing genotypes do not affect this classification; entirely missing data are unphased.

Allele codes 1 and 2 refer to the user-designated major and minor alleles; these labels are not inferred from observed frequencies. R missing values, `"--"`, and `"-"` represent missing genotypes. Partially missing pairs and other allele codes are not supported. Matrix format preserves the supplied individual and marker order. Invalid matrix input leaves an existing data set unchanged.

Value

Returns `NULL` invisibly. The function loads marker data into `data.set`.

`st.read.performance.data`
st.read.performance.data()

Description

Reads phenotypic values for a marker data set and retains only marker-data individuals for which a phenotype is available.

Usage

```
st.read.performance.data(filename, out.filename = "y.vector", auxfiles = TRUE,
                         data.set ="default")
```

Arguments

`filename`	Character string naming the phenotype input file. Relative names are resolved using `st.input.dir`.
`out.filename`	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
`auxfiles`	Logical. If `TRUE`, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
`data.set`	Character string naming the marker data set whose individuals are matched to the phenotype file.

Details

The input contains individual identifiers and phenotypic values. The marker matrix must already be loaded because individual names are matched against the marker-data individuals.

Phenotype records for individuals that are not present in the marker matrix are silently ignored. This is a normal situation and does not generate a warning. Conversely, individuals present in the marker matrix for which no phenotypic record is found are removed from the marker data set. Thus a successful call leaves the individual intersection of marker and phenotype data. Marker order and the linkage map are preserved, while marker statistics are recalculated for the retained individuals.

If a known individual occurs more than once in the phenotype input, the last record is used. A phenotype for a known individual must be a valid finite numeric value. Unknown individuals are ignored before their value field is interpreted. If no phenotype record matches any marker-data individual, the operation fails rather than creating an empty marker data set.

The operation is transactional. The reduced marker matrix, names, phenotype vector, map copy, and marker statistics are prepared before replacing the existing data set. A read or allocation failure therefore leaves the previous marker data set unchanged.

Value

Invisibly returns a `data.frame` with columns `i` (individual identifier) and `y` (performance value) when `auxfiles = TRUE` and the operation succeeds; otherwise `NULL`. The main effect is updating performance data in `data.set`.

`st.recode.hbc` *st.recode.hbc()*

Description

The function *st.recode.hbc* recodes the haplotype data for backcross populations according to a given reference individual.

Usage

```
st.recode.hbc(reference = 1, data.set = "default")
```

Arguments

`reference`	Specifies which individual is treated as reference for recoding. Usually the recurrent parent is set to be the reference individual. Must be between 1 and the number of individuals in the data set.
`data.set`	Name of data set that is used for recoding.

Details

If, at a given locus, an individual carries a haplotype that is also present in the reference genotype, then this haplotype is recoded to 1. If the haplotype is not present in the reference genotype, then it is recoded to 2.
Using the function *st.plot.ggt* after the function *st.recode.hbc* gives a good graphical overview, which haplo blocks are not inherited from the reference genotype (if reference individual = 1, the reference parent genome is plotted in yellow. The haplo blocks inherited from the donor parent are plotted in blue.

Value

On a successful run, an invisible implementation-level `list` returned by the final timing/information helper; the meaningful result is the side effect of recoding marker data according to the HBC reference convention. Validation failures may return `NULL`.

---

st.recode.hil *Recode Defined Haplotype Blocks as Marker Loci*

Description

Recodes marker sequences within previously defined haplotype blocks and uses the resulting haplotype variants as alleles of new haplotype-block loci.

Usage

```
st.recode.hil(out.filename = "blocks", auxfiles = TRUE, data.set = "default")
```

Arguments

`out.filename`	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
`auxfiles`	Logical. If `TRUE`, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
`data.set`	Name of data set that is used for recoding.

Details

Haplotyping is a two-step procedure. First, `st.def.hblocks` defines the marker boundaries of the haplotype blocks. Then `st.recode.hil` compares the stored marker sequence on each homologue within each block and identifies identical haplotype sequences.

Within a block, identical marker sequences receive the same haplotype allele code. Haplotype alleles are ordered by observed frequency, with the most frequent haplotype receiving the first code. A haplotype containing a missing marker allele is represented as missing after recoding. If a block would produce more than 254 distinct haplotype alleles, recoding stops with an error rather than merging or truncating haplotypes.

The operation modifies `data.set`. After successful recoding, the original marker-level representation is replaced by a new marker data set in which the defined haplotype blocks are the loci and the observed haplotype variants are their alleles. Make a copy of a marker data set before haplotyping if the original marker-level representation is also required.

The returned haplotype description identifies each block and haplotype allele and gives the sequence of original marker allele codes defining that haplotype. The external haplotype description uses `-1` for missing marker alleles.

Using `st.plot.ggt` after recoding provides a graphical overview of haplotype-block sharing among individuals.

Value

Invisibly returns a `data.frame` with columns `Block`, `AlleleNr`, and `AlleleDef` describing recoded haplotype-block alleles when `auxfiles = TRUE` and the operation succeeds; otherwise `NULL`.

See Also

`st.def.hblocks`, `st.calc.ld`, `st.plot.ggt`, `st.plot.gene.diversity`

---

st.recode.ref *Recode Marker Alleles Relative to a Reference Individual*

Description

Recodes every marker genotype according to its allele sharing with one reference individual.

Usage

```
st.recode.ref(reference = 0, descending = FALSE, data.set = "default")
```

Arguments

`reference`	Reference individual. A character value is interpreted as an individual identifier, whereas a positive whole-number numeric value is interpreted as a one-based individual position in `data.set`.
`descending`	Logical. Selects the allele-sharing rule used for recoding; see Details.
`data.set`	Character string giving the SelectionTools marker data set to recode.

Details

A character reference is looked up directly among the individual names in the specified data set. Thus `reference = "142"` means the individual named `"142"`, whereas `reference = 142` means the individual at position 142. The reference must identify exactly one existing individual; invalid names and out-of-range numeric positions are rejected.

With `descending = FALSE` (the default), a genotype with the same two alleles as the reference genotype is recoded to `1/1`; a genotype sharing one reference allele is recoded to `1/2`; and a genotype

sharing no reference allele is recoded to 2/2. The reference individual is therefore 1/1 wherever its source genotype is non-missing.

With descending = TRUE, a genotype is recoded to 1/1 when both of its alleles can originate from the reference genotype, to 1/2 when one allele can originate from the reference, and to 2/2 otherwise. Missing source genotypes remain missing.

Whenever a genotype is recoded to 1/2, the implementation stores the reference-sharing state as allele1 = 1 and allele2 = 2. Consequently, this recoded 1/2 state is a summary of allele sharing with the reference and must not be interpreted as inferred or preserved gametic phase.

Value

No useful value is intended. The function modifies the selected marker data set in place according to the reference-individual coding.

---

st.recode.ref.2 *Recode reference-marker information using the package-specific convention*

Description

Recode reference-marker information using the package-specific convention.

Usage

```
st.recode.ref.2(r1 = 0, r2 = 1, descending = FALSE,
                data.set = "default")
```

Arguments

r1	Function argument used by this operation.
r2	Function argument used by this operation.
descending	Function argument used by this operation.
data.set	Name of the SelectionTools data set.

Value

On a successful run, an invisible implementation-level list returned by the final timing/information helper; the meaningful result is the side effect of recoding reference-marker information for two parents. Validation failures may return NULL.

st.reset *Reset SelectionTools data and settings*

Description

Reset SelectionTools data and settings.

Usage

```
st.reset()
```

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of resetting SelectionTools marker-data state.

st.restrict.marker.data *st.restrict.marker.data()*

Description

Restricts an existing marker data set by individual names, marker names, missingness, allele number, and expected heterozygosity.

Usage

```
st.restrict.marker.data(ind.list = NULL, ind.file = NULL, mar.list = NULL,
                        mar.file = NULL, NoAll.MAX = 999, MaMis.MAX = 1,
                        ExHet.MIN = 0, InMis.MAX = 1, data.set = "default")
```

Arguments

`ind.list`	Optional character vector of individual names to retain. Unknown names are ignored. Do not supply together with `ind.file`.
`ind.file`	Optional file containing individual names to retain. Relative names are read from the input directory. Do not supply together with `ind.list`.
`mar.list`	Optional character vector of marker names to retain. Unknown names are ignored. Do not supply together with `mar.file`.
`mar.file`	Optional file containing marker names to retain. Do not supply together with `mar.list`.
`NoAll.MAX`	Maximum number of alleles allowed per marker.
`MaMis.MAX`	Maximum proportion of missing genotypes per marker.

ExHet.MIN	Minimum expected heterozygosity.
InMis.MAX	Maximum proportion of missing genotypes per individual.
data.set	Character string naming the marker data set to restrict.

Details

The function modifies data.set in place, retaining only individuals and markers that satisfy all applicable criteria. Use st.copy.marker.data beforehand if the original data set should be preserved.

Markers are retained when their number of non-missing alleles does not exceed NoAll.MAX, their missing-data proportion does not exceed MaMis.MAX, and their expected heterozygosity is at least ExHet.MIN. Individuals are retained when their missing-data proportion does not exceed InMis.MAX. Missingness thresholds are proportions, for example 0.1 for 10 percent.

All criteria are evaluated using the statistics available at the start of the call. Statistics are not recalculated between criteria within the same call. To filter in stages, use separate calls. For example, remove individuals with high missingness first and then filter markers using the updated statistics.

The linkage map and associated performance data are restricted along with the marker data. Removing markers invalidates marker-dependent model results, which must be recalculated before further use.

Value

On a successful run, an invisible implementation-level list returned by the final timing/information helper. The meaningful result is the side effect of restricting the marker data in the selected data set. Validation failures may return NULL.

Note

Restrictions are based on the marker statistics present at the start of each call. Repeating the function after statistics have been recalculated can therefore give a different result from applying several criteria simultaneously.

Examples

```
data("v-tropmaize-vcf")
data("v-tropmaize-phe")

st.load.vcf.data(v.tropmaize.vcf)
st.load.performance.data(v.tropmaize.phe)

st.restrict.marker.data( NoAll.MAX = 2    )
st.restrict.marker.data( MaMis.MAX = 0.05 )
st.restrict.marker.data( ExHet.MIN = 0.1  )
st.restrict.marker.data( InMis.MAX = 0.1  )
```

st.return.performance.data *Return performance data from a SelectionTools data set*

Description

Return performance data from a SelectionTools data set.

Usage

```
st.return.performance.data(out.filename = "y.vector", auxfiles = TRUE,
                           data.set = "default")
```

Arguments

out.filename	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
auxfiles	Logical. If TRUE, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
data.set	Name of the SelectionTools data set.

Value

Invisibly returns a data.frame with character column i (individual identifier) and numeric column y (performance value) when the data can be returned; otherwise NULL. Numeric-looking individual identifiers are retained as character strings.

st.select.phen *Select individuals using phenotypic information*

Description

Select individuals using phenotypic information supplied as a data frame or stored in a SelectionTools data set.

Usage

```
st.select.phen(p = NULL, n = 0, t = 0, decreasing = TRUE, nsmall = 1,
               data.set = NULL)
```

Arguments

p	Optional data frame containing individual identifiers in a column named i or ii and numeric performance values in a column named y or yy. If p is NULL, performance data are obtained from data.set.
n	Maximum number of individuals to return. A value of 0 returns all individuals remaining after threshold selection.
t	Optional performance threshold. If greater than 0, values greater than t are retained when decreasing=TRUE; values smaller than t are retained when decreasing=FALSE.
decreasing	Logical. If TRUE, sort performance from largest to smallest; otherwise sort from smallest to largest.
nsmall	Minimum number of digits after the decimal point used in the generated descr column.
data.set	Optional name of a SelectionTools data set. Used when p=NULL; the performance values stored in that data set are selected.

Value

A data.frame containing the selected individuals, sorted and filtered as requested, with an added character column descr combining the individual identifier and performance value. The original identifier/performance column names and column types are retained when p is supplied. When performance data are obtained from data.set, individual identifiers are character strings.

---

st.set.hblocks *Set haplotype-block information*

Description

Set haplotype-block information.

Usage

```
st.set.hblocks(haplotype.list, hap.symbol = "s", out.filename = "blocks",
               auxfiles = TRUE, data.set = "default")
```

Arguments

haplotype.list	Function argument used by this operation.
hap.symbol	Function argument used by this operation.
out.filename	Character string retained for compatibility and used as part of the name of an internal file in the R session temporary directory. The temporary file is removed before the function returns.
auxfiles	Logical. If TRUE, the compiled result is written to the R session temporary directory, read back by the R wrapper, and removed before the function returns.
data.set	Name of the SelectionTools data set.

Value

Invisibly returns a `data.frame` with columns `Chrom`, `Pos`, `Name`, `Class`, and `Markers` describing the installed haplotype blocks when `auxfiles = TRUE` and the operation succeeds; otherwise `NULL`.

---

st.set.info.level *Set info level used by SelectionTools*

Description

Set info level used by SelectionTools.

Usage

```
st.set.info.level(level)
```

Arguments

`level`	Function argument used by this operation.

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of setting the SelectionTools information level.

---

st.set.matrix.ops *Set matrix ops used by SelectionTools*

Description

Set matrix ops used by SelectionTools.

Usage

```
st.set.matrix.ops(select.s="mt", select.p="st")
```

Arguments

`select.s`	Function argument used by this operation.
`select.p`	Function argument used by this operation.

Value

`NULL`, returned invisibly. The function is called for its side effect of configuring matrix operations.

st.set.num.threads *Set num threads used by SelectionTools*

Description

Set num threads used by SelectionTools.

Usage

```
st.set.num.threads(active)
```

Arguments

active	Function argument used by this operation.

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of setting the number of computational threads.

st.set.openblas.threads *Set openblas threads used by SelectionTools*

Description

Set openblas threads used by SelectionTools.

Usage

```
st.set.openblas.threads(active)
```

Arguments

active	Function argument used by this operation.

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of setting the OpenBLAS thread count.

`st.set.sim.ef` *Transfer Marker Effects to the Simulation Backend*

Description

Transfers estimated marker effects from a SelectionTools marker data set directly to the active simulation backend.

Usage

```
st.set.sim.ef(name="effect", data.set="default")
```

Arguments

`name`	Character string naming the effect in the simulation backend.
`data.set`	Character string naming the SelectionTools marker data set that provides the estimated marker effects.

Details

The function transfers marker effects directly between the genomic-selection and simulation data structures; no intermediate effect file is written.

With the default allele codes used by the genomic-selection routines, `gs.set.allele.codes()` assigns the additive dosage codes 0, 1, and 2 to the genotypes with zero, one, and two copies of the effect allele, respectively. For marker j with estimated effect u_j, the marker contribution in the genomic-selection representation is therefore 0, u_j, or $2u_j$. The default codes for genotypes containing a missing allele are 0.5, 1, and 1.5.

The simulation backend stores an additive effect per allele. For each transferred marker, the effect allele receives $+u_j/2$ and the complementary allele receives $-u_j/2$. Before adjustment of the general mean, the three diploid genotype contributions are consequently $-u_j$, 0, and $+u_j$. This representation is equivalent to changing the marker dosage from z_{ij} to

$$z^*_{ij} = z_{ij} - 1.$$

For the genomic-selection representation, the estimated genetic value is

$$\hat{g}_i = \hat{\mu}_{012} + \sum_{j=1}^{m} z_{ij}\hat{u}_j.$$

Using $z_{ij} = z^*_{ij} + 1$ gives

$$\hat{g}_i = \left(\hat{\mu}_{012} + \sum_{j=1}^{m} \hat{u}_j\right) + \sum_{j=1}^{m} z^*_{ij}\hat{u}_j.$$

Thus, the general mean stored for the simulation effect is

$$\hat{\mu}_{\mathrm{sim}} = \hat{\mu}_{012} + \sum_{j=1}^{m} \hat{u}_j .$$

This is the coding-shift correction described by Stranden and Christensen (2011), Equation (2). In their general notation the change in the estimated general mean is the coding-shift vector transposed times the marker-effect vector. For the shift from 0/1/2 to -1/0/1, the coding-shift vector consists of ones and the correction is therefore the sum of marker effects.

This shift is distinct from allele-frequency centering. If a marker column is centered as $z_{ij}^{c} = z_{ij} - 2p_j$, the corresponding change of the general mean is $2\sum_j p_j u_j$. The simulation transfer uses $z_{ij}^{*} = z_{ij} - 1$, not allele-frequency centering. The two shifts are the same for a marker only when $p_j = 0.5$.

Only markers that are present in the active simulation map are transferred. Consequently, the sum in the intercept correction is taken over transferred markers only. Markers that are absent from the simulation map are ignored with a warning.

The exact equivalence described above applies to the default allele codes. `gs.set.allele.codes()` permits power users to define other codes; an arbitrary custom coding is not in general represented by the fixed $-u_j/2$, $+u_j/2$ simulation allele effects.

Marker effects are optional. If `data.set` contains no estimated marker effects, the function returns successfully without changing the simulation effects. A partially inconsistent marker-effect state is treated as an error.

When marker effects are present, a diploid simulation genome and a linkage map must already be installed. Marker effects are connected to simulation loci by marker name. The data set supplying the effects therefore need not be the data set that originally supplied the simulation map.

If an effect with the requested `name` is already loaded, the operation fails. After a new effect is installed successfully, stored population genetic and phenotypic values are removed because their effect dimension is no longer current.

Value

An invisible `list` returned by the underlying `.C` routine. Component `r` is the status value used by the simulation bridge; the function is primarily called for its side effect of transferring the requested data to the simulation backend.

References

Stranden I, Christensen OF (2011) Allele coding in genomic evaluation. Genetics Selection Evolution 43:25. See Equation (2).

See Also

`gs.set.allele.codes`, `st.set.sim.gp`, `st.set.sim.mp`, `st.set.sim.pp`, `st.set.simpop`

`st.set.sim.gp` *Set sim gp used by SelectionTools*

Description

Derives simulation genome parameters from a SelectionTools marker-data map and installs them in the simulation backend.

Usage

```
st.set.sim.gp(data.set="default")
```

Arguments

`data.set`	Character string naming the SelectionTools marker data set that provides the linkage map.

Details

The marker data set must contain a valid linkage map. Chromosome identifiers must define a contiguous sequence from chromosome 1 through the maximum chromosome number, and positions must be finite and non-negative. Chromosome lengths are derived from the mapped positions.

The SelectionTools marker matrix is diploid, so the bridge installs genome parameters compatible with two homologues. This is the genome-parameter stage of transferring a marker data set into the simulation representation. It does not by itself install map points or create a population.

Value

An invisible `list` returned by the underlying `.C` routine. Component `r` is the status value used by the simulation bridge; the function is primarily called for its side effect of transferring the requested data to the simulation backend.

`st.set.sim.mp` *Set sim mp used by SelectionTools*

Description

Copies the mapped loci of a SelectionTools marker data set into the active simulation linkage map.

Usage

```
st.set.sim.mp(data.set="default")
```

Arguments

`data.set`	Character string naming the SelectionTools marker data set that provides the map points.

Details

Map points are transferred with chromosome, position, marker name, and class information. The transfer respects the SelectionTools distinction between physical map storage and map access order and preserves the source access relationship used by the marker matrix.

Simulation genome parameters should already be compatible with the marker-data map. This stage installs map points but does not create the simulation population.

Value

An invisible `list` returned by the underlying `.C` routine. Component `r` is the status value used by the simulation bridge; the function is primarily called for its side effect of transferring the requested data to the simulation backend.

`st.set.sim.pp` *Transfer a Marker Population to the Simulation Backend*

Description

Creates a simulation population from the diploid marker matrix of a SelectionTools data set.

Usage

```
st.set.sim.pp(pop.name="default", data.set="default")
```

Arguments

`pop.name`	Character string naming the simulation population to create. The name must contain 2 to 255 ASCII letters or digits.
`data.set`	Character string naming the SelectionTools marker data set supplying individuals and alleles.

Details

A diploid simulation genome and linkage map must already be installed. The source marker data set does not need to contain the linkage map that was used to initialize the simulation. Instead, the active simulation loci are matched to rows of the source marker matrix by marker name.

Every marker in the active simulation linkage map must occur in the source marker matrix. The source data set may contain additional markers; these are ignored during population transfer. Consequently, populations from several compatible SelectionTools data sets can be added to one initialized simulation without reloading genome parameters or the linkage map.

For each imported individual, allele values are placed at the corresponding simulation loci. SelectionTools biological allele codes 1 through 254 are transferred unchanged, while the internal SelectionTools missing representation is converted to the simulation missing-allele indicator. After construction, the imported population is checked marker by marker against the source marker matrix. If verification fails, the newly created population is removed.

Compatibility of biological coding across independently supplied data sets is the responsibility of the user. In particular, matching marker names are assumed to identify the same loci and allele coding. The requested `pop.name` must not already exist in the simulation backend.

Value

An invisible `list` returned by the underlying `.C` routine. Component `r` is the status value used by the simulation bridge; the function is primarily called for its side effect of transferring the requested data to the simulation backend.

See Also

`st.set.sim.gp`, `st.set.sim.mp`, `st.set.sim.ef`, `st.set.simpop`

---

st.set.simpop *Initialize the Simulation Backend from a SelectionTools Data Set*

Description

Initializes the simulation backend from a SelectionTools marker data set and creates a diploid simulation population.

Usage

```
st.set.simpop(pop.name, data.set="default", effect.name="effect")
```

Arguments

`pop.name`	Character string naming the simulation population to create. The name must contain 2 to 255 ASCII letters or digits.
`data.set`	Character string naming the source SelectionTools marker data set.
`effect.name`	Character string naming transferred marker effects in the simulation backend. The default is `"effect"`.

Details

The simulation routines and the marker-data/genomic-prediction routines use different internal representations. Marker data sets store genotypes in marker matrices, whereas the simulation backend reconstructs individual chromosomes and uses a count-location representation of chromosome segments for the simulation of meiosis. This conversion is therefore required before a SelectionTools marker data set can be used by the simulation routines.

The function first calls `reset.all`. Consequently, previously defined simulation populations, effects, and map information are discarded before the new simulation state is initialized. It then performs the bridge stages in sequence: `st.set.sim.gp`, `st.set.sim.mp`, `st.set.sim.ef`, and `st.set.sim.pp`. Processing stops immediately if a stage reports failure.

The source data set must provide marker data and a valid linkage map for the genome-parameter and map stages. The chromosome parameters, linkage map, and marker genotypes are transferred to the simulation backend and used to reconstruct the simulation chromosomes.

Estimated marker effects are optional. If no estimated marker effects are present, `st.set.sim.ef` succeeds without installing an effect and population transfer continues normally. If effects are present, they are transferred directly under `effect.name`; no intermediate effect file is created. The genomic-selection routines normally use additive marker dosage coding 0/1/2, whereas the simulation representation is equivalent to -1/0/1. The required baseline correction of the general mean is therefore applied during effect transfer. See `st.set.sim.ef` for the exact coding and formula.

The individual bridge functions can also be called separately. In particular, a simulation may be initialized once with genome parameters, map, and optional effects and then populated from additional compatible marker data sets using repeated calls to `st.set.sim.pp`. This staged route avoids the `reset.all` performed by another call to `st.set.simpop`.

Value

Invisibly returns the `list` produced by the last completed simulation-bridge stage. Component `r` is the bridge status value. The main effect is initialization of the simulation backend from the SelectionTools data set.

References

Maurer HP, Melchinger AE, Frisch M (2008) Population genetic simulation and data analysis with Plabsoft. Euphytica 161:133–139.

Stranden I, Christensen OF (2011) Allele coding in genomic evaluation. Genetics Selection Evolution 43:25. See Equation (2).

See Also

`st.set.sim.gp`, `st.set.sim.mp`, `st.set.sim.ef`, `st.set.sim.pp`, `reset.all`

`st.start.timer` *Start or stop the SelectionTools timing helper*

Description

Start or stop the SelectionTools timing helper.

Usage

```
st.start.timer(depth=1)
```

Arguments

`depth`	Function argument used by this operation.

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of starting the SelectionTools timing helper.

---

st.stop.timer *Start or stop the SelectionTools timing helper*

Description

Start or stop the SelectionTools timing helper.

Usage

```
st.stop.timer(depth=1, info.level=0)
```

Arguments

`depth`	Function argument used by this operation.
`info.level`	Function argument used by this operation.

Value

An invisible implementation-level `list` returned by the final information-message call after elapsed time is calculated and reported.

---

st.STvcf.to.dataframe *Convert Compact STvcf Data to a VCF-Like Data Frame*

Description

Expands a compact STvcf list to a wide VCF-like R data frame with one marker per row and one genotype column per individual.

Usage

```
st.STvcf.to.dataframe(STvcf)
```

Arguments

`STvcf`	A compact STvcf list containing marker metadata, individual identifiers, chromosome mapping, and two allele matrices.

Details

The function validates the STvcf object before expansion. The list must contain exactly the nine documented STvcf components, each exactly once; extra, missing, duplicated, unnamed, or unknown components are rejected. Component types and dimensions must agree, chromosome codes must be consecutive and used, map positions must be finite and nonnegative, allele indices must agree with REF/ALT, and marker and individual identifiers must obey the same restrictions as `st.load.vcf.data()`.

Marker and individual identifiers must be unique, shorter than 256 bytes, and contain only ASCII letters and digits. Dots and all other punctuation are rejected. These checks are intentionally the same on conversion in both directions.

REF may not be missing, `"."`, empty, or comma-separated. ALT may be `"."` or a comma-separated list. ALT entries must be non-empty, unique, and different from REF. Invalid or duplicated allele labels are rejected.

The returned data frame contains `CHROM`, `POS`, `ID`, `REF`, `ALT`, and `FORMAT`, followed by one sample column per entry in `STvcf$individual`. FORMAT is written as `"GT"` for every marker. Including FORMAT is deliberate: it makes the sample boundary unambiguous even when an individual identifier is itself an alphanumeric name such as POS, INFO, CM, or FORMAT.

CHROM is written using `CHROM.levels`. POS is written as the current STvcf double-precision genetic position in cM without intentional rounding.

Genotypes use VCF allele indices, with 0 for REF and 1, 2, ... for ALT alleles. Allele order is retained. If the STvcf object has logical attribute `phased=TRUE`, non-missing genotypes are written with `"|"` and missing genotypes as `".|."`. Otherwise `"/"` and `"./."` are used. STvcf objects created before this attribute was introduced, or manually created objects without it, are treated as unphased.

The two allele matrices must use the same storage type. Raw matrices use raw value 255 for missing data; integer matrices use `NA_integer_` for missing data. STvcf objects accepted by SelectionTools may define at most 253 ALT alleles per marker, so valid non-missing VCF allele indices range from 0 through 253. Allele indices are checked against the corresponding REF/ALT definition.

STvcf does not retain QUAL, FILTER, INFO, non-GT FORMAT fields, phase-set metadata, or a physical POS field that was superseded by CM. These fields cannot be reconstructed. For an STvcf object produced by `st.dataframe.to.STvcf()`, conversion to a data frame and back preserves the data-set level phased/unphased state through the logical `phased` attribute.

Value

A VCF-like `data.frame` with fixed columns `CHROM`, `POS`, `ID`, `REF`, `ALT`, and `FORMAT`, followed by one genotype column for each individual.

---

st.switch.error *Calculate Switch Error against Known Phase*

Description

Compares a phased SelectionTools marker data set with a data set containing known true phase and calculates the local switch-error rate chromosome by chromosome.

Usage

```
st.switch.error(true.data.set, data.set = "default")

## S3 method for class 'st.switch.error'
print(x, complete = FALSE, digits = 3, ...)
```

Arguments

true.data.set	Character string naming the SelectionTools marker data set containing the true ordered homologues.
data.set	Character string naming the phased data set to evaluate.
x	An object returned by st.switch.error().
complete	Logical. If FALSE (the default), print only switch.error, coverage, and genotype.mismatches. If TRUE, print the complete result.
digits	Number of significant digits used for printing. The default is 3. This affects display only; stored values retain their full precision.
...	Additional arguments passed to print.data.frame.

Details

Individuals and markers are matched by their unique names, so the two data sets need not use the same storage order. They must contain the same numbers and names of individuals and markers and complete linkage maps.

At each observed heterozygote in the true data, the candidate phase is recorded as having the same or opposite orientation relative to the truth. Two consecutive scorable heterozygotes on the same chromosome define one switch link. A switch error is counted when their relative orientations differ. The orientation state is reset at chromosome boundaries, so an arbitrary exchange of the two haplotype labels for an entire chromosome has no effect on the result. Missing candidate genotypes and incompatible candidate genotypes are not used to form links and are reported separately.

The current implementation is for biallelic markers, matching the current st.phase() implementation. Allele codes are reduced to marker-specific ranks before comparison, which permits direct comparison with data exported by write.vcf(), phased by Beagle, and re-imported by read.vcf().

The switch-error rate is the number of switch errors divided by the number of scored links. This is the standard local switch-error definition used in statistical-phasing evaluations; see Browning and Browning (2007) and Browning and Browning (2022).

Value

An object of class "st.switch.error" inheriting from "data.frame", with one column, result. The row names are errors, links, switch.error, switch.error.percent, heterozygotes, scored.heterozygotes, coverage, missing.heterozygotes, and genotype.mismatches. switch.error and coverage are proportions; the former is NA when no switch link is available and the latter is NA when the true data contain no observed heterozygotes. The full result is always returned. Its print method displays only switch.error, coverage, and genotype.mismatches by default; use complete=TRUE to print all rows.

References

Browning SR, Browning BL (2007) Rapid and accurate haplotype phasing and missing-data inference for whole-genome association studies by use of localized haplotype clustering. *American Journal of Human Genetics* 81:1084–1097. doi:10.1086/521987.

Browning BL, Browning SR (2022) Genotype error biases trio-based estimates of haplotype phase accuracy. *American Journal of Human Genetics* 109:1016–1025. doi:10.1016/j.ajhg.2022.04.019.

See Also

`st.destroy.phase`, `st.phase`, `read.vcf`, `write.vcf`

`st.write.map` *Write linkage-map information for a SelectionTools data set*

Description

Write linkage-map information for a SelectionTools data set.

Usage

```
st.write.map(mfilename, auxfiles = TRUE, data.set = "default")
```

Arguments

`mfilename`	Required base name for the map file. The file is written with extension '`.mmp`'.
`auxfiles`	Logical. If `TRUE`, the written map file is read back and returned as a data frame. The map file itself is written regardless of this argument.
`data.set`	Name of the SelectionTools data set.

Value

Invisibly returns a `data.frame` with columns `Chrom`, `Pos`, `Name`, and `Class` when `auxfiles = TRUE` and map output succeeds; otherwise `NULL`. The map file is also written as requested.

st.write.marker.data *st.write.marker.data()*

Description

Writes marker data, and where applicable map and genome information, in one of the supported external representations.

Usage

```
st.write.marker.data(format = "m", lfilename = NULL, mfilename = NULL,
                     nfilename = NULL, ifilename = "", f.ind = 0,
                     l.ind = 0, add.inum = ".1", auxfiles = FALSE,
                     data.set = "default")
```

Arguments

format	Character string selecting "l", "m", or "n". The C writer also supports "a" as append-list mode.
lfilename	Base name for list-format output. Required when format = "l" or format = "a".
mfilename	Base name for matrix-format marker and map output. Required when format = "m".
nfilename	Base name for NTSys/Plabsim marker, map, and genome-parameter output. Required when format = "n".
ifilename	Optional file containing individual names to include in the output.
f.ind	First individual position to write when selecting by range. Positions are interpreted in the current data-set order.
l.ind	Last individual position to write when selecting by range.
add.inum	Character string appended to individual names in NTSys/Plabsim output.
auxfiles	Logical flag retained by the interface. The marker writer itself writes the files implied by format.
data.set	Character string naming the marker data set to write.

Details

Format "l" writes list records consisting of individual, marker, and allele. A heterozygous genotype is written as two allele records. The output file uses suffix '.lpo'. The internal append variant "a" appends list records to the list output file without writing a new header.

Format "m" writes a marker-major genotype matrix to a file with suffix '.mpo'. When a linkage map is available, a corresponding map file with suffix '.mmp' is also written. Genotypes are written with an explicit slash between the two allele codes, which preserves multi-digit allele numbers unambiguously.

Format "n" writes the Plabsim/NTSys group of files: allele-incidence marker data in '.npo', map data in Morgan units in '.nmp', and genome parameters (chromosome lengths in Morgan) in '.ngp'. Individual names are adapted to the restrictions of this representation and add.inum is appended to the written name. Marker names containing a period are not suitable for this output because the period separates marker name and allele number. Name transformations that would make two output individuals indistinguishable are rejected.

The writer does not modify the in-memory marker data set. In particular, writing a data set without a linkage map uses the marker matrix directly and does not create a synthetic map or alter map dimensions. Unsupported format codes are rejected.

Individuals can be restricted either by a positional range or by a file of individual names. These output selections affect only the written files and do not alter the stored marker data.

Value

NULL, returned invisibly. The function is called for its side effect of writing marker-data files in the selected output format.

---

st_mixed *Fit a mixed linear model*

Description

Fit a mixed linear model for genomic prediction.

Usage

```
st_mixed(y, id = NULL, Z, Z_ids = NULL, pool1 = NULL, pool2 = NULL,
         hybrid_id = NULL, include_sca = FALSE, ridge = 1e-8,
         method = c("std", "vanraden", "sommer"),
         coding = c("detect", "0 0.5 1", "-1 0 1", "0 1 2"), varcmp = NULL,
         maxcyc = 100L, tol = 1e-10, calcVC = 1,
         model_type = c("kernel", "marker"),
         bootstrap = list(nObs=NULL, nMarkers=NULL, nRep=1, seed=NULL))
```

Arguments

y	Response vector.
id	Identifier vector.
Z	Marker or design matrix.
Z_ids	Identifiers corresponding to rows of the marker matrix.
pool1	Function argument used by this operation.
pool2	Function argument used by this operation.
hybrid_id	Function argument used by this operation.
include_sca	Logical flag controlling inclusion of specific combining ability.

ridge	Ridge or numerical-stabilization parameter.
method	Method to use.
coding	Marker coding convention.
varcmp	Variance-component values.
maxcyc	Maximum number of fitting cycles.
tol	Convergence tolerance.
calcVC	Function argument used by this operation.
model_type	Function argument used by this operation.
bootstrap	Function argument used by this operation.

Value

A named list or NULL The component set depends on the selected mode.

For the population mode the list contains beta0 (intercept), varcmp (variance components), ghat (predicted genetic values), yhat (predicted response values), missing_ids (identifiers not matched to marker rows), diag (diagnostic counts and marker-coding information), and retval (compiled status code).

For hybrid modes the list contains beta0, varcmp, yhat_obs (predictions aligned to the observed-response vector), diag, retval, and ridge.

---

summarize.gvalue *summarize.gvalue()*

Description

Summarizes the genetic values of the individuals of a population with respect to defined effects

Usage

```
summarize.gvalue(pops, effectfile=NULL)
```

Arguments

pops	Names separated by blanks of the populations to be summarized.)
effectfile	Name of the effect

Details

If no effects are specified the weighted sum of all effects are used, otherwise only the single specified effects is used without using weights.

Value

A numeric matrix with one row per population and columns Obs, Mean, SDev, Min, Q10, Med, and Max, summarizing genetic values.

Note

See the `lib00.example1` for an application of the command.

`swap.population.name` *Swap population names*

Description

Swap population names.

Usage

```
swap.population.name(NameP1, NameP2)
```

Arguments

`NameP1`	Name of the first parental population.
`NameP2`	Name of the second parental population.

Value

Returns the same invisible implementation-level `list` as `population.swap.name()`. The function is a compatibility wrapper and the meaningful result is the delegated population operation.

`talk.to.me` *Enable or control SelectionTools console output*

Description

Enable or control SelectionTools console output.

Usage

```
talk.to.me()
```

Value

Returns the same invisible implementation-level `list` as `set.info.level(1)`; the main effect is enabling package information output.

v-tropmaize-map *Genetic Map for Tropical Maize*

Description

Marker names, chromosome assignments, and map positions for tropical maize.

Usage

```
data("v-tropmaize-map")
```

Format

A data frame with 7140 rows and 3 variables:

`name` Character marker name.

`chrom` Integer chromosome number.

`pos` Numeric marker position in the units used in the original map.

Source

Jose Crossa, Gustavo de los Campos, Paulino Perez, Daniel Gianola, Juan Burgueno, Jose Luis Araus, Dan Makumbi, Ravi P Singh, Susanne Dreisigacker, Jianbing Yan, Vivi Arief, Marianne Banziger, Hans-Joachim Braun, Prediction of Genetic Values of Quantitative Traits in Plant Breeding Using Pedigree and Molecular Markers, Genetics, Volume 186, Issue 2, 1 October 2010, Pages 713-724, doi:10.1534/genetics.110.118521

v-tropmaize-phe *Phenotypic Data for Tropical Maize*

Description

Phenotypic observations for 264 tropical maize individuals.

Usage

```
data("v-tropmaize-phe")
```

Format

A data frame with 264 rows and 2 variables:

`ii` Integer individual identifier.

`yy` Numeric phenotypic value.

Source

Jose Crossa, Gustavo de los Campos, Paulino Perez, Daniel Gianola, Juan Burgueno, Jose Luis Araus, Dan Makumbi, Ravi P Singh, Susanne Dreisigacker, Jianbing Yan, Vivi Arief, Marianne Banziger, Hans-Joachim Braun, Prediction of Genetic Values of Quantitative Traits in Plant Breeding Using Pedigree and Molecular Markers, Genetics, Volume 186, Issue 2, 1 October 2010, Pages 713-724, doi:10.1534/genetics.110.118521

---

v-tropmaize-pop *Genotype Data for Tropical Maize*

Description

Compact, lossless representation of genotype data for 1135 markers and 264 tropical maize individuals.

Usage

```
data("v-tropmaize-pop")
```

Format

A list with 4 components:

`marker` Character vector of 1135 marker names.

`individual` Character vector of 264 individual identifiers.

`genotype.levels` Character vector giving the genotype text represented by raw codes 0, 1, and 2, respectively: `"-1/-1"`, `"1/1"`, and `"2/2"`.

`genotype` A 1135 by 264 raw matrix. Each element is a raw code in `as.raw(0:2)` and indexes `genotype.levels` after conversion to integer and addition of one. Rows correspond to markers and columns correspond to individuals.

Source

Jose Crossa, Gustavo de los Campos, Paulino Perez, Daniel Gianola, Juan Burgueno, Jose Luis Araus, Dan Makumbi, Ravi P Singh, Susanne Dreisigacker, Jianbing Yan, Vivi Arief, Marianne Banziger, Hans-Joachim Braun, Prediction of Genetic Values of Quantitative Traits in Plant Breeding Using Pedigree and Molecular Markers, Genetics, Volume 186, Issue 2, 1 October 2010, Pages 713-724, doi:10.1534/genetics.110.118521

`v-tropmaize-vcf` *Compact STvcf Genotype and Genetic-Map Data for Tropical Maize*

Description

A compact STvcf representation of genotypes and genetic linkage positions for 264 tropical maize inbred lines. It contains markers occurring in both the tropical-maize marker matrix and the available genetic map.

Usage

```
data("v-tropmaize-vcf")
```

Format

A list of class `"STvcf"` with 1076 marker rows and 264 individuals:

`CHROM` Integer running chromosome codes 1 through 10.

`POS` Double-precision genetic positions in cM retained from the available linkage map without intentional rounding.

`ID` Character vector containing 1076 unique marker IDs.

`REF` Character vector containing `"1"` for every marker.

`ALT` Character vector containing `"2"` for every marker.

`individual` Character vector `"1"`, ..., `"264"`.

`allele1` A 1076 by 264 raw matrix. Raw 0 denotes REF, raw 1 denotes ALT, and raw 255 denotes missing.

`allele2` A 1076 by 264 raw matrix with the same coding.

`CHROM.levels` Character vector `"1"`, ..., `"10"`.

Details

The original tropical-maize genotype object contains 1135 genotyped markers and the available linkage map contains 7140 entries. This STvcf object contains their 1076-marker intersection. Genotype-only markers lacking a map position and map-only loci lacking genotype data are not included.

POS contains the original double-precision cM positions from the distributed linkage map. The positions are not rounded to integer cM. Exact ties are permitted in STvcf; when installed with `st.load.vcf.data()`, ties are separated internally by small deterministic offsets because the existing SelectionTools map and simulation structures require distinct ordered locus positions.

REF label `"1"` is VCF allele index 0 and ALT label `"2"` is VCF allele index 1. Loading maps these indices to internal SelectionTools alleles 1 and 2 and maps raw 255 to the internal SelectionTools missing representation.

The marker and individual identifiers obey the strict STvcf/SelectionTools name convention used by the new converters: only ASCII letters and digits are permitted. The tropical-maize marker IDs are alphanumeric and the individual IDs are the digit strings 1 through 264.

Marker statistics and individual missing-data frequencies after loading are calculated from the 1076 markers in this object. Applying NoAll.MAX=2, MaMis.MAX=0.1, ExHet.MIN=0.1, and InMis.MAX=0.1 leaves 828 markers and 230 individuals, the intended tropical-maize analysis data set.

Source

Jose Crossa, Gustavo de los Campos, Paulino Perez, Daniel Gianola, Juan Burgueno, Jose Luis Araus, Dan Makumbi, Ravi P Singh, Susanne Dreisigacker, Jianbing Yan, Vivi Arief, Marianne Banziger, Hans-Joachim Braun, Prediction of Genetic Values of Quantitative Traits in Plant Breeding Using Pedigree and Molecular Markers, Genetics, Volume 186, Issue 2, 1 October 2010, Pages 713-724, doi:10.1534/genetics.110.118521

---

write.vcf *Write SelectionTools Marker Data for Beagle*

Description

Writes a SelectionTools marker data set as a minimal GT-only VCF intended for direct use as Beagle target genotype input.

Usage

```
write.vcf(vcffile, data.set = "default")
```

Arguments

`vcffile`	Character string giving the output VCF file. If the name ends in `.gz`, the native writer first creates a temporary plain VCF and base R `gzfile()` compresses it to the requested file.
`data.set`	Character string naming an existing SelectionTools marker data set.

Details

The output is deliberately small and Beagle-oriented. It contains VCF 4.3 header information and one FORMAT field, `GT`. A marker data set whose stored phase state is known is written with the phased separator `|`; an unphased data set is written with `/`. Missing diploid genotypes are written consistently as `.|.` or `./.`, respectively. Thus VCF round trips retain the SelectionTools data-set phase state.

SelectionTools allele codes are normalized independently at each marker to VCF allele indices 0, 1, and so on. REF and ALT are valid synthetic DNA allele labels used only for interchange; they must not be interpreted as the original biological nucleotide labels. Marker IDs and individual names are retained.

SelectionTools genetic positions are in centiMorgans. To make a VCF usable by Beagle without a separate genetic-map file, positions are converted to pseudo-base-pair positions with one centiMorgan equal to one megabase, which is Beagle's documented no-map scale. Tied map positions are

made strictly increasing within a chromosome. The file records the scale in a `SelectionToolsPositionScale` metadata line.

No zlib, htslib, gzip executable, or other additional system library is linked by SelectionTools. Compression of `.vcf.gz` output uses only the public base-R `gzfile()` interface.

Value

No return value. The VCF file is written for its side effect.

See Also

`read.vcf`, `st.destroy.phase`, `st.phase`

`write.version.2` *Write version information in the package-specific format*

Description

Write version information in the package-specific format.

Usage

```
write.version.2()
```

Value

An invisible `list` returned by the underlying `.C` routine. The list is an implementation-level result containing the arguments returned by the compiled routine; the function is primarily called for its side effect of requesting/printing package version information.

Index

∗ **datasets**
DT_technow, 21
gd.dummy.populations, 38
gd.maize.aflp, 41
gd.maize.lines, 42
gd.maize.populations, 42
gd.salad.aflp, 44
lib00.map1, 116
lib00.map1a, 117
lib00.map2, 117
masbc.input, 139
Md_technow, 140
Mf_technow, 140
v-tropmaize-map, 271
v-tropmaize-phe, 271
v-tropmaize-pop, 272
v-tropmaize-vcf, 273
∗ **hplot**
st.LDheatmap, 211

append.population, 9

be.quiet, 10

concat.population, 10
copy.population, 11
cross, 12

data.params, 14
define.effects, 14
define.effects.df, 15
define.genome, 16
define.map, 17
deprecated, 19
dh, 19
divide.population, 20
DT_technow, 21

effect.weight.set.all, 21
effmap.remove, 22
effmap.remove.all, 22
evaluate.all.loci, 23
evaluate.allele, 23, *27*
evaluate.allele.freq, 24, *24*, *26*
evaluate.allele.freq2, *24*, 25, *27*, *161*
evaluate.allele2, *24*, 26
evaluate.genome, 27
evaluate.genotype, 28
evaluate.hap.calc, 30
evaluate.hap.calc.d, 30
evaluate.hap.free, 31
evaluate.hap.init, 31
evaluate.hap.return.genotype, 32
evaluate.hap.return.table, 32
evaluate.haplotype, 33
evaluate.ld, 33
evaluate.locus, 34
evaluate.mdp, 34
evaluate.population, 35

gd.allele.frequencies, 36
gd.allow.zero.frequencies, 36
gd.correct.missing, 37
gd.data.parameters, 37
gd.distance.similarity, 38
gd.dummy.populations, 38
gd.genetic.distance, 39, *42, 43*, *45*
gd.list.irregular, 40
gd.list.missing, 41
gd.maize.aflp, 41, *42, 43*, *45*
gd.maize.lines, 42, *42, 43*, *45*
gd.maize.populations, 42, *42*, *45*
gd.mk.matr, 43
gd.pcoa, 44
gd.salad.aflp, *42, 43*, 44
gd.similarity.coefficient, *42, 43*, 45, *45*
gd.splitdot, 46
gd.status.zero.frequencies, 47
generate.effect.file, 47
generate.map.file, 48
generate.population, 48

genome.contribution, 49
genome.parameter.get, 50
genome.parameter.set, 50
genome.segments, 51
genotype.population, 52
get.genome.par, 53
get.map, 53
get.mdp, 54
get.population, 54
get.population.gvalue, 55
get.population.info, 55
get.population.pvalue, 56
get.population.size, 57
get.score, 57
gs.build.esvs, 58
gs.build.tsps, 59
gs.build.V, 60
gs.build.Z, 60
gs.check.pvals, 61
gs.compare.effects, 61
gs.cross.eval.es, 62
gs.cross.eval.gd, 64
gs.cross.eval.gd.fct, 65
gs.cross.eval.ma, 65
gs.cross.eval.mi, 66
gs.cross.eval.mu, 67
gs.cross.eval.va, 68
gs.cross.info, 69
gs.cross.info.gd, 70
gs.cross.validation, 70
gs.esteff.lsq, 71
gs.esteff.rmla, 72
gs.esteff.rmlc, 73
gs.esteff.rmlr, 74
gs.esteff.rmlv, 74
gs.esteff.rr, 75
gs.estimate.gv, 77
gs.get.im, 77
gs.get.info.level, 78
gs.get.V, 78
gs.get.y, 79
gs.get.Z, 79
gs.info, 80
gs.lambda.aov, 80
gs.lambda.const, 81
gs.lambda.emstep, 81
gs.lambda.hsq, 82
gs.lambda.reg, 83
gs.lambda.rmla, 83
gs.lambda.rmla.02, 84
gs.lambda.rmlc, 85
gs.lambda.rmlr, 86
gs.lambda.rmlv, 86
gs.lambda.rrblup, 87
gs.mme.coeff, 88
gs.mme.invcoeff, 88
gs.mme.restcoeff, 89
gs.mme.restrhs, 89
gs.mme.rhs, 90
gs.mme.solve, 90
gs.mmet.coeff, 91
gs.mmet.coeff.3, 92
gs.mmet.rhs, 92
gs.mmet.solve, 93
gs.neg.effall, 93
gs.plot.effects, 94
gs.plot.model.fit, 95
gs.plot.validation, 96
gs.pos.effall, 96
gs.predict.genotypes, 97
gs.reset, 98
gs.restrict.marker.data.01, 98
gs.return.effects, 99
gs.return.pvals, 99
gs.set.all.info.levels, 100
gs.set.allele.codes, 100
gs.set.effects, 101
gs.set.info.level, 102
gs.set.lambda, 102
gs.set.num.threads, 103
gs.set.performance.data, 103
gs.single.marker.aov, 104
gs.single.marker.reg, 104
gs.start.timer, 105
gs.stop.timer, 105
gs.test.mp, 106
gs.testeff.lsq, 106
gs.testeff.rmlc, 107
gs.testeff.rmlv, 107
gs.vc.rrblup, 108
gs.w.aov, 109
gs.write.pseff, 109

homozygote, 111

il.eval.library, 111
il.eval.lines, 112

il.ideal.library, 112
il.overlapping.library, 113
info, 114
info.cat, 114
init.population, 115

lib00.map1, 116
lib00.map1a, 117
lib00.map2, 117
linkage.drag, 118
linkage.map.get, 118
linkage.map.load, 119
linkage.map.save, 119
list.effects, 120
list.populations, 121
load.effmap, 121
load.internal.effmap, 122
load.linkage.map, 123

mab.compare, 124, *131*, *137*, *139*
mab.df, 130
mab.Examples, *130*, 131, *132, 133*, *137*, *139*
mab.Input.files, *130*, 131, *131*, *137*, *139*
mab.load.data, *130, 131*, 132, *137*, *139*
mab.save.inds, 133
mab.simulate, *130–133*, 134, *139*
mab.tabulate, *130, 131*, *133*, *137*, 138, *139*
maize.aflp *(*gd.maize.aflp*)*, 41
masbc.input, 139
Md_technow, 140
Mf_technow, 140

optimize.population, 141

ph.linkage.map.create, 141
phenotype.population, 142
plabsim, 143
plabsim.init, 143
plabsim.R.date, 144
plabsim.R.version, 144
plabsim.version, 144
plant, 145
population.append, *9*, 145
population.concat, 146
population.copy, 147
population.divide, 147
population.exist, 148
population.individual.remove, 149
population.info.get, 149
population.info.set, 150
population.list, 150
population.matrix.load, 151
population.matrix.save, 151
population.name.swap, 152
population.optimize, 152
population.plabsim.load, 153
population.plabsim.save, 153
population.remove, 154
population.remove.all, 154
population.rename, 155
population.resize, 155
population.sample, 156, *170*
population.size.get, 157
population.sort, 157
population.swap.name, 158
population.transfer, 159
print.st.data.set.list
*(*st.data.set.list*)*, 194
print.st.switch.error
*(*st.switch.error*)*, 264

read.vcf, 159
remove.all.populations, 160
remove.effmaps, 161
remove.evaluate.population, 161
remove.genotype.population, 162
remove.map, 163
remove.population, 163
rename.population, 164
reset.all, 164
reset.mdp, 165
resize.population, 165
resources, 166
return.population, 166
rng.choose, 167
rng.info, 167
rng.init, 168
rng.list, 169

sample.population, 169
save.linkage.map, 170
sdev, 171
sel.target.define, 171
select.all.best, 172
select.all.best.intern, 173
select.genotypes, 174
select.n.best, 175
select.n.best.segments, 176

set.co.freq, 177
set.eff.weight, 177
set.genome.par, *18*, 178
set.info.level, 179
set.mdp, 179
set.NoLociInit, 180
set.population.gvalue, 180
set.population.info, 181
show.phase, 182
single.cross, 182
sm.start.timer, 183
sm.stop.timer, 183
splitdt, 184
ssd.mating, 184
st.calc.ld, 185
st.calc.ld.2, 187
st.calc.q, 188
st.calc.rf, 189
st.calculate.enrichment, 189
st.chrom.stats, 191
st.compact.data.set, 192
st.copy.marker.data, 193, *252*
st.data.set.list, 194
st.data.set.remove, 195
st.data.set.remove.all, 196
st.datadir, 197
st.dataframe.to.STvcf, 197, *218*, *219*
st.dd, 199
st.def.hblocks, 200
st.destroy.phase, 201
st.genetic.distances, 202
st.genetic.distances.02, 204
st.genetic.distances.fct, 205
st.get.info.level, 206
st.get.map, 206, *219*
st.get.num.threads, 207
st.get.simpop, 207
st.get.simpop.perfdata, 208
st.get.simpop2, 209
st.id, 209
st.indir, 210
st.info, 210
st.keep.intersection.markers, 211
st.LDheatmap, 211
st.LDplot.ld, 215
st.LDplot.map, 216
st.load.performance.data, 216
st.load.vcf.data, 217
st.mark.alleles, 220
st.marker.data.statistics, 220
st.markerdata.to.STvcf, *219*, 222
st.mxd, 224
st.od, 225
st.outdir, 225
st.phase, 226
st.phase.v1, 229
st.plot.corr, 236
st.plot.corr.l, 237
st.plot.gene.diversity, *221*, 238
st.plot.ggt, 239
st.plot.ggt.src, 241
st.read.map, 242
st.read.map.df, 243
st.read.marker.data, 244
st.read.marker.data.df, 245
st.read.performance.data, *217*, 246
st.recode.hbc, 247
st.recode.hil, 248
st.recode.ref, 249
st.recode.ref.2, 250
st.reset, 251
st.restrict.marker.data, *221*, 251
st.return.performance.data, 253
st.select.enrichment
(st.calculate.enrichment), 189
st.select.phen, 253
st.set.hblocks, 254
st.set.info.level, 255
st.set.matrix.ops, 255
st.set.num.threads, 256
st.set.openblas.threads, 256
st.set.sim.ef, 257
st.set.sim.gp, 259
st.set.sim.mp, 259
st.set.sim.pp, 260
st.set.simpop, 261
st.start.timer, 262
st.stop.timer, 263
st.STvcf.to.dataframe, *219*, 263
st.switch.error, 264
st.write.map, 266
st.write.marker.data, 267
st_mixed, 268
summarize.gvalue, 269
swap.population.name, 270

talk.to.me, 270

v-tropmaize-map, 271
v-tropmaize-phe, 271
v-tropmaize-pop, 272
v-tropmaize-vcf, 273
v.tropmaize.map *(v-tropmaize-map)*, 271
v.tropmaize.phe *(v-tropmaize-phe)*, 271
v.tropmaize.pop *(v-tropmaize-pop)*, 272
v.tropmaize.vcf *(v-tropmaize-vcf)*, 273

write.vcf, 274
write.version.2, 275